2015

Advanced Error Control Scheme for Noncoherent Random Linear Network Coding

Hongmei Xie
Lehigh University

ADVANCED ERROR CONTROL SCHEME FOR NONCOHERENT RANDOM LINEAR NETWORK CODING

by

Hongmei Xie

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Lehigh University

January 2015

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Prof. Zhiyuan Yan
(Dissertation Advisor)

_____
Accepted Date

Committee Members:

_____
Prof. Zhiyuan Yan
(Committee Chair)

_____
Prof. Tiffany Jing Li

_____
Prof. Bruce A. Dodson

_____
Dr. Bruce W. Suter
Air Force Research Lab.

Acknowledgments

My gratitude goes first to my respectful advisor, Prof. Yan, for his guidance and support through my PhD life. Pursuing a PhD degree is a hard and long process, and it won't be possible for me to achieve that without Prof. Yan's dedicated effort in every aspect, from his expertise in academy to his constructive advice in career planing. I'm really grateful to have Prof Yan both as an advisor and a mentor to start my life at Lehigh, to dig research topics over the years, and to formalize this dissertation.

Thank my honorable committee members, Prof. Tiffany J. Li, Prof. Bruce A. Dodson, and Dr. Bruce W. Suter, for serving in my committee. Prof. Li's broad knowledge and enthusiasm in research inspired and encouraged me to better perform my work. It's an honor to collaborate with Dr. Suter on a couple of research topics, and I certainly gained a lot from his professional work style and abundant experience. Though I haven't had the opportunity to work directly with Prof. Dodson, I admire his expertise in algebra and cryptography. Dear committee members, please accept my sincere acknowledgment!

I shall thank Prof. Baoming Bai, Prof. Xinmei Wang, and Prof. Ying Li of Xidian University for their inspiring teaching and research guidance, from which

I gained beneficial experience in conducting research and projects. The Channel Coding Lab would always be proud of its free research atmosphere and mutual support and learning among its members. Thank Prof. Weihua Zong of Qingdao University, who taught me the first lesson in research, if I have done any, for my bachelor's thesis. I was impressed by her dedication to research as well as little interest in other people's pursuits.

I'm glad to have spent my PhD years together with a bunch of warm hearted friends at Lehigh: Xuebin Wu, Feng Shi, Lihua Jiao, Chenrong Xiong, Yan Yang, Chen Chen, Yang Liu, Jun Lin, Xingjian Zhang, and so many others. I shall thank them for helping me and having fun together. Thank Ahbishek Mishra, who supported me through hard times.

Finally, I'd love to express my deepest appreciation for my family members' devotion. My dearest parents always encourage me to step out to the unknown, and back me up no matter what happens. They have sacrificed so much for me to carry out my happy life, hiding their own dreams behind mine. They, together with my dear brother and sisters, are the source of my courage and happiness. Thank you, and I'm so proud of you!

Contents

List of Tables

List of Figures

Abstract

Random linear network coding (RLNC) has shown advantages in improved throughput, robustness, and reduced delay over traditional routing in a communication network. However, the underlying finite field has to be large enough for RLNC to work effectively, leading to high computational complexity. This dissertation proposes efficient decoding algorithms for RLNC with and without error control schemes, as well as new error control code constructions for a particular realization of RLNC, the coding for distributed storage systems (DSSs).

In RLNC, neither the source nor the sink node has knowledge of the channel transfer characteristic. To deal with errors and erasures in this scenario, subspace codes have been proposed in the literature, including Kötter and Kschischang (KK) codes, lifting of Gabidulin code, and Mahdavifar and Vardy (MV) codes. All these codes can be constructed from evaluations of linearized polynomials. Hence we propose a general interpolation algorithm over linearized polynomials ring, and decode all the three families of codes efficiently. For Gabidulin code, our general interpolation algorithm is deterministic compared to another current decoding algorithm, i.e., it is always be able to produce the correct decoding result when errors are within the error correction capability. For KK codes and MV codes, our algorithm

has lower complexity than solving linear equations, especially for MV codes with large list sizes.

For RLNC without error control technique, rank deficient decoding (RDD) has been proposed to decode the package at the receiver, which transforms the decoding problem into that of a linear block code. To implement RDD efficiently, we first adopt an existing linear programming (LP) approach to accommodate equations with both even and odd parities over the binary field GF(2). Then, we propose a simplified LP algorithm for codes over extension fields of GF(2), and provide some simulation results to show that less packages are required at the receiver to get a same rate of correctly decoded packets.

The data repair and reconstruction problems in distributed storage systems (DSS) were shown to be a multicast problem, thus can be solved by RLNC. Effort has been devoted to explicit code constructions for different optimization goals. We view the DSS coding from a vector space's perspective, and transform data reconstruction and repair requirements into intersection properties of certain subspaces. Three code constructions are proposed for DSSs under the same vector space structure, aiming at low repair complexity, minimized repair bandwidth, and maximized minimum distance given a repair locality, respectively.

Chapter 1

Introduction

Communication systems suffer from signal distortions caused by the underlying channel noise, hence error control is critical in providing reliable communications between two ends of a system. Different error correction codes have been proposed to tackle various kinds of channel noise, such as Reed Solomon codes in satellite communications, Low-Density Parity-Check codes in hard disk and flash memory channels, Turbo codes in 3G and WiMAX wireless communication networks, and so on.

On the other hand, error control technique also brings redundancy in the message to be sent over the channel, as well as extra latency and computational complexity at both the encoder and decoder sides. Hence to design a good error control scheme, multiple factors such as error correction capability, encoding and decoding complexity, and so on have to be taken into consideration.

Recently, following increased storage and exchange of data over all kinds of networks, such as the computer and social networks, wireless networks, cloud storage,

to name a few, how to improve the throughput of a network poses a great challenge. To solve this problem, Ahlswede *et al.* proposed network coding to provide improved throughput, robustness, and reduced delay over traditional routing [2]. In particular, random linear network coding (RLNC) [3] is an effective realization that requires no knowledge of the network at both the sender and receiver ends. However, for RLNC to work effectively, the underlying field has to be very large, leading to high computational complexity and delay at the receiver.

Another problem coming with RLNC is the error and erasure from malicious attacks and packet loss during transmission, as the random nature of generated packets in the intermediate nodes makes it hard to tell which packets are erroneous. Meanwhile, RLNC does not assume any channel information at either the sender or the receiver, hence traditional error correction codes such as the afore mentioned ones do not apply in this scenario. To introduce error control mechanism into RLNC, subspace codes are proposed in the literature. These codes are also constructed over large field size, while existing decoding algorithms either are not efficient or cannot guarantee correct decoding results.

One important application of RLNC is coding problem for a distributed storage system (DSS), where storage nodes such as servers or disks are physically distributed over the network. To gather information from the network, certain subsets of the storage nodes should be able to reconstruct the original message, or recover partial message stored in one node when it fails or leaves the network. Though it has been show that these problems can be modeled as multicast problems [4], hence can be solved by RLNC, explicit code constructions need to be explored in a specific manner for optimization of different metrics.

This dissertation investigates error control schemes for RLNC, including a general decoding algorithm of subspace codes, linear programming algorithms to decode RLNC efficiently, and code constructions for DSS with desirable repair features, such as low computational complexity, reduced bandwidth consumption, and small I/O overhead. We first give in Sec. 1.1 some background on the specific topics we will discuss, an then present the major contributions of our work, as well as the organization of this dissertation in Sec. 1.2.

1.1 Background

In traditional routing technique for a communication network, intermediate nodes do not perform extra process of their received packets but simply forward them along the directed links. As a result, the throughput could be limited by some bottleneck nodes, which are connected with fewer links while contribute to minimum cuts of the network, according to the max-flow min-cut theorem. To solve this problem, network coding is invented in [2] that allows intermediate nodes to "mix" the packets they receive by some combinations over certain finite field, and then transmit them to the next connected nodes. Latter, it is shown that linear combinations [5] are sufficient to achieve the maximized throughput. Furthermore, random linear network coding (RLNC) is proposed in [3], where random linear combinations are used to form the mixed packets.

RLNC has been shown to improve the throughput, robustness, security *etc.* of the underlying network under the error-free assumption [2] [3]. It also reduces the overhead of the network as no record of the linear combination coefficients is

necessary to be stored in the packet header. However, for RLNC to approach the maximal throughput with probability of 1, the underlying field has to be large enough for the linear combinations seen at the receive to be of full rank. As a result, the computational complexity is pretty high for the decoder. Also, to further handle error and erasures over the network, complex error control schemes have to be employed. Hence efficient decoding algorithms are necessary for RLNC with and without error control schemes.

1.1.1 Error Control Schemes in RLNC

In network coding settings, errors occur from unreliable links, wiretappers or malicious nodes. If unchecked, errors greatly deteriorate the throughput gains of network coding and seriously undermine both reliability and security of data. To address these problems, error correction for network coding has been investigated in the literature. Network error correcting codes was first introduced in [6], and generalizations of fundamental bounds in classical algebraic coding theory, such as the Hamming bound and the Gilbert-Varshamov bound were derived [6] [7] [8]. However, all the work about error correction in network coding assumes *coherent* network, i.e., the network topology and network code used is known at the sink node. However, in scenarios such as random linear network coding (RLNC) [3], arguably the most important class of network coding, a *noncoherent* model serves better to describe the changing network conditions.

In noncoherent RLNC, neither the source nor the sink node has knowledge of the channel transfer characteristic. Error control in this scenario utilized the vector space preserving property of the network, where subspaces were transmitted and

received over an operator channel [9]. Over the operator channel, errors and erasures are defined to be additions and deletions of dimensions of the transmitted subspace. The set of subspaces form a subspace code, which is able to accommodate errors and erasures over the operator channel.

A particular family of subspace codes with constant dimensions was proposed in [9], referred to as KK codes, as well as a Sudan-style list-1 minimum-distance decoding algorithm. This list decoding algorithm has a list size of one, and hence it is essentially a bounded distance decoder with a decoding radius approximately half the minimum distance. The work was extended in [10] so that list decoding with arbitrary list sizes was enabled, and we call this family of subspace codes MV codes. Subspace codes can also be obtained from lifting of rank metric codes [11], say Gabidulin codes [12]. All these three families of codes can be obtained via evaluation of linearized polynomials, just as Reed-Solomon (RS) codes can be constructed from evaluation of polynomials. Similarly, the decoding of these codes is composed of an interpolation step and a factorization step. In particular, the interpolation is by linearized polynomials, and a high decoding complexity is induced if conducted by solving linear equations, especially for large list sizes.

In this dissertation, we will propose an efficient interpolation algorithm that works for all the three families of codes, and show its advantages over existing algorithms.

1.1.2 Efficient Decoding of RLNC

Due to its promise of significant throughput gains as well as other advantages, network coding is already used or considered for a wide variety of wired and wireless

networks (see, for example, [13–17]). One significant drawback of network coding is that a full rank of received packets at the receiver nodes of a multicast (or a unicast) is needed before decoding can start, leading to long delays and low throughputs, especially when the number of packets of a session is large. This is particularly undesirable for applications with stringent delay requirements.

Aiming to solve this problem, Yan *et al.* [18] propose rank deficient decoding (RDD) for linear network coding, which can start even when the received packets are not full rank. By reformulating the decoding problem of network coding in a different fashion, the decoding problem reduces to a collection of syndrome decoding problems. In particular, the Hamming norm (HN) decoders from traditional linear block codes can be adopted to implement the syndrome decoding, which take advantage of the sparsity inherent in data and produce the data vectors with the smallest Hamming weight. However, the HN decoders have high complexities for large size systems, hence efficient decoding algorithms are necessary to make the best of RDD decoders. In this dissertation, we will employ linear programming (LP) algorithm to efficiently implement the RDD decoders.

1.1.3 Coding for Distributed Storage Systems

In distributed storage networks, data are stored in nodes that may be individually unreliable. Hence coding is used to introduce redundancy for improved system reliability against node failures. Two types of data recovery [4] are of interest: one is the recovery of the entire message file, called data reconstruction, and the other is the repair of partial messages stored in some nodes using supporting nodes, referred to as data repair. The amount of data downloaded to repair one node is called

repair bandwidth. The number of supporting nodes to be connected for the repair of a failed node is called repair locality, which is closely related to I/O overhead of the system.

Data reconstruction from a subset of nodes is equivalent to erasure decoding in traditional error correction codes, hence maximum distance separable (MDS) codes can be used to maximize protection against erasures [19]. However, the data repair problem is a new challenge for DSS coding, as failed servers or disks are not uncommon, or some nodes may leave or join the system dynamically. Hence new code constructions should be considered to accommodate practical data repair, with features like low computational complexity, reduced bandwidth, and small locality.

Currently, there exist code constructions aiming at different metrics, such as repair bandwidth, maximized minimum distance given a repair locality. However, those constructions are scattered in the sense that different methods are used for different optimization goals. In this dissertation, we tackle the DSS coding problem from a new perspective of a vector space, and transform the data reconstruction and repair requirements into desired properties of subsets of subspaces. Code constructions under different optimization metrics are presented under the uniform structure of a vector space.

1.2 Contributions and Organization

Main contributions of this dissertation are listed as follows, along with the organization of the following chapters.

- In Chapter 2, we tackle the decoding problem of subspaces codes used in

RLNC error control, namely, Gabidulin, KK, and MV codes. Based on the common fact that all of them can be constructed from evaluation of linearized polynomials, we devise a general interpolation algorithm in a free module of the linearized polynomial ring. Analytical results show that our interpolation algorithm has a polynomial time complexity. When used to decode Gabidulin codes, the resulted decoding algorithm resembles Loidreau's decoding algorithm and both algorithms have quadratic complexity, but the two differ in several key aspects. Our general interpolation approach is also used to decode KK codes. In fact, in this case, our algorithm is equivalent to the Sudan-style list-1 decoding algorithm. That is, the Sudan-style list-1 decoding algorithm is a special case of our general interpolation algorithm, when some operations and parameters are specified. Finally, we use our general interpolation algorithm to obtain the multivariate polynomial for the list decoding of MV codes. To the best of our knowledge, there is no other efficient algorithm to accomplish the task. We also show that our algorithm has lower complexity than solving linear equations, both from analytical and hardware implementation results.

- In Chapter 3, we further extend our work on the decoding of MV codes to address some drawbacks of the code construction and their decoding algorithm in [10]. First, no erasures are handled by the current code. To accommodate erasures, we treat the degree of the multivariate linearized polynomial at the interpolation step as a variable, and derive a new decodability condition. We find that errors and erasure are asymmetric in the sense that erasures are more

costly in terms of the code distance. Second, we attempt to expand the decoding radius for high rate codes by defining multiplicity for each interpolation node, as the Guruswami–Sudan algorithm for RS codes [20]. Unfortunately, analytical results show that the decoding radius is slightly reduced due to some undesirable property of linearized polynomials. Finally, after the decoding list is obtained, we form a nearest neighbor decoder, and calculate the decoder error probability (DEP). Assuming no erasures, we obtain an upper bound on the DEP, which decreases exponentially with the list size as well as the dimension of the subspaces in the code. When erasures occur during transmission, a closed-form expression of the DEP is obtained.

- In Chapter 4, we formalize linear programming (LP) algorithms for the RDD decoders based on the work in [18]. To efficiently implement RDD, we adapt LP algorithms to handle both even and odd parities over GF(2), and then propose a simplified LP algorithm for extension fields of GF(2). We prove that our LP algorithm has the desired ML certificate property, and has a much lower complexity than the original formulation in [21]. Simulation results show that our LP decoding algorithm indeed requires less received packets for RLNC to be decoded correctly, while runs faster than other traditional decoding algorithms for linear block codes.

- In Chapter 5, we consider coding problems for DSS. We view the coding for DSS from a new perspective of vector space, where nodes are represented by different subspaces. As a result, data reconstruction and repair requirements are transformed into union and intersection properties among some subset

of subspaces. We present three code constructions under this uniform subspace structure, aiming at different targets. First, we propose a new class of MDS codes with low repair complexity, and then propose a two-layer encoding scheme for optimized repair locality. Finally, we apply the subspace approach to obtain minimum storage regenerating (MSR) codes with optimized repair bandwidth. Though our current construction for MSR codes only works for small parameters, our two-layer construction for LRC codes is a general approach, including some current constructions as special cases. Furthermore, the LRC codes derived from our construction facilitates practical implementation when it comes to degraded reads.

- Conclusions of this dissertation are provided in Chapter 6, as well as some ideas on future work.

Chapter 2

Linearized Polynomial Interpolation and Its Applications

2.1 Introduction

Given a set of points, polynomial interpolation finds one or more polynomials that pass through these points. Since error control codes are often defined through polynomials, polynomial interpolation is instrumental in decoding such codes. For instance, Reed-Solomon (RS) codes can be defined using evaluation of polynomials [22], and bivariate polynomial interpolation has been used in RS decoders. In particular, the Kötter interpolation [23] implements the interpolation step of the Guruswami-Sudan algorithm [20] for RS codes with low complexity. Also, the Welch-Berlekamp key equation can be viewed as a rational interpolation problem, and the Welch-Berlekamp algorithm (WBA) solves this problem [24].

Polynomial interpolation was extended by Wang *et al.* [25] to interpolation in

a free module that is defined over a polynomial ring over some finite field $\mathbb{F}$ and admits an ordering. Since the free module is also a vector space over $\mathbb{F}$, given any set of linear functionals, the interpolation problem is to find a minimum element in the intersection of the kernels of the linear functionals. Wang *et al.* proposed an interpolation algorithm, and showed that the Kötter interpolation and the WBA are both special cases of this general interpolation algorithm [25].

Recently, error control codes defined using evaluation of **linearized polynomials**, such as Gabidulin codes [12] and a family of subspace codes proposed by Kötter and Kschischang (referred to as KK codes) [9], have attracted growing attention. While both Gabidulin and KK codes are important to error control in random linear network coding (see, for example, [9, 11, 26]), Gabidulin codes are also considered for potential applications in wireless communications [27], public-key cryptosystems [28], and storage systems [22, 29]. A decoding algorithm of Gabidulin codes through linearized polynomial reconstruction was proposed by Loidreau [30], and Kötter and Kschischang proposed a Sudan-style list-1 decoding algorithm for KK codes based on bivariate linearized polynomial interpolation [9]. Mahdavifar and Vardy [10] proposed a new class of codes (referred to as MV codes henceforth) and list decoding of MV codes with arbitrary list size.

Parallel to the work of Wang *et al.* [25], we investigate interpolation in a free module of a **linearized polynomial ring**. The main contributions of this chapter are listed as follows:

- We propose a polynomial complexity interpolation algorithm in a well ordered free module of a linearized polynomial ring.
- We apply our interpolation algorithm to decode Gabidulin codes. The resulted

decoding algorithm and Loidreau's decoding algorithm (cf. [30, Table 1]) both have quadratic complexity, but the two differ in their update rules for zero discrepancies, and Loidreau's algorithm malfunctions for a particular discrepancy pattern.

- Our interpolation approach is also used to decode KK codes. In fact, the Sudan-style list-1 decoding algorithm in [9] is a special case of our interpolation algorithm.

- We use our interpolation algorithm to obtain the multivariate polynomial for the list decoding of MV codes in [10]. Although Gaussian elimination can be used for the list decoding of MV codes, our algorithm has a lower complexity.

- Finally, an efficient implementation of an interpolator for decoding MV codes has been proposed. The synthesis results show that it has advantages in throughput and efficiency than Gaussian elimination.

In this chapter, Section 2.2 reviews interpolation over free modules of polynomial rings, and then introduces Gabidulin, KK and MV codes, as well as their respective decoding algorithms. In Section 2.3, we propose our interpolation algorithm over a free module of a linearized polynomial ring, and analyze its computational complexity. We apply our interpolation algorithm to decode Gabidulin, KK, and MV codes in Sections 2.4, 2.5, and 2.6, respectively. In Section 2.7, an interpolator for list decoding of l-dimensional MV codes is implemented in hardware and is compared with Gaussian elimination. The concluding remarks are provided in Section 2.8.

2.2 Preliminaries

2.2.1 General Polynomial Interpolation over Polynomials Ring

Let $\mathbb{F}[x]$ be the ring of all the polynomials over some finite field $\mathbb{F}$ and V be a free $\mathbb{F}[x]$-module. Motivated by the Kötter interpolation, Wang *et al.* [25] consider interpolation in V. Since V is a vector space over $\mathbb{F}$ with some basis M, one can define a set of C (a positive integer) linear functionals D_i's from V to $\mathbb{F}$, with kernels K_i's, where $i = 1, 2, \ldots, C$. If there is a total ordering on M, V admits an ordering. That is, for a subset of V we can find an element with the smallest order, and the element is a *minimum* in this subset. The general interpolation algorithm in [25] finds a minimum in $K_1 \cap K_2 \cap \cdots \cap K_C$.

2.2.2 Linearized Polynomial Ring

Suppose $\mathrm{GF}(q^m)$ is an extension field of $\mathrm{GF}(q)$, where q is a prime power and m is a positive integer. A polynomial of the form $l(x) = \sum_{i=0}^{n} a_i x^{q^i}$ with coefficients $a_i \in \mathrm{GF}(q^m)$ is called a *linearized polynomial* over $\mathrm{GF}(q^m)$. We assume q is fixed, and denote x^{q^i} as $x^{[i]}$ in this chapter. For a linearized polynomial $l(x) = \sum_{i=0}^{n} a_i x^{[i]}$ over $\mathrm{GF}(q^m)$, its q-degree, denoted as $\deg_q(l(x))$, is given by $\max_{a_i \neq 0} \{i\}$.

Consider the set of linearized polynomials over $\mathrm{GF}(q^m)$, denoted by $\mathbb{L}[x]$. Linearized polynomials are so named because for a linearized polynomial $l(x)$ over $\mathrm{GF}(q^m)$, β_1 and β_2 in an extension field $\mathbb{K}$ of $\mathrm{GF}(q^m)$, and $\lambda_1, \lambda_2 \in \mathrm{GF}(q)$, we have $l(\lambda_1\beta_1 + \lambda_2\beta_2) = \lambda_1 l(\beta_1) + \lambda_2 l(\beta_2)$. In other words, $l(x)$ can be treated as a

linear mapping from $\beta \in \mathbb{K}$ to $l(\beta) \in \mathbb{K}$ with respect to GF(q) [31]. Given two linearized polynomials $l_1(x)$ and $l_2(x)$ over GF(q^m), their GF(q^m)-linear combination $\alpha_1 l_1(x) + \alpha_2 l_2(x)$ with $\alpha_1, \alpha_2 \in \text{GF}(q^m)$, is also a linearized polynomial over GF(q^m). We define the multiplication between $l_1(x)$ and $l_2(x)$ as $l_1(x) \otimes l_2(x) \stackrel{\text{def}}{=} l_1(l_2(x))$, and $l(x) = l_1(x) \otimes l_2(x)$ is also a linearized polynomial over GF(q^m). Since $l_1(x) \otimes l_2(x)$ does not necessarily equal $l_2(x) \otimes l_1(x)$, $\mathbb{L}[x]$ with polynomial addition and the multiplication $\otimes$ forms a noncommutative ring. Note that there is no left or right divisor of zero in $\mathbb{L}[x]$ [32].

2.2.3 Gabidulin Codes and Loidreau's Reconstruction Algorithm

The rank of a vector $\mathbf{x} \in \text{GF}(q^m)^n$ is the **maximal** number of coordinates that are linearly independent over GF(q), denoted as $r(\mathbf{x}; q)$. The rank distance between two vectors $\mathbf{x}, \mathbf{y} \in \text{GF}(q^m)^n$ is defined to be $d_r(\mathbf{x}, \mathbf{y}) = r(\mathbf{x} - \mathbf{y}; q)$. It is shown in [12] that the rank distance is a metric on a vector space, and one can consider the rank distance properties of a linear block code $\mathcal{C}$. The minimum rank distance of $\mathcal{C}$, denoted as $d_r(\mathcal{C})$, is simply the minimum rank distance over all possible pairs of distinct codewords, that is, $d_r(\mathcal{C}) = \min_{\mathbf{x}_i \neq \mathbf{x}_j \in \mathcal{C}} d_r(\mathbf{x}_i, \mathbf{x}_j)$.

The maximum cardinality of a rank metric code in GF(q^m)^n with minimum rank distance d is $\min\{q^{m(n-d+1)}, q^{n(m-d+1)}\}$ [12,33,34]. We refer to codes with maximum cardinality as maximum rank distance (MRD) codes. A family of linear MRD codes was proposed by Gabidulin [12], and is often referred to as Gabidulin codes. An

(n, k) Gabidulin code $\mathcal{C}_{\mathcal{R}}$ over GF(q^m) $(n \leq m)$ is defined by a generator matrix

$$\mathbf{G} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-1} \\ g_0^{[1]} & g_1^{[1]} & \cdots & g_{n-1}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ g_0^{[k-1]} & g_1^{[k-1]} & \cdots & g_{n-1}^{[k-1]} \end{pmatrix}, \tag{2.1}$$

where $g_0, g_1, \ldots, g_{n-1}$ are linearly independent over GF(q). We introduce the vector $\mathbf{g} = (g_0, g_1, \ldots, g_{n-1})$ for future reference, called the *generator vector*. For a message vector $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ and its corresponding message polynomial $f(x) = \sum_{i=0}^{k-1} u_i x^{[i]}$, the codeword to be transmitted is $\mathbf{x} = (f(g_0), f(g_1), \ldots, f(g_{n-1}))$. Suppose an additive error $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ occurs, and the received vector is $\mathbf{y} = \mathbf{x} + \mathbf{e} = (y_0, y_1, \ldots, y_{n-1})$, where $y_i = x_i + e_i$ for $i = 0, 1, \ldots, n-1$. Given $\mathbf{y}$, a bounded distance decoder with decoding radius $t \leq (n-k)/2$ tries to find $\mathbf{x}' \in \mathcal{C}_{\mathcal{R}}$ and $\mathbf{e}' \in \text{GF}(q^m)^n$ such that $\mathbf{y} = \mathbf{x}' + \mathbf{e}'$ with $d_r(\mathbf{y}, \mathbf{x}') \leq t$. If such $\mathbf{x}'$ and $\mathbf{e}'$ exist, the received vector $\mathbf{y}$ is said to be decodable [12].

Gabidulin codes can be defined using evaluation of linearized polynomials, analogous to RS codes, which are defined using evaluation of polynomials. Hence Loidreau devised a method to decode Gabidulin codes through *reconstruction of linearized polynomials* (cf. [30, Table 1]), where a pair of linearized polynomials, $V(y)$ and $N(x)$, are constructed such that $V(y_i) = N(g_i)$ for $i = 0, 1, \ldots, n-1$, with $\deg_q(V(y)) \leq t$ and $\deg_q(N(x)) \leq k + t - 1$. It was shown in [30] that if $t \leq (n-k)/2$, one gets a solution of decoding Gabidulin codes from any solution of the reconstruction problem. Loidreau's algorithm [30] constructs two pairs of polynomials $(V_0(y), N_0(x))$ and $(V_1(y), N_1(x))$, and updates them iteratively by

discrepancy-based update rules, so that each pair satisfies the constraints defined by the first i points after the ith iteration. To implement the degree constraints on the linearized polynomials, Loidreau's algorithm starts with initial polynomials of designated q-degrees, and then aims to increase the q-degrees of each pair of polynomials strictly once every two iterations. The algorithm outputs $N_1(x)$ with q-degree no more than $k + \lfloor (n-k)/2 \rfloor - 1$ and $V_1(y)$ of q-degree no more than $\lfloor (n-k)/2 \rfloor$.

2.2.4 KK Codes and Their Decoding Algorithm

KK codes [9] are a type of subspace codes for random linear network coding, where subspaces are transmitted and received at both ends. Suppose W is a vector space over GF(q), and $\mathcal{P}(W)$ is the set of all subspaces of W. For $U, V \in \mathcal{P}(W)$, the subspace distance d_s [9] between V and U is defined as $d_s(V,U) \stackrel{\text{def}}{=} \dim(V+U) - \dim(V \cap U)$, where $\dim(A)$ denotes the dimension of a subspace $A \in \mathcal{P}(W)$, $V \cap U$ is the intersection space of V and U, and $V + U$ is the smallest subspace that contains both V and U.

Suppose an l-dimensional subspace $V \in \mathcal{P}(W)$ is a codeword of a KK code. The basis of V is obtained via evaluation of linearized polynomials. First we select l ($l \leq m$) elements $\alpha_0, \alpha_1, \ldots, \alpha_{l-1} \in \text{GF}(q^m)$ that are linearly independent over GF(q). Theses l elements span an l-dimensional vector space $\langle A \rangle \subseteq \text{GF}(q^m)$, where $A = \{\alpha_i : i = 0, 1, \ldots, l-1\}$. We then construct W by $W = \langle A \rangle \oplus \text{GF}(q^m) = \{(\alpha, \beta) : \alpha \in \langle A \rangle, \beta \in \text{GF}(q^m)\}$. Given a message vector $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ over GF(q^m), the message polynomial is defined to be $u(x) = \sum_{i=0}^{k-1} u_i x^{[i]}$. Finally, the subspace spanned by $\{(\alpha_i, \beta_i) : \beta_i = u(\alpha_i), i = 0, 1, \ldots, l-1\}$ is an l-dimensional subspace of W, as all the pairs (α_i, β_i) are linearly independent [9].

Suppose V is transmitted over the operator channel [9], and an $(l-\rho+t)$-dimensional subspace U of W is received, with $\dim(U \cap V) = l - \rho$ and $d_s(U, V) = \rho + t$. It is proved that the error is decodable by the list-1 decoding algorithm [9] if $\rho + t < l - k + 1$. Let $l - \rho + t = r$, and $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{r-1}, y_{r-1})\}$ be a basis for U. The decoding algorithm in [9] consists of an interpolation step and a factorization step. First the interpolation procedure finds a nonzero bivariate polynomial $Q(x, y) = Q_x(x) + Q_y(y)$ such that

$$Q(x_i, y_i) = 0 \text{ for } i = 0, 1, \ldots, r-1, \tag{2.2}$$

where $Q_x(x)$ and $Q_y(y)$ are linearized polynomials of q-degrees at most $\tau - 1$ and $\tau - k$ respectively. Then a message polynomial $\hat{u}(x)$ is obtained in the factorization step by right division [9] if $Q(x, \hat{u}(x)) \equiv 0$. Decodability is guaranteed if we select $\tau = \lceil (r+k)/2 \rceil$ [9].

The interpolation procedure of the decoding algorithm in [9], called a Sudan-style list-1 decoding algorithm, adopts some discrepancy based update rules. During the i-th iteration, the algorithm generates an x-minimal bivariate polynomial and a y-minimal bivariate polynomial, $f_0^{(i)}(x, y)$ and $f_1^{(i)}(x, y)$, that interpolate through the first i points for $i = 1, 2, \ldots, r$, where r is the total number of points to be interpolated. Finally, the minimum one between $f_0^{(r)}(x, y)$ and $f_1^{(r)}(x, y)$, defined under an order of $\prec$ [9], is the decoding output.

2.2.5 MV Codes and Their List Decoding Algorithm

MV codes are similar to but different from KK codes [9]. To enable list decoding, different code constructions are proposed for different code dimensions in [10].

To construct an l-dimensional MV code over GF(q^{ml}), l has to be a positive integer that divides $q-1$. Then the equation $x^l - 1 = 0$ has l distinct roots $e_1 = 1, e_2, \ldots, e_l$ over GF(q). Choose a primitive element γ over GF(q^{ml}) with $\gamma, \gamma^{[1]}, \ldots, \gamma^{[ml-1]}$ being a normal basis for GF(q^{ml}). Then construct elements α_i over GF(q^{ml}) by $\alpha_i = \gamma + e_i\gamma^{[m]} + e_i^2\gamma^{[2m]} + \cdots + e_i^{l-1}\gamma^{[m(l-1)]}$ for $i = 1, 2, \ldots, l$. It is shown in [10] that the set $\{\alpha_i^{[j]} : i = 1, 2, \ldots, l, j = 0, 1, \ldots, m-1\}$ is a basis of GF(q^{ml}) over GF(q).

For a message vector $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ over GF(q), the message polynomial is $u(x) = \sum_{i=0}^{k-1} u_i x^{[i]}$. Let $u^{\otimes i}(x)$ denote the composition of $u(x)$ with itself by i times for any nonnegative integer i, while $u^{\otimes 0}(x) = x$. Then the codeword V corresponding to the message $\mathbf{u}$ is spanned by a set of vectors v_i for $i = 1, 2, \ldots, l$, where $v_1 = (\alpha_1, u(\alpha_1), u^{\otimes 2}(\alpha_1), \ldots, u^{\otimes L}(\alpha_1))$, $v_i = (\alpha_i, \frac{u(\alpha_i)}{\alpha_i}, \ldots, \frac{u^{\otimes L}(\alpha_i)}{\alpha_i})$, and L is the desired list size. Note that $\frac{u^{\otimes j}(\alpha_i)}{\alpha_i} \in$ GF(q^m) for any $j \geq 0$ and $i = 2, 3, \ldots, l$ [10]. Then V is an l-dimensional subspace of the $(Lm + l)$-dimensional ambient space $W = \langle \alpha_1, \alpha_2, \ldots, \alpha_l \rangle \oplus \underbrace{\text{GF}(q^m) \oplus \cdots \oplus \text{GF}(q^m)}_{L \text{ times}}$. Suppose an error of dimension t occurs, and an $(l+t)$-dimensional subspace U of W is received. The decoder first finds subspaces U_i such that $U_i = \{(x, y_1, y_2, \ldots, y_L) : x \in \langle \alpha_i \rangle\}$ for $i = 1, 2, \ldots, l$. Then, a basis $\{(x_{1,j}, y_{1,1,j}, y_{1,2,j}, \ldots, y_{1,L,j}) : j = 1, 2, \ldots, r_1\}$ of U_1 is found, where r_1 is the dimension of U_1. If $l = 1$, we ignore the first step and simply find a basis for the $(t+1)$-dimensional received subspace U_1. For $i = 2, 3, \ldots, l$, the decoder obtains $U_i' = \{(x, \alpha_i y_1, \alpha_i y_2, \ldots, \alpha_i y_L) : (x, y_1, y_2, \ldots, y_L) \in U_i\}$, and

finds a basis $\{(x_{i,j}, y_{i,1,j}, y_{i,2,j}, \ldots, y_{i,L,j}) : j = 1, 2, \ldots, r_i\}$ of U_i', where r_i is the dimension of U_i. Finally, the decoder constructs a nonzero multivariate polynomial $Q(x, y_1, y_2, \ldots, y_L) = Q_0(x)+Q_1(y_1)+Q_2(y_2)+\cdots+Q_L(y_L)$, where Q_s is a linearized polynomials over GF(q^{ml}) of q-degree at most $ml - s(k-1) - 1$ for $s = 0, 1, \ldots, L$, such that for $i = 1, 2, \ldots, l$, $j = 1, 2, \ldots, r_i$, and $h = 0, 1, \ldots, m-1$,

$$Q(x_{i,j}^{[h]}, y_{i,1,j}^{[h]}, \ldots, y_{i,L,j}^{[h]}) = 0. \tag{2.3}$$

The decoder then finds all possible polynomials $\hat{u}(x)$'s, using an LRR algorithm in [10], such that $Q(x, \hat{u}(x), \hat{u}^{\otimes 2}(x), \ldots, \hat{u}^{\otimes L}(x)) \equiv 0$. It is shown in [10] that (2.3) has a nonzero solution if $t < lL - L(L+1)\frac{k-1}{2m}$, and there are at most L solutions, among which the transmitted message polynomial $u(x)$ is guaranteed to be included.

2.3 Interpolation by Linearized Polynomials

In this section, we investigate the interpolation problem by linearized polynomials. We first present the interpolation problem, then propose our interpolation algorithm, which follows a strategy similar to that in [25].

2.3.1 Interpolation over Free $\mathbb{L}[x]$-Modules

Suppose $\mathbb{L}[x]$ is the ring of linearized polynomials over GF(q^m), and V is a free $\mathbb{L}[x]$-module with a basis $B = \{b_0, b_1, \ldots, b_L\}$. We denote the multiplication between an element in $\mathbb{L}[x]$ and an element in the module by $\circ$, and any element $Q \in V$ can be represented by $Q = \sum_{j=0}^{L} l_j(x) \circ b_j = \sum_{j=0}^{L} \sum_{i \geq 0} a_{i,j} x^{[i]} \circ b_j$, where $l_j(x) \in \mathbb{L}[x]$ and

$a_{i,j} \in \mathrm{GF}(q^m)$. Thus V is also a vector space over $\mathrm{GF}(q^m)$ with a basis

$$M = \{x^{[i]} \circ b_j, i \geq 0, j = 0, 1, \ldots, L\}. \tag{2.4}$$

Suppose there exists a total ordering $<$ on M that satisfies 1) $x^{[i]} \circ b_j < x^{[i']} \circ b_j$ if $i < i'$, and 2) we can write $M = \{\phi_j\}_{j \geq 0}$ with $\phi_i < \phi_j$ when $i < j$. Then $Q \in V$ can be represented by $Q = \sum_{j=0}^{J} a_j \phi_j$, where $\phi_j \in M$ and $a_J \neq 0$. J is called the *order* of Q, denoted as $\mathrm{order}(Q)$, ϕ_J is the *leading monomial* of Q, denoted as $\mathrm{LM}(Q)$, and $a_j \phi_j$ is the *leading term* of Q. We write $Q <_o Q'$ if $\mathrm{order}(Q) < \mathrm{order}(Q')$, and $Q =_o Q'$ if $\mathrm{order}(Q) = \mathrm{order}(Q')$. An element Q is a minimum in a subset of V if its order is the lowest among all the elements in the subset. Further, we define $\mathrm{Ind}_y(l(x) \circ b_j) = j$ and $\mathrm{Ind}_y(Q) = \mathrm{Ind}_y(\mathrm{LM}(Q))$, and then introduce a partition of V: $V = \bigcup_j S_j$, where $S_j = \{Q \in V : \mathrm{Ind}_y(Q) = j\}$.

Suppose C is a positive integer. For the vector space V over $\mathrm{GF}(q^m)$, we consider a set of C linear functionals D_i from V to $\mathrm{GF}(q^m)$: $D_1, D_2, \ldots, D_C$. Suppose K_i is the kernel of D_i and $\overline{K}_i = K_1 \cap K_2 \cap \cdots \cap K_i$ is an $\mathbb{L}[x]$-submodule, then the interpolation problem is to find a minimum $Q^* \in \overline{K}_C$, that is, to find an element $Q^* \in V$ such that it lies in the kernels of all the given linear functionals. Furthermore, we can show the uniqueness of Q^* as in [25].

Lemma 1. *The minimum in $\overline{K}_C$ is unique up to a scalar.*

The proof can be found in the appendix.

Define $T_{i,j} = \overline{K}_i \cap S_j$, and $g_{i,j} = \min_{g \in T_{i,j}} g$, then the interpolation problem is equivalent to finding $g_{C,j}$. The key idea is to iteratively construct $g_{i,j}$ from its previous values by a discrepancy based update for $i = 1, \ldots, C$, starting from some

Algorithm 1 Interpolation Algorithm

for $j = 0$ to L **do**
  $g_{0,j} \leftarrow b_j$
end for
for $i = 0$ to $C - 1$ **do**
  for $j = 0$ to L **do**
    $g_{i+1,j} \leftarrow g_{i,j}$
    $\Delta_{i+1,j} \leftarrow D_{i+1}(g_{i,j})$
  end for
  $J \leftarrow \{j : \Delta_{i+1,j} \neq 0\}$
  if $J \neq \emptyset$ **then**
    $j^* \leftarrow \underset{j \in J}{\operatorname{argmin}}\{g_{i,j}\}$
    for $j \in J$ **do**
      if $j \neq j^*$ **then**
        $g_{i+1,j} \leftarrow \Delta_{i+1,j^*} g_{i,j} - \Delta_{i+1,j} g_{i,j^*}$
      else if $j = j^*$ **then**
        $g_{i+1,j} \leftarrow \Delta_{i+1,j}(x^{[1]} \circ g_{i,j}) - D_{i+1}(x^{[1]} \circ g_{i,j}) g_{i,j}$
      end if
    end for
  end if
end for
$Q^* \leftarrow \min_j \ g_{C,j}$

initial values. We propose Algorithm 1 to solve this interpolation problem. In the initialization step of Algorithm 1, $g_{0,j}$ is set to b_j for $j = 0, 1, \ldots, L$ respectively. In the intermediate steps, there are three cases, and in each case a different update is used to obtain $g_{i+1,j}$ based on $g_{i,j}$.

1. If $g_{i,j} \in K_{i+1}$, then $g_{i+1,j} = g_{i,j}$.

2. For $g_{i,j}$'s not in K_{i+1}, we find one of them with the lowest order, denoted as g_{i,j^*}. Then for any $g_{i,j}$ with $j \neq j^*$, we update $g_{i+1,j} = D_{i+1}(g_{i,j^*})g_{i,j} - D_{i+1}(g_{i,j})g_{i,j^*}$. We call this type of update a cross-term update. Note in this case, the order of $g_{i,j}$ is preserved, that is, $g_{i+1,j} =_o g_{i,j}$.

3. For g_{i+1,j^*}, we construct g_{i+1,j^*} by $g_{i+1,j^*} = D_{i+1}(g_{i,j^*})(x^{[1]} \circ g_{i,j^*}) - D_{i+1}(x^{[1]} \circ g_{i,j^*})g_{i,j^*}$. We call this type of update an order-increase update. In this case, g_{i+1,j^*} takes a higher order than g_{i,j^*}, that is, $g_{i,j^*} <_o g_{i+1,j^*}$.

Lemma 2. *In each of the three cases, $g_{i+1,j}$ is a minimum in $T_{i+1,j}$.*

The proof in the appendix follows a similar approach as in [35] and [25].

2.3.2 Complexity Analysis of Algorithm 1

There are a total of C iterations in Algorithm 1. In each iteration, $L + 1$ linear functionals are first carried out to calculate the discrepancies, followed by at most L finite field additions (subtractions) to find the minimum candidate and its index among those with nonzero discrepancies. Then to update the candidates, we conduct at most $2(L + 1)^2(D + 1)$ finite field multiplications, $(L + 1)^2(D + 1)$ finite field additions, one multiplication between elements in the ring $\mathbb{L}[x]$ and elements

in the module V, and one computation of the linear functional, where D is the highest q-degree of the linearized polynomials in x for all iterations. Note that the q-degree of each candidate is non-decreasing in an iteration based on the update rules. Hence it is safe to choose D to be the highest q-degree of the polynomial in x of the ultimate output. To sum up, the complexity of Algorithm 1 is dominated by $O(CDL^2)$ finite field additions, $O(CDL^2)$ field multiplications, $O(CL)$ linear functional calculations, and $O(C)$ multiplications between elements in the ring $\mathbb{L}[x]$ and elements in the module V. Since the complexity of the linear functional calculations and the multiplications between elements in the ring and elements in the module might vary in different situations, we consider the complexity of each realization of Algorithm 1 on a case-by-case basis.

2.4 Decoding of Gabidulin Codes

2.4.1 Decoding of Gabidulin Codes

We consider an (n, k) Gabidulin code over $\mathrm{GF}(q^m)$ $(n \leq m)$ as defined in Section 2.2.3, and the ring of linearized polynomials $\mathbb{L}[x]$ over $\mathrm{GF}(q^m)$ discussed in Section 2.2.2. Based on Loidreau's polynomial reconstruction approach [30], we consider the decoding problem of Gabidulin codes from an interpolation point of view. Suppose we have a set of points (x_i, y_i) with $y_i = f(x_i) + e_i$ for $i = 0, 1, \ldots, n-1$, where x_i's are linearly independent and $r(\mathbf{e}; q) \leq t$, and try to construct a nonzero bivariate polynomial $Q(x, y) = Q_1(x) + Q_2(y)$ with $Q_1(x)$ and $Q_2(y)$ being linearized polynomials over $\mathrm{GF}(q^m)$, such that $\max\{\deg_q(Q_1(x)), k - 1 + \deg_q(Q_2(y))\}$ is as

small as possible and

$$Q(x_i, y_i) = Q_1(x_i) - Q_2(y_i) = 0 \text{ for } i = 0, 1, \ldots, n-1. \tag{2.5}$$

We will show that a solution of (2.5) gives a solution to the decoding problem of Gabidulin codes under some conditions. Then we formalize (2.5) to an interpolation problem over free $\mathbb{L}[x]$-modules, and solve it by Algorithm 1.

Suppose $\deg_q(Q_1) = \tau + k - 1$, and $\deg_q(Q_2) = \tau$. To have a nonzero solution of (2.5), the number of unknown coefficients should be greater than the number of equations, that is, $2\tau > n - k - 1$. Suppose $Q(x, y) = Q_1(x) - Q_2(y)$ is a nonzero solution of (2.5). Substituting y by $f(x)$, we get $Q(x, f(x)) = Q_1(x) - Q_2(f(x))$. When $Q(x, f(x)) \equiv 0$, i.e., $Q_1(x) - Q_2(f(x))$ is the zero polynomial, $f(x)$ satisfies $Q_1(x) = Q_2(x) \otimes f(x)$ and thus can be obtained by right division over the linearized polynomial ring [9].

It remains to identify the condition under which $Q(x, f(x))$ is identically zero. Since $Q(x, y) = Q_1(x) - Q_2(y)$ is a nonzero solution of (2.5), $Q(x_i, y_i) = Q_1(x_i) - Q_2(y_i) = 0$, i.e., $Q_1(x_i) - Q_2(f(x_i)) = Q_2(e_i)$ with $(Q_2(e_0), Q_2(e_1), \ldots, Q_2(e_{n-1}))$ of rank no more than t. Then there exists a nonzero linearized polynomial W of q-degree at most t such that $W(Q_2(e_i)) = W(Q_2(x_i) - Q_2(f(x_i))) = 0$ for $i = 0, 1, \ldots, n-1$. Then we have a linearized polynomial $W(Q_1(x) - Q_2(f(x)))$ of q-degree at most $t+\tau+k-1$ with n linearly independent roots x_i for $i = 0, 1, \ldots, n-1$. Thus when $t + \tau + k - 1 < n$, we have $W(Q_1(x) - Q_2(f(x))) \equiv 0$. Since there is no left or right divisor or zero in the linearized polynomial ring [32] and W is nonzero, we have $Q_1(x) - Q_2(f(x)) \equiv 0$, hence $f(x)$ can be obtained by right division over

the linearized polynomial ring. The condition $t + \tau + k - 1 < n$ will be satisfied by forcing $t \leq \tau$, and restricting $2\tau < n - k + 1$. Combining these conditions, we select $\tau = \lfloor (n-k)/2 \rfloor$, and have

$$t \leq \lfloor (n-k)/2 \rfloor. \tag{2.6}$$

Hence if (2.6) is satisfied, a solution of (2.5) gives a solution to the decoding problem of Gabidulin codes. Next we formalize the interpolation problem in (2.5) to an interpolation problem over free $\mathbb{L}[x]$-modules.

We select $B = \{b_0, b_1\} = \{x, y\}$ as a basis, and construct a free $\mathbb{L}[x]$-module $V = \{Q(x,y)\}$ from $Q(x,y) = l_0(x) \circ b_0 + l_1(x) \circ b_1$, where $l_0(x), l_1(x) \in \mathbb{L}[x]$, and the multiplication $\circ$ is defined as

$$l(x) \circ b_j \stackrel{\text{def}}{=} l(b_j), \text{ for } j = 0, 1. \tag{2.7}$$

Hence $Q(x,y) = l_0(x) + l_1(y)$, and we call such $Q(x,y) \in V$ a *bivariate linearized polynomial*. To ensure that V is well ordered, we define a $(1, k-1)$-weighted degree for any $x^{[i]} \circ b_j \in M$ to be $\deg_{1,k-1}(x^{[i]} \circ b_j) \stackrel{\text{def}}{=} i + j*(k-1)$ for $i \geq 0$, $j \in \{0,1\}$, and a positive integer k. A total ordering on M is established by writing $x^{[i]} \circ b_j < x^{[i']} \circ b_{j'}$ if $\deg_{1,k-1}(x^{[i]} \circ b_j) < \deg_{1,k-1}(x^{[i']} \circ b_{j'})$, or if $\deg_{1,k-1}(x^{[i]} \circ b_j) = \deg_{1,k-1}(x^{[i']} \circ b_{j'})$ and $j < j'$, for any $i, i' \geq 0$ and $j, j' \in \{0,1\}$. Thus, both conditions on the total ordering of M in Section 2.3 are satisfied, and given a subset of V, a minimum element in V can be found.

Finally, we define a set of linear functionals D_i from V to $\mathrm{GF}(q^m)$ to be $D_i(Q) = Q(x_i, y_i) = l_0(x_i) + l_1(y_i)$ for $i = 0, 1, \ldots, n-1$, where (x_i, y_i)'s are the points to be interpolated. If $D_i(Q(x,y)) = 0$, $Q(x,y)$ is said to be in the kernel K_i of D_i. The

kernels are $\mathbb{L}[x]$-submodules by the following lemma.

Lemma 3. *K_i is an $\mathbb{L}[x]$-submodule.*

The proof is straightforward and hence omitted. Based on Lemma 3, $\overline{K}_i$ is also an $\mathbb{L}[x]$-submodule. Consequently, the interpolation problem in (2.5) is to find a minimum $Q \in V$ such that Q is a minimum in $\overline{K}_{n-1}$, and Algorithm 1 solves it by finding a **minimum** nonzero solution.

To use Algorithm 1, first we set $g_{0,0} = x$, and $g_{0,1} = y$ in the initialization step. In the following iterations, multiplication between an element in $\mathbb{L}[x]$ and an element in V in the cross-term and order-increase updates follow (2.7). In particular, $g_{i+1,j^*} = D_{i+1}(g_{i,j^*})(x^{[1]} \circ g_{i,j^*}) - D_{i+1}(x^{[1]} \circ g_{i,j^*})g_{i,j^*}$. Since $D_{i+1}(g_{i,j^*}) \neq 0$, we omit it from the right hand side, and instead use $g_{i+1,j^*} = g^q_{i,j^*} - (D_{i+1}(g_{i,j^*}))^{q-1}g_{i,j^*}$, as scaling by a nonzero scalar does not affect the order of an element in V.

Table 2.1: Example 1: Use Loidreau's algorithm and Algorithm 1 to decode Gabidulin codes

Iteration	Loidreau's algorithm	Algorithm 1
0	Precomputation	$g_0 = x$ $g_1 = y$
1	Initialization step	$\Delta_0 = \alpha^{31}, \Delta_1 = \alpha^{31}$ $g_0 = x^2 + \alpha^{31}x$ $g_1 = \alpha^{31}x + \alpha^{31}y$
2	$N_0 = x^4 + \alpha^5 x^2 + \alpha^{31}x, V_0 = 0$ $N_1 = \alpha^{48}x^2 + \alpha^{33}x, V_1 = y$	$\Delta_0 = 1, \Delta_1 = \alpha^{16}$ $g_0 = x^4 + \alpha^5 x^2 + \alpha^{31}x$ $g_1 = \alpha^{16}x^2 + \alpha x + \alpha^{31}y$
3	$s_0 = \alpha^7, s_1 = 0$ $N_0 = \alpha^{33}x^4 + \alpha^3 x^2, V_0 = y^2$ $N_1 = \alpha^{55}x^2 + \alpha^{40}x, V_1 = \alpha^7 y$	$\Delta_0 = \alpha^7, \Delta_1 = 0$ $g_0 = x^8 + \alpha^{39}x^4 + \alpha^{34}x^2 + \alpha^{38}x$ $g_1 = \alpha^{16}x^2 + \alpha x + \alpha^{31}y$
4	$s_0 = \alpha^{17}, s_1 = \alpha^{47}$ $N_0 = \alpha^{47}x^4 + \alpha^4 x^2 + \alpha^{24}x, V_0 = \alpha^{14}y^2 + \alpha^{54}y$ $N_1 = \alpha^{17}x^4 + \alpha^{37}x^2 + \alpha^{57}x, V_1 = \alpha^{47}y^2 + \alpha^{24}y$	$\Delta_0 = \alpha^{50}, \Delta_1 = \alpha^8$ $g_0 = \alpha^8 x^8 + \alpha^{47}x^4 + \alpha^{46}x^2 + \alpha^{45}x + \alpha^{18}y$ $g_1 = \alpha^{32}x^4 + \alpha^{52}x^2 + \alpha^9 x + \alpha^{62}y^2 + \alpha^{39}y$
5	$s_0 = \alpha^{31}, s_1 = \alpha$ $N_0 = \alpha^{34}x^8 + \alpha^{37}x^4 + \alpha^{10}x^2 + \alpha^{58}x$ $V_0 = \alpha^{31}y^4 + \alpha^{25}y$ $N_1 = 0, V_1 = 0$	$\Delta_0 = \alpha^{18}, \Delta_1 = \alpha^{16}$ $g_0 = \alpha^{24}x^8 + \alpha^{22}x^4 + \alpha^{47}x^2 + \alpha^{58}x + \alpha^{17}y^2 + \alpha^{49}y$ $g_1 = \alpha x^8 + \alpha^4 x^4 + \alpha^{40}x^2 + \alpha^{25}x + \alpha^{61}y^4 + \alpha^{55}y$
6	$s_0 = \alpha^{16}, s_1 = 0$ $N_1 = 0, V_1 = 0, N_0 = 0, V_0 = 0$	$\Delta_0 = \alpha^6, \Delta_1 = \alpha^{46}$ $g_1 = \alpha^4 x^4 + x^2 + \alpha^{29}x + \alpha^4 y^4 + y^2 + \alpha^{29}y$

2.4.2 Comparison to Loidreau's Reconstruction Algorithm

Although our cross-term and order-increase update rules are similar to that of the alternate increasing degree step in Loidreau's algorithm, we observe that Algorithm 1 differs from Loidreau's algorithm in two key aspects, stated as follow.

First, Loidreau's algorithm uses another algorithm [36] in the precomputation step before initializing the main algorithm, for the purpose of reducing complexity, whereas our decoding algorithm carries out all the iterations solely from the interpolation approach. However, we can show the equivalence of the polynomials derived after the initialization step of Loidreau's algorithm and the ones obtained after the first k iterations of Algorithm 1. The initialization step of Loidreau's algorithm actually introduces two bivariate polynomials $Q_0 = N_0(x) - V_0(y)$ and $Q_1 = N_1(x) - V_1(y)$. Given our previous notations, Algorithm 1 produces two bivariate polynomials $g_{k,0}$ and $g_{k,1}$ after the first k iterations. The relation between these four polynomials are stated in Lemma 4.

Lemma 4. *The initial bivariate polynomials of Loidreau's algorithm and the bivariate polynomials derived after the first k iterations of Algorithm 1 are of the same order correspondingly, i.e., $Q_0 =_o g_{k,0}$ and $Q_1 =_o g_{k,1}$.*

The proof can be found in the appendix. Note that the q-degree of $N_0(x)$ is exactly k, as it actually interpolates over k linearly independent points $x_0, x_1, \ldots, x_{k-1}$. $N_1(x)$ is a linear combination of polynomials of q-degree $k-1$, but its q-degree might be lower than $k-1$, as the most significant coefficients may cancel each other. Thus the claim in [30] that after the final iteration $\deg_q(V_1(y)) = \lfloor (n-k)/2 \rfloor$ is inaccurate.

The second difference between Loidreau's and our decoding algorithms lies in the

update of the interpolation steps when some of the discrepancies are zero. It should be pointed out that in the alternate increasing degree step of Loidreau's algorithm, s_0 in operations (c) and (d) should be $s_0^{(q-1)}$ in ([30, Table 1]). After the correction of this typo, the key difference between Loidreau's algorithm and Algorithm 1 is that the latter accounts for zero discrepancies, while the former only covers it partially. To be specific, Loidreau's algorithm [30, Table 1] malfunctions when $s_1 = 0$ but $s_0 \neq 0$, as shown in Lemma 5.

Lemma 5. *If $s_1 = 0$ but $s_0 \neq 0$ at the beginning of any iteration, all four linearized polynomials of the V_0, N_0, V_1 and N_1 in Loidreau's algorithm will be the zero polynomial after a certain number of iterations.*

The proof can be conducted simply by tedious calculations, hence it is omitted here. Instead, an example is given to illustrate Lemma 5, where $s_1 = 0$ but $s_0 \neq 0$ happens during an intermediate iteration. To solve the problem in Lemma 5, one can update the candidates when the zero discrepancy is involved. However, such an operation breaks the rule of updating the q-degrees of the candidates alternately, which is designed to ensure strict degree constraints on the output of the algorithm. Since s_0 and s_1 are involved in different types of update rules for the two pairs of candidate polynomials, for the case of $s_1 \neq 0$ but $s_0 = 0$, the algorithm in [30, Table 1] works properly.

Example 1. *We construct a $(6, 2)$ Gabidulin code over GF(2^6) with a generator vector $\mathbf{g} = (\alpha^{31}, \alpha^{48}, \alpha^{32}, \alpha^{16}, 1, \alpha^{47})$, where α is a primitive element of GF(2^6) and is a root of $x^6 + x + 1 = 0$. Given the message vector $\mathbf{u} = (1, 0)$, the message polynomial is $f(x) = x$, with a codeword $\mathbf{x} = (f(g_0), f(g_1), \ldots, f(g_{n-1})) = \mathbf{g}$. Suppose*

the error vector is $\mathbf{e} = (0, \alpha^{48}, \alpha^{54}, 0, 0, 0)$*, and the received vector is* $\mathbf{y} = \mathbf{x} + \mathbf{e} = (\alpha^{31}, 0, \alpha^{19}, \alpha^{16}, 1, \alpha^{47})$*. The decoding procedures by Loidreau's algorithm and Algorithm 1 are presented in Table 2.1. For both algorithms, the inputs are the same generator vector* $\mathbf{g}$ *and the same received vector* $\mathbf{y}$*. Algorithm 1 outputs a nonzero bivariate linearized polynomial* $Q(x, y)$ *such that* $Q(g_i, y_i) \equiv 0$ *for* $i = 0, 1, \ldots, 5$*, while Loidreau's algorithm is expected to output nonzero* $Q_x(x)$ *and* $Q_y(u)$ *such that* $Q_x(g_i) = Q_y(y_i)$ *for* $i = 0, 1, \ldots, 5$ *respectively. Based on Lemma 4, we start from the initial polynomials of Loidreau's algorithm and the polynomials after the first* k *iterations by Algorithm 1. Note that* g_0 *in the final iteration is not listed, as it is of higher order than* g_1*. Since* $r(\mathbf{e}; q) = 2 \leq t = (n-k)/2$*,* $\mathbf{y}$ *is decodable. As shown in Table 2.1, however, Loidreau's algorithm fails. On the other hand, our algorithm produces a bivariate polynomial* $g_{n,1} = \alpha^4 x^4 + x^2 + \alpha^{29} x + \alpha^4 y^4 + y^2 + \alpha^{29} y$*, from which the correct decoding result* $f(x) = x$ *is obtained.*

Finally we consider the complexity of Algorithm 1 when used to decode Gabidulin codes. Adopting the same set of parameters in the complexity analysis in Section 2.3.2, we have $L = 1$, $C = n$, and $D = \lfloor \frac{n+k}{2} \rfloor$ based on the decodability conditions. Second, each linear functional in this case carries out evaluations of the bivariate linearized polynomial by the given points, which require a number of multiplications and additions determined by the q-degree of the linearized polynomial. Finally, the multiplication between $x^{[1]}$ and $g_{i,j}$ is accomplished by raising the coefficients of $g_{i,j}$ to the q-th power, which is simply a cyclic shift if a normal basis is chosen [37][38].

Actually, we do not have to use this maximum D to count the number of coefficients in the linearized polynomials in each iteration. Based on Lemma 4, for

iteration i with $1 \leq i \leq k$, $g_{0,i}$ is a linearized polynomial of q-degree $i-1$ at the beginning of each iteration, and is to be updated by the order-increase rule (see iterations 1 and 2 in Table 2.1). On the other hand, $g_{1,i}$ is a bivariate linearized polynomial with of the form $c_1 y + g'_{0,i-1}(x)$, where c_1 is a constant and $g'_{0,i-1}(x)$ is a linearized polynomial of x with q-degree no greater than $i-2$, and $g_{1,i}$ is to be updated by the cross-term rule (see iterations 1 and 2 in Table 2.1). For the last $n-k$ iterations, each bivariate linearized polynomial within each iteration has a q-degree of no more than $D = \lfloor \frac{n+k}{2} \rfloor$ as analyzed above, which leads to a number of at most $D + D - (k-1) = n-1$ coefficients for $g_{j,0}$ and $g_{j,1}$, where $k+1 \leq j \leq n$.

According to Algorithm 1, for $1 \leq i \leq k$, it will take $i-1$ finite field multiplications and $i-1$ finite field additions over GF(q^m) to obtain Δ_0, and $i-1$ multiplications and $i-1$ additions to update $g_{0,i}$. To obtain Δ_1, we need to carry out $i-1$ multiplications and $i-1$ additions, plus $2(i-1)$ multiplications and $i-1$ additions to update $g_{1,i}$. Hence there are $5i-5$ multiplications and $4i-4$ additions in each iteration, with a total of $\frac{5}{2}k^2 - \frac{5}{2}k$ finite field multiplications and $2k^2 - 2k$ additions in the first k iterations (see iterations 1 and 2 in Table 2.1). For the last $n-k$ iterations, we assume the q-degree of each linearized polynomial is $D = \lfloor \frac{n+k}{2} \rfloor$ for each bivariate linearized polynomial within each iteration as analyzed above, which leads to a number of at most $D + D - (k-1) = n-1$ coefficients for $g_{j,0}$ and $g_{j,1}$, where $k+1 \leq j \leq n$. Following similar arguments as in the first k iteration, the last $n-k$ iterations conduct $(5n-5)(n-k)$ multiplications and $(4n-4)(n-k)$ additions. To sum up, Algorithm 1 requires at most $5n^2 - 5nk + \frac{5}{2}k^2 - 5n + \frac{5}{2}k$ multiplications and at most $4n^2 - 4nk + 2k^2 - 4n + 2k$ additions over GF(q^m).

The algorithm in [30] needs an overall of $\frac{5}{2}n^2 - \frac{3}{2}k^2 + \frac{n-k}{2}$ multiplications and

$\frac{5}{2}n^2 - \frac{3}{2}k^2 + \frac{n-k}{2}$ additions over GF(q^m). The difference between the numbers of multiplications required by the two algorithms is $\frac{5}{2}(n-k)^2 + \frac{3}{2}(k+1)^2 - \frac{11}{2}n - \frac{3}{2}$, which is approximately $\frac{3}{2}n^2 - \frac{5}{2}n$ for high rate codes, and $\frac{5}{2}n^2 - \frac{11}{2}n$ for low rate codes. Hence both algorithms are of quadratic complexity, but Loidreau's algorithm will have a lower complexity in general. However, it should be pointed out that our complexity analysis of Algorithm 1 is only an upperbound, where a maximum q-degree is estimated for each linearized polynomial for the last $n-k$ iterations.

2.5 Decoding of KK Codes

For a KK code over GF(q^m) as described in Section 2.2.4, the decoding algorithm in [9] finds a **minimum** solution to (2.2) based on an interpolation procedure. In this section, we will show that this list-1 decoding algorithm is a special case of our interpolation algorithm over free $\mathbb{L}[x]$-modules, where $\mathbb{L}[x]$ is the ring of linearized polynomials over GF(q^m).

Lemma 6. *When $L = 1$, Algorithm 1 reduces to the Sudan-style list-1 decoding algorithm in [9].*

Proof. Suppose the received subspace is U at the decoder, with a dimension of r, and a basis set is $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{r-1}, y_{r-1})\}$. We assume that the condition of decodability [9] is satisfied so that an interpolation approach works to gives a solution of $Q(x, y)$. Given the linearized polynomial ring $\mathbb{L}[x]$ over GF(q^m), we set $L = 1$, choose a set $B = \{b_0, b_1\} = \{x, y\}$ as a basis, and construct the same free $\mathbb{L}[x]$-module $V = \{Q(x, y)\}$ with the same ordering as that in Section 2.4. We define a set of r linear functionals D_i to be $D_i(Q(x, y)) = Q(x_i, y_i)$ for $i = 0, 1, \ldots, r-1$.

Then Algorithm 1 has exactly the same initial values and the same update rules as the Sudan-style list-1 decoding algorithm in [9] (it should be pointed out that the pseudocode in [9] contains a typo, and no update is going to take place when both discrepancies are zero). □

It should be pointed out that in [9], bivariate linearized polynomials of *x-minimal* and *y-minimal* are constructed iteratively, while we find minimums in Algorithm 1. Here we show that the two definitions are equivalent and the final outputs of the two algorithms are the same (of the same order). According to the definition in [9], $f_0^{(i)}(x, y)$ is x-minimal if it interpolates through the first i points and is a minimal polynomial under $\prec$, while its leading term is in x. Comparing this definition to that in our interpolation construction, we find that this $f(x, y)$ is a minimum in $T_{i,0}$, hence $f_0^{(i)}(x, y) =_o g_{i,0}$. Similarly, $f_1^{(i)}(x, y)$ being y-minimal means that $f_1^{(i)}(x, y) =_o g_{i,1}$ in Algorithm 1. Since KK's decoding algorithm finds x-minimal and y-minimal bivariate linearized polynomials in each iteration, it works the same as Algorithm 1 during intermediate iterations. Finally, KK's decoding algorithm outputs the one with a smaller $(1, k-1)$-weighted degree, which equals to finding the minimum among $g_{C,0}$ and $g_{C,1}$ as performed in Algorithm 1. Hence the outputs or the two algorithms are the same.

2.6 List Decoding of MV Codes

In [10], the list decoding first constructs a multivariate polynomial $Q(x, y_1, y_2, \ldots, y_L)$ that interpolates through a number of given points as indicated by (2.3). Hence we

call this process the interpolation step of the list decoding of MV codes. No specific algorithm is mentioned in [10] on how to obtain this multivariate polynomial. Of course, a nonzero solution can be obtained by solving the corresponding homogeneous systems using Gaussian elimination, but it requires high computational complexity. Here, we utilize the interpolation over free $\mathbb{L}[x]$-modules to solve this problem efficiently. The complexity of our algorithm is compared to that of solving homogeneous equations.

As in Section 2.4, we have to construct a free module for a given ring, and define relative operations so that Algorithm 1 can be carried out. We consider an l-dimensional MV codes over GF(q^{ml}) defined in Section 2.2.5, with a message vector length of k and dimension of subspace l. In this case, the linearized polynomials ring $\mathbb{L}[x]$ is defined over GF(q^{ml}), and a set $B = \{b_0, b_1, \ldots, b_L\} = \{x, y_1, \ldots, y_L\}$ is selected to form a free $\mathbb{L}[x]$-module $V = \{Q(x, y_1, \ldots, y_L)\}$. Following a similar definition of the multiplication between $\mathbb{L}[x]$ and V, V is constructed in the same way as in Section 2.4. Hence an element $Q(x, y_1, \ldots, y_L) \in V$ can be written as $Q(x, y_1, \ldots, y_L) = Q_0(x) + Q_1(y_1) + \cdots + Q_L(y_L)$, called a *multivariate linearized polynomial*, where $Q_i(x) \in \mathbb{L}[x]$ for $i = 0, 1, \ldots, L$.

Following a similar process as in the previous section, V is also a vector space over GF(q^{ml}) with a vector space basis $M = \{x^{[i]} \circ b_j, i \geq 0, j = 0, 1, \ldots, L\}$. Then a $(1, k-1)$-weighted degree, and a total ordering on M can be defined in a similar way as in Section 2.5, by allowing j to be in $\{0, 1, \ldots, L-1\}$. It can be verified that the two conditions on the total ordering in Section 2.3 are satisfied. Further, we define the leading monomial and the order of any $Q \in V$ in the same way as in Section 2.3.1, as well as the minimum elements in a subset of V. Finally, a

set of linear functionals D_i for $i = 1, 2, \ldots, (t+l)m$ from V to $\mathrm{GF}(q^{ml})$ are also defined to be evaluations of multivariate linearized polynomials by the given points, as indicated in (2.3).

Since the total number of points to be interpolated in (2.3) is $(t+l)m$, the numbers of linear functionals D_i are $(t+l)m$. Furthermore, the kernels K_i are also $\mathbb{L}[x]$-submodules by Lemma 3. In summary, the interpolation problem in (2.3) is to find a nonzero $Q \in V$ such that $Q \in \overline{K}_{(t+l)m}$. Hence this is an interpolation problem over free $\mathbb{L}[x]$-modules, thus can be solved by Algorithm 1, which gives a **minimum** nonzero solution to (2.3), as stated in the following lemma.

Lemma 7. *The interpolation algorithm solves the interpolation problem of the list decoding algorithm for l-dimensional MV codes if the dimension of the error $t < lL - L(L+1)\frac{k-1}{2m}$.*

Proof. As shown in [10], when $t < lL - L(L+1)\frac{k-1}{2m}$, there exist nonzero solutions for the interpolation step of the list decoding algorithm for MV codes. Hence we assume $t < lL - L(L+1)\frac{k-1}{2m}$, then Algorithm 1 solves the interpolation problem by finding a **minimum** nonzero solution to (2.3), when we adopt the free modules and related operations as described above. □

For Algorithm 1, we set $g_{0,0} = x, g_{0,i} = y_i$ for $i = 1, 2, \ldots, L$ in the initialization step. The update rules in the intermediate iterations are the same as in Section 2.4, except that we have to use the new ordering related definitions in this section to determine a minimum among the $L+1$ candidates. We give an example of list decoding an MV code using Algorithm 1, with a list size of greater than one.

Example 2. *We construct an MV code with each message vector length* $k = 1$ *and corresponding subspace dimension* $l = 1$ *over GF*(2^2)*, where* $q = 2$ *and* $m = 2$*, leading to* $ml = 2$*. Finally, we fix a decoding list size of* $L = 2$*. Suppose* γ *is a root of the polynomial* $x^2 + x + 1$ *irreducible over GF*(2)*. It can be verified that* $(\gamma, \gamma^{[1]})$ *is a normal basis for GF*(2^2) *over GF*(2)*. Suppose the message vector is* $\mathbf{u} = (1)$*, with* $u(x) = x$ *and* $u^{\otimes 2}(x) = x$*. Then the subspace to be transmitted is spanned by* $(\gamma, u(\gamma), u^{\otimes 2}(\gamma)) = (\gamma, \gamma, \gamma)$*. Suppose an error of dimension one occurs, spanned by* $(\gamma^0, 0, 0)$*. Then the decoder is to find a bivariate linearized polynomial* $Q(x, y, z) = Q_0(x)+Q_1(y)+Q_2(z)$ *that interpolate through the following three points,* $(\gamma, \gamma, \gamma), (\gamma^{[1]}, \gamma^{[1]}, \gamma^{[1]}), (\gamma^0, 0, 0)$*, as* $(\gamma^0)^{[1]} = \gamma^0$*. The interpolation algorithm is carried out as in Table 2.2, where the ultimate output is* $f_2 = \gamma^2 z + \gamma^2 y$*. It can be verified that* $u(x) = x$ *satisfies* $f_2(x, u(x), u^{\otimes 2}(x)) \equiv 0$*, hence the original message vector is included in the decoding list.*

Table 2.2: Example 2: Use Algorithm 1 to decode an MV code

i	$f_0(x, y, z)$	$f_1(x, y, z)$	$f_2(x, y, z)$
1	$\Delta_0 = \gamma$ $f_0 = x^2 + \gamma x$	$\Delta_1 = \gamma$ $f_1 = \gamma y + \gamma x$	$\Delta_2 = \gamma$ $f_2 = \gamma z + \gamma x$
2	$\Delta_0 = \gamma^2$ $f_0 = x^4 + x$	$\Delta_1 = 0$ $f_1 = \gamma y + \gamma x$	$\Delta_2 = 0$ $f_2 = \gamma z + \gamma x$
3	$\Delta_0 = 0$ $f_0 = x^4 + x$	$\Delta_1 = \gamma$ $f_1 = \gamma^2 y^2 + \gamma^2 x^2$ $+\gamma^2 y + \gamma^2 x$	$\Delta_2 = \gamma$ $f_2 = \gamma^2 z + \gamma^2 y$

Finally we compare the complexities of Gaussian elimination and Algorithm 1 when used to decode l-dimensional MV codes. Since it is cumbersome and difficult to derive the exact number of multiplications or additions conducted, we display only the most significant terms for both algorithms. As mentioned above, a nonzero

multivariate linearized polynomial $Q(x, y_1, \ldots, y_L)$ can also be obtained by solving the homogeneous system determined by (2.3). The size of the coefficient matrix is $(t+l)m \times [ml(L+1) - \frac{k-1}{2}L(L+1)]$. If solved by Gaussian elimination, the calculation complexity is dominated by $m^3(t+l)^2lL/2 - m^2(t+l)^2L^2(k-1)/4$ finite field multiplications and $m^3(t+l)^2lL/2 - m^2(t+l)^2L^2(k-1)/4$ additions. Given the fact that $ml - L(k-1) - 1 \geq 0$ (the q-degree of $Q_L(y_L)$ has to be nonnegative), this complexity can be simplified to be $L^2m^2(t+l)^2(k-1)/4$, with an order of $O(L^2m^2(t+l)^2(k-1))$, for multiplications and additions respectively.

For Algorithm 1, we have $C = (t+l)m$ linear functionals in this case and a total of $L+1$ elements in the basis of the free module, and the highest q-degree of the linearized polynomials in x is at most $ml - 1$ for all iterations. Since the linear functional operation and the multiplication between elements in the ring and elements in the module are defined in the same manner as for KK codes, Algorithm 1 requires about $2m^2(t+l)lL^2 - m(t+l)L^2(k-1)$ finite field multiplications and $2m^2(t+l)lL^2 - m(t+l)L^2(k-1)$ addition, with orders of $O(L^2m^2(t+l)l)$, respectively. Hence the complexity of Gaussian elimination is $(t+l)(k-1)/(4l)$ times that of Algorithm 1 when $k > 5$. The complexities of the two algorithms are shown in Table 2.3, where GE stands for Gaussian elimination.

Table 2.3: Computational complexities of Gaussian elimination and Algorithm 1 for MV codes

Computation	Finite Field Multiplication	Finite Field Addition
GE	$L^2m^2(t+l)^2(k-1)/4$	$L^2m^2(t+l)^2(k-1)/4$
Algorithm 1	$2m^2(t+l)lL^2$	$2m^2(t+l)lL^2$

2.7 Hardware Implementations and Comparison

In order to demonstrate the advantage of the proposed interpolation algorithm in hardware implementation, an interpolator for list decoding of l-dimensional MV codes is implemented in hardware and is compared with Gaussian elimination. To this end, we focus on an MV code over GF(4^6) with k=1, l=3, m=2, t=5, and L=2. However, the proposed architecture and the implementation results can be readily extended to other MV codes.

For this code, Algorithm 1 reduces to Algorithm 2, where the interpolation finishes in N iterations. During iteration s ($s = 0, 1, \cdots, N-1$), a packet $(a_{0,s}, a_{1,s}, a_{2,s})$ is received and processed, where $a_{i,s} \in \mathrm{GF}(4^6)$, $i = 0, 1, 2$. To compare the throughput, N received packets are treated as a received word. Since the list decoder for MV codes fails when $N > (t + l)m$, N is set to $(t + l)m$. The polynomials $f_0(x, y, z)$, $f_1(x, y, z)$ and $f_2(x, y, z)$ are updated based on their values and the received packet, where $f_i(x, y, z) = fx_i(x) + fy_i(y) + fz_i(z) = \sum_{j=0}^{N_x-1} \mathrm{COEX}_i(j)x^{[j]} + \sum_{j=0}^{N_y-1} \mathrm{COEY}_i(j)y^{[j]} + \sum_{j=0}^{N_z-1} \mathrm{COEZ}_i(j)z^{[j]}$. fx_i, fy_i and fz_i are linearized polynomials in x, y and z, respectively, with coefficients $\mathrm{COEX}_i(j)$, $\mathrm{COEY}_i(j)$ and $\mathrm{COEZ}_i(j)$, respectively. N_x, N_y and N_z are the highest possible powers of fx_i, fy_i and fz_i. The PolyEvl function in Algorithm 2 evaluates $f_i(x, y, z)$ at $x = a_{0,s}$, $y = a_{1,s}$ and $z = a_{2,s}$, and computes $\Delta_i = f_i(x, y, z)|_{x=a_{0,s}, y=a_{1,s}, z=a_{2,s}}$. The OrderComp function in Algorithm 2 computes the order of $f_i(x, y, z)$, expressed as $O_i = \max(dx_i, dy_i + k - 1, dz_i + 2(k - 1))$, where $dx_i = \deg_q(fx_i(x))$, $dy_i = \deg_q(fy_i(y))$, $dz_i = \deg_q(fz_i(z))$.

Algorithm 2 Interpolation algorithm for L=2

Input: $(a_{0,s}, a_{1,s}, a_{2,s}); s = 0, \cdots, N-1$
Output: $d(x, y, z)$

$f_0(x, y, z) = x,\ f_1(x, y, z) = y,\ f_2(x, y, z) = z$
for $s = 0$ to $N - 1$ **do**
  for $i = 0$ to 2 **do**
    $\Delta_i = \text{PolyEvl}(f_i(x, y, z), a_{0,s}, a_{1,s}, a_{2,s})$
    $O_i = \text{OrderComp}(f_i(x, y, z))$
  end for
  $I_0 = \{i : \Delta_i \neq 0\}; I_1 = \{i : \Delta_i = 0\}$
  if $I_0 \neq \emptyset$ **then**
    $i^* \leftarrow \underset{i \in I_0}{\text{argmin}}\{O_i\}$
    for $i \in I_0$ **do**
      if $i \neq i^*$ **then**
        $f_i(x, y, z) = \Delta_{i^*} f_i(x, y, z) + \Delta_i f_{i^*}(x, y, z)$
      else
        $f_i(x, y, z) = \Delta_i (f_i(x, y, z))^{[1]} + \Delta_i^{[1]} f_i(x, y, z)$
      end if
    end for
  end if
  if $I_1 \neq \emptyset$ **then**
    for $i \in I_1$ **do**
      $f_i(x, y, z) = f_i(x, y, z)$
    end for
  end if
end for
$O_i = \text{OrderComp}(f_i(x, y, z)),\ i = 0, 1, 2$
$i^* \leftarrow \underset{i=0,1,2}{\text{argmin}}\{O_i\}$
$d(x, y, z) = f_{i^*}(x, y, z)$

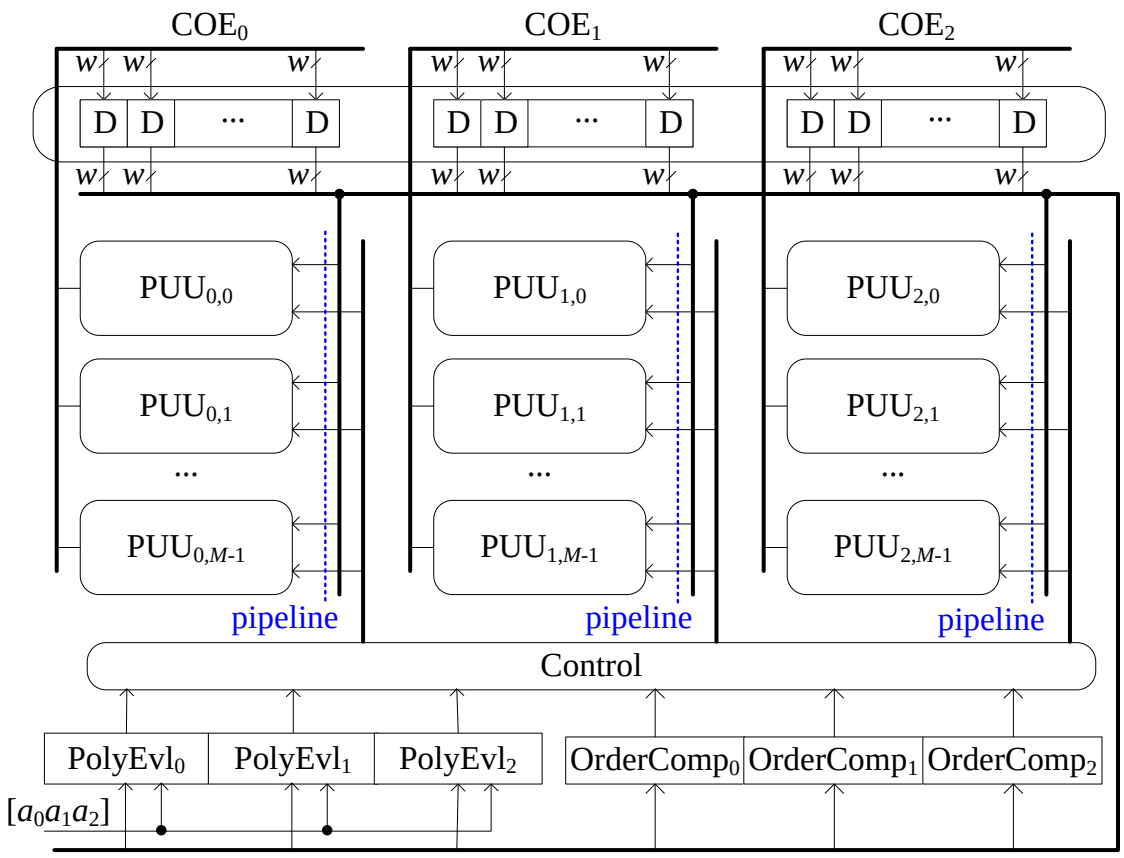

Figure 2.1: Interpolator top architecture

2.7.1 Hardware implementation of the interpolation algorithm

The top architecture of the proposed interpolator, shown in Fig. 2.1, consists of coefficient registers, polynomial update unit (PUU), PolyEvl and OrderComp units. COE_i stores the coefficients of $f_i(x, y, z)$, and contains three parts: COEX_i, COEY_i and COEZ_i. Each coefficient is an element in $\text{GF}(q^{ml})$, and hence requires $w = ml \log_2 q$ bits to represent. For the MV code mentioned above, each coefficient is an element of $\text{GF}(4^6)$ and hence needs a 12-bit register. It takes N cycles to finish the

interpolation. During each cycle, the PolyEvl$_i$ unit evaluates $f_i(x, y, z)$ to obtain Δ_i and the OrderComp$_i$ unit computes O_i. Using Δ_i and O_i, the PUU$_{i,j}$'s update the coefficients of $f_i(x, y, z)$, which are written back to COE$_i$.

As shown in Fig. 2.2, the PolyEvl$_i$ unit evaluates $f_i(x, y, z)$. Each element over GF(4^6) is represented as a six-dimensional vector over GF(4) with respect to some basis. For the proposed interpolator, finite field multiplications assume a polynomial basis representation. On the other hand, the exponentiation $\alpha^{[n]}$ reduces to a cyclic shift of six-dimensional vector when a normal basis is used. Thus, in Fig. 2.2, the polynomial basis representation is first transformed to its corresponding normal basis representation by using the Trans unit, and then an inverse transformation after cyclic shifts using the ITrans unit. We denote the polynomial and normal basis representations as $\mathbf{c} = (c_0, c_1, c_2, c_3, c_4, c_5)^T$ and $\mathbf{c'} = (c'_0, c'_1, c'_2, c'_3, c'_4, c'_5)^T$, respectively, where $c_i, c'_i \in$ GF(4). The Trans and ITrans units implement the conversions between them, namely, $\mathbf{c'} = \mathbf{Tc}$ and $\mathbf{c} = \mathbf{T^{-1}c'}$, respectively, where

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 3 & 1 & 0 & 0 \\ 1 & 3 & 2 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 & 3 & 0 \\ 1 & 0 & 0 & 2 & 2 & 0 \\ 1 & 1 & 1 & 2 & 0 & 0 \end{pmatrix}, \mathbf{T^{-1}} = \begin{pmatrix} 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 1 & 1 & 2 & 3 & 1 \\ 0 & 1 & 0 & 1 & 2 & 2 \\ 0 & 3 & 2 & 0 & 0 & 1 \\ 0 & 0 & 3 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

The summation in the PolyEvl$_i$ unit is simply bit-wise XOR.

The circuitry of the PUU$_{i,j}$ is shown in Fig. 2.4. Using the results of polynomial evaluation and order computation and control signals generated by the control unit

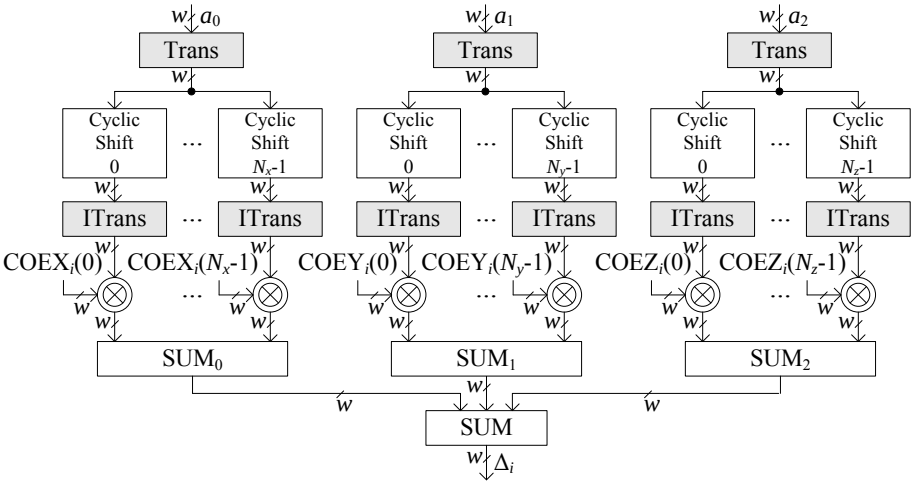

Figure 2.2: The architecture of PolyEvl_i

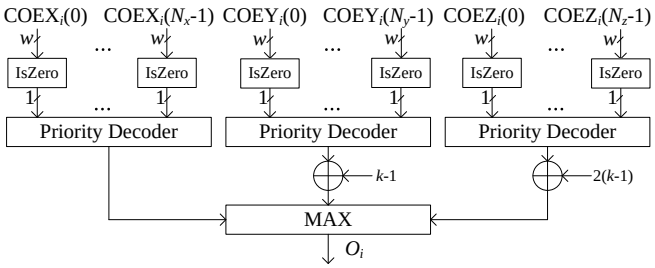

Figure 2.3: The architecture of OrderComp_i

in Fig. 2.1, the $\text{PUU}_{i,j}$ performs the corresponding updates according to Algorithm 2. When $j = 0$, COEX_i(-1), COEY_i(-1) and COEZ_i(-1) are set to zero.

For a (k, l, m, t) MV code with list size L, the hardware complexity of the proposed interpolator is dominated by $4c$ multipliers over $\text{GF}(q^{ml})$ and $c\ ml\ (\log_2 q)$-bit registers, where $c = (L+1)\left[(N+1)(L+1) - \frac{L(L+1)}{2}(k-1)\right]$ and $N = (t+l)m$. Without any pipelining, the critical path delay (CPD) of the proposed interpolator architecture is $\max(T_P, T_O) + T_C + T_{\text{PUU}}$, where T_P, T_O, T_C and T_{PUU} are the delays of polyEvl, OrderComp, Control and PUU, respectively. T_P and T_{PUU} are dominated by T_M (delay of a finite field multiplier) and $2T_M$, respectively. Since

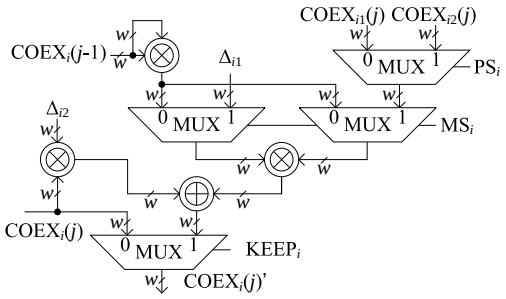

Figure 2.4: The circuitry of $\mathrm{PUU}_{i,j}$

T_M increases as the field size grows, to improve the clock rate of the interpolator, a stage of pipeline registers is inserted at the inputs of the PUUs, as shown in Fig. 2.1.

2.7.2 Implementation of Gaussian elimination

The interpolation for the list decoding of MV codes can also be done by applying Gaussian elimination to a $(t+l)m \times \left[ml(L+1) - \frac{k-1}{2}L(L+1)\right]$ coefficient matrix $\mathbf{M}$. For hardware implementation, Gaussian elimination can be performed using [39, Alg. 7] , which involves three operations: the eliminate, shiftup and shiftleft operations. For the MV code described above, we use the fully parallel architecture in [39], which consists of a 2-D array of 16×18 processing elements.

2.7.3 Implementation results comparison

For the MV code mentioned above, we synthesize the interpolator and Gaussian eliminator using the FreePDK 45nm standard cell library [40]. The results are shown in Table 2.4, where the gate count is the number of two-input one-output NAND gates and the efficiency is throughput per million NAND gates.

The proposed interpolator outperforms the Gaussian eliminator in two aspects.

Table 2.4: Hardware implementation results of the interpolator and Gaussian Eliminator.

	Int.	GE
Frequency (MHz)	751	869
Gate count	439K	550K
number of cycles	32	18 to 136
Throughput (10^6 words/second)	46.9	6.4 to 48.2
Efficiency	106.8	11.6 to 87.6

First, while the throughput of the interpolator is constant, the throughput of the Gaussian eliminator is variable and much smaller in the worst case. The interpolator finishes a round of interpolation in exactly 32 cycles. However, even when a fully parallel architecture is used, the throughput of the Gaussian eliminator varies with the received words. Under the best condition when $M_{0,0}$ is always non-zero, the Gaussian eliminator requires 18 cycles, and achieves a throughput of 48.2M words/s. Under the worst condition, the Gaussian eliminator takes $T(T+1)/2$ cycles [39], where $T = 16$ for the aforementioned MV code, and has a throughput of only 6.4M words/s. The constant throughput of the proposed interpolator is attractive for some applications. Second, the proposed interpolator saves 20% hardware compared to that of the Gaussian eliminator, and its efficiency is about 1.22 to 8.9 times of that of the Gaussian eliminator. The hardware implementation results are consistent with the complexity comparison in Section 2.6.

2.8 Conclusion

In this chapter, we investigate the interpolation problem over free modules of a linearized polynomial ring, and propose an interpolation algorithm. Our interpolation algorithm is used to decode Gabidulin codes and KK codes. Comparisons are made between our algorithm for Gabidulin codes and Loidreau's decoding algorithm. Analysis shows that the Sudan-style list-1 decoding algorithm for KK codes is a special case of our interpolation algorithm. Our interpolation algorithm is also used to find the multivariate linearized polynomial in the list decoding of MV codes. An interpolator for list decoding of MV codes has also been implemented in hardware, and hardware implementation results demonstrate the advantages of the proposed interpolator over a Gaussian eliminator.

Chapter 3

On List Decoding of Mahdavifar–Vardy Codes

3.1 Introduction

Error control is important to network coding due to network coding's vulnerability to errors, caused by unreliable links or malicious nodes. If unchecked, errors greatly deteriorate the throughput gains of network coding and seriously undermine both reliability and security of data. For *noncoherent* random linear network coding (RLNC), an operator channel is proposed in [9] to describe the transmission model, over which subspaces are transmitted and received, with errors and erasures defined to be addition and deletion, respectively, of vectors (packets) in the transmitted subspace. A family of subspace codes, referred to as KK codes, is also proposed in [9]. Just as Reed–Solomon (RS) codes can be constructed from evaluation of polynomials, KK codes are constructed from evaluation of linearized polynomials.

KK codes are shown to be asymptotically optimal [9], and can be decoded by a Sudan-style list decoding algorithm [9]. This list decoding algorithm has a list size of one, and hence it is essentially a bounded distance decoder with a decoding radius of approximately half the minimum distance.

To enable a larger list size and to achieve a greater decoding radius, Mahdavifar and Vardy have recently proposed a family of subspace codes, referred to as MV codes henceforth, and a list decoding algorithm for MV codes [10, 41]. Assuming that no erasures have occurred, this algorithm has a greater decoding radius than the decoding algorithm in [9] for low rate codes, and it is analogous to Sudan's algorithm for RS codes [42]. But MV codes and their decoding algorithm in [10] have several drawbacks. First, the assumption of no erasures is not feasible in practice, as erasures are common due to various reasons, such as dropped packets, node or link failure, or malicious attacks. Second, a greater decoding radius is achieved only for low rate codes. Third, no analytical performance evaluation is provided for the decoding algorithm of MV codes in [10].

In this chapter, we address these three drawbacks. First, to accommodate erasures, we treat the degree of the multivariate linearized polynomial at the interpolation step as a variable, and derive the condition of decodability. We also explain the asymmetric importance of errors and erasures in the new decodability condition. Second, motivated by the Guruswami–Sudan algorithm for RS codes [20], we introduce multiplicity into the interpolation step, attempting to achieve a greater decoding radius for high rate codes. Unfortunately, our results show that the decoding radius is slightly reduced due to properties of linearized polynomials. Finally,

after the decoding list is obtained, we form a nearest neighbor decoder, and calculate the decoder error probability (DEP). Assuming no erasures, we obtain an upper bound on the DEP, which decreases exponentially with the list size as well as the dimension of the subspaces in the code. When erasures occur during transmission, a closed-form expression of the DEP is obtained based on the results in [43].

The rest of this chapter is organized as follows. Section 3.2 reviews MV codes and their list decoding algorithm. Erasures and multiplicities are considered in Sections 3.3 and 3.4, respectively. A nearest neighbor decoder is formed in Section 3.5, and its DEP with and without erasures is analyzed. Section 3.6 offers some concluding remarks.

3.2 List Decoding of MV Codes

A subspace code is a subset of the projective space of an ambient space, which is the set of all the subspaces of an ambient space. Suppose W is a vector space over a finite field GF(q), where q is a prime power, and $\mathcal{P}(W)$ is the set of all subspaces of W. For $U, V \in \mathcal{P}(W)$, the subspace distance d_s [9] between them is defined as $d_s(V,U) \stackrel{\text{def}}{=} \dim(V+U) - \dim(V \cap U)$, where $\dim(A)$ denotes the dimension of a subspace $A \in \mathcal{P}(W)$, $V \cap U$ is the intersection space of V and U, and $V+U$ is the smallest subspace that contains both V and U.

To construct an l-dimensional MV code over GF(q^{ml}), an extension field of GF(q), l has to be a positive integer that divides $q-1$. Then $x^l - 1 = 0$ has l distinct roots $e_1 = 1, e_2, \ldots, e_l$ over GF(q). In [10], first a primitive element γ in GF(q^{ml})

with $\gamma, \gamma^{[1]}, \ldots, \gamma^{[ml-1]}$ being a normal basis for $\mathrm{GF}(q^{ml})$ is chosen. Then the elements α_i are constructed over $\mathrm{GF}(q^{ml})$ by $\alpha_i = \gamma + e_i\gamma^{[m]} + e_i^2\gamma^{[2m]} + \cdots + e_i^{l-1}\gamma^{[m(l-1)]}$ for $i = 1, 2, \ldots, l$, where $[m] = q^m$. It is proved [41] that the set $\{\alpha_i^{[j]} : i = 1, 2, \ldots, l, j = 0, 1, \ldots, m-1\}$ is a basis of $\mathrm{GF}(q^{ml})$.

For a message vector $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ over $\mathrm{GF}(q)$, the message polynomial is $u(x) = \sum_{i=0}^{k-1} u_i x^{[i]}$, where $[i] = q^i$. $u(x)$ is a *linearized polynomial* with a *q-degree* of i^* if i^* is the largest index with $u_{i^*} \neq 0$. The product between two linearized polynomials $u(x)$ and $v(x)$ is defined to be $u(x) \otimes v(x) \stackrel{\text{def}}{=} u(v(x))$. Let $u^{\otimes i}(x)$ denote the product of $u(x)$ with itself by i times for any nonnegative integer i, with $u^{\otimes 0}(x) = x$. Then the codeword V corresponding to the message $\mathbf{u}$ is spanned by a set of vectors $\mathbf{v}_i$, where $\mathbf{v}_1 = (\alpha_1, u(\alpha_1), u^{\otimes 2}(\alpha_1), \ldots, u^{\otimes L}(\alpha_1))$, $\mathbf{v}_i = (\alpha_i, \frac{u(\alpha_i)}{\alpha_i}, \ldots, \frac{u^{\otimes L}(\alpha_i)}{\alpha_i})$ for $i = 2, 3, \ldots, l$, and L is the desired list size. Then V is an l-dimensional subspace of the $(Lm + l)$-dimensional ambient space $W = \langle \alpha_1, \alpha_2, \ldots, \alpha_l \rangle \oplus \underbrace{\mathrm{GF}(q^m) \oplus \cdots \oplus \mathrm{GF}(q^m)}_{L \text{ times}}$. Suppose an error of dimension t occurs, and an $(l + t)$-dimensional subspace U of W is received. The decoder first finds subspaces U_i such that $U_i = \{(x, y_1, y_2, \ldots, y_L) : x \in \langle \alpha_i \rangle\}$ for $i = 1, 2, \ldots, l$. Then, a basis $\{(x_{1,j}, y_{1,1,j}, y_{1,2,j}, \ldots, y_{1,L,j}) : j = 1, 2, \ldots, r_1\}$ of U_1 is found, where r_1 is the dimension of U_1. If $l = 1$, we ignore the first step and simply find a basis for the $(t + 1)$-dimensional received subspace U_1. For $i = 2, 3, \ldots, l$, the decoder obtains $U_i' = \{(x, \alpha_i y_1, \alpha_i y_2, \ldots, \alpha_i y_L) : (x, y_1, y_2, \ldots, y_L) \in U_i\}$, and finds a basis $\{(x_{i,j}, y_{i,1,j}, y_{i,2,j}, \ldots, y_{i,L,j}) : j = 1, 2, \ldots, r_i\}$ of U_i', where r_i is the dimension of U_i. Finally, the decoder constructs a nonzero *multivariate linearized polynomial* $Q(y_0, y_1, y_2, \ldots, y_L) = \sum_{i=0}^{L} Q_i(y_i)$, where Q_i is a linearized polynomials over $\mathrm{GF}(q^{ml})$ of q-degree at most $ml - i(k-1) - 1$, such that for $n = 1, 2, \ldots, l$,

$j = 1, 2, \ldots, r_i$, and $h = 0, 1, \ldots, m-1$,

$$Q(x_{n,j}^{[h]}, y_{n,1,j}^{[h]}, \ldots, y_{n,L,j}^{[h]}) = 0. \tag{3.1}$$

We call this procedure the *interpolation step* of the list decoding algorithm for MV codes. Using the LRR algorithm in [10], the decoder finds all possible polynomials $\hat{u}(x)$'s such that $E(x) = Q(x, \hat{u}(x), \hat{u}^{\otimes 2}(x), \ldots, \hat{u}^{\otimes L}(x)) \equiv 0$, and we call such a process the *factorization step* of the decoding algorithm. It is proved [41] that if

$$t < lL - L(L+1)\frac{k-1}{2m}, \tag{3.2}$$

Eq. (3.1) has a nonzero solution and there are at most L solutions satisfying $E(x) \equiv 0$, among which the transmitted message polynomial $u(x)$ is guaranteed to be included.

3.3 Correction of Erasures

For MV codes [10], errors and erasures are not equally important, that is, MV codes and their decoding algorithm in [10] do not handle erasures over the operator channel. For one-dimensional MV codes, no erasure can be corrected, as a single erasure of dimension one results in a total loss of all the information of the transmitted subspace. Hence we consider only l-dimensional MV codes with $l > 1$. The list decoding algorithm in [10] constructs a nonzero multivariate linearized polynomial $Q(y_0, y_1, \ldots, y_L)$ of q-degree $ml - 1$ in the interpolation step. If an erasure of dimension ρ happens, there are $m(l-\rho)$ linearly independent zeros for the linearized

polynomial $E(x) = Q(x, \hat{u}(x), \ldots, \hat{u}^{\otimes L}(x))$ in the factorization step. To have $E(x)$ identically zero, we have $m(l-\rho) > ml-1$ in the factorization step. Hence ρ has to be zero, that is, no erasures are accommodated in this decoding algorithm. We observe that this happens because the q-degree of $Q(y_0, y_1, \ldots, y_L)$ is set to be a fixed value, $ml-1$. Here, we set the q-degree of this multivariate linearized polynomial as a variable, and derive the condition of decodability when erasures occur over the operator channel.

Suppose $Q(y_0, y_1, \ldots, y_L) = \sum_{i=0}^{L} Q_i(y_i)$, where the q-degree of $Q_i(y_i)$ is $\tau - 1 - i(k-1)$ for $0 \leq i \leq L$. We assume an erasure of dimension ρ and an error of dimension t occur during the transmission. Then for a nonzero solution of $Q(y_0, y_1, \ldots, y_L)$, we have

$$m(l-\rho+t) < \sum_{i=0}^{L} \tau - i(k-1). \tag{3.3}$$

To let the linearized polynomial $Q(x, \hat{u}(x), \ldots, \hat{u}^{\otimes L}(x))$ be identically zero, we have $m(l-\rho) > \tau - 1$. Then the condition of decodability is

$$L\rho + t < Ll - \frac{k-1}{2m} L(L+1). \tag{3.4}$$

When we select $\tau = \lceil \frac{r+1}{L+1} m + \frac{k-1}{2} L \rceil$, the condition of decodability is satisfied. In this case, an erasure of dimension ρ can be handled as long as (3.4) is satisfied. Note that when we let $\rho = 0$, (3.4) is the same as (3.2).

The condition of decodability in (3.4) indicates that an erasure of dimension one is equivalent to an error of dimension L, where L is the list size. The reason of this asymmetry is explained here. $m(l-\rho) > \tau - 1$ implies that a one-dimensional

erasure causes a reduction of the q-degree of $Q_i(y_i)$ for $i = 0, 1, \ldots, L$ by m compared to the case where only errors happen. Then the number of unknowns is reduced by $m(L+1)$ in (3.3). Note that (3.3) involves the erasure on the left hand side and a coefficient of m on both sides. Hence each dimension of erasure causes a decrease in the number of unknowns by L compared to the case with only errors. As a result, the largest possible dimension of the error is reduced by L. Hence an erasure of dimension one is equivalent to an error of dimension L.

3.4 Effects of Multiplicities on the List Decoding

In the Guruswami-Sudan algorithm for RS codes [35], multiplicities are imposed on each point to be interpolated so that a greater decoding radius can be achieved. Naturally we try to improve the list decoding algorithm for MV codes by adding multiplicity too. We first introduce a definition of multiplicity for multivariate linearized polynomials, and then examine its effects on the list decoding algorithm.

3.4.1 Definitions

Suppose a multivariate linearized polynomial is given by $Q(y_0, y_1, \ldots, y_L) = \sum_{i=0}^{L} Q_i(y_i)$, where $Q_i(y_i) = \sum_{j=0}^{n_i} a_{j,i} y_i^{[j]}$ is a linearized polynomial with $a_{j,i} \in \mathrm{GF}(q^m)$, and n_i a non-negative integer for $i = 0, 1, \ldots, L$. We say the multivariate linearized polynomial $Q(y_0, y_1, \ldots, y_L)$ has a *zero* at a point $(\beta_0, \beta_1, \ldots, \beta_L)$ if $Q(\beta_0, \beta_1, \ldots, \beta_L) = 0$, where $\beta_0, \beta_1, \ldots, \beta_L$ are in some extension field $\mathcal{K}$ of $\mathrm{GF}(q^m)$, and the point $(\beta_0, \beta_1, \ldots, \beta_L)$ is called a *root* of $Q(y_0, y_1, \ldots, y_L)$. Next we introduce the concept of *multiplicity* for multivariate linearized polynomials.

Definition 1. *$Q(y_0, y_1, \ldots, y_L) = \sum_{i=0}^{L} \sum_{j=0}^{n_i} a_{j,i} y_i^{[j]}$, a multivariate linearized polynomial is said to have a zero of multiplicity q^s at $(0, 0, \ldots, 0)$ if $a_{j,i} = 0$ for any $j < s$ and any $i \in \{0, 1, \ldots, L\}$.*

Definition 2. *We say a multivariate linearized polynomial $Q(y_0, y_1, \ldots, y_L)$ has a zero of multiplicity q^s at $(\beta_0, \beta_1, \ldots, \beta_L)$ if $Q(y_0 + \beta_0, y_1 + \beta_1, \ldots, y_L + \beta_L)$ has a zero of multiplicity q^s at $(0, 0, \ldots, 0)$.*

The sufficient and necessary condition for a nonzero point to have a multiplicity of q^s is given by the following lemma.

Lemma 8. *A multivariate linearized polynomial $Q(y_0, y_1, \ldots, y_L) = \sum_{i=0}^{L} \sum_{j=0}^{n_i} a_{j,i} y_i^{[j]}$ has a zero of multiplicity q^s at $(\beta_0, \beta_1, \ldots, \beta_L)$ if and only if $Q(\beta_0, \beta_1, \ldots, \beta_L) = 0$ and $a_{j,i} = 0$ for any $j < s$ and any $i \in \{0, 1, \ldots, L\}$.*

The proof is omitted due to limited space. Lemma 8 and Definition 1 indicate that any nonzero root of a multivariate linearized polynomial has the same multiplicity as the all-zero root. Hence we conclude that all the roots of a multivariate linearized polynomial have the same multiplicity. This interesting fact could be explained as follows. Since $a_{j,i} = 0$ for any $j < s$ and any $i = 0, 1, \ldots, L$, we can write the polynomial as $Q(y_0, y_1, \ldots, y_L) = (Q'(y_0, y_1, \ldots, y_L))^{[s]}$, where $Q'(y_0, y_1, \ldots, y_L) = \sum_{i=0}^{L} \sum_{j=0}^{n_i - s} b_{j,i} y_i^{[j]}$ is a multivariate linearized polynomial such that $b_{j,i}^{[s]} = a_{j-s,i}$. Hence every root of $Q'(y_0, y_1, \ldots, y_L)$ is also a root of $Q(y_0, y_1, \ldots, y_L)$, with a multiplicity of q^s.

3.4.2 Effect of Multiplicities

For an l-dimensional MV code ($l \geq 1$), we suppose each root has a multiplicity of q^{sl} for $Q(y_0, y_1, \ldots, y_L) = \sum_{i=0}^{L} Q_i(y_i)$, where $Q_i(y_i)$ has a q-degree of $\tau - i(k-1) - 1$. As explained in Section 3.4.1, we can write $Q = (Q')^{[sl]}$ for some multivariate linearized polynomial $Q' = \sum_{i=0}^{L} Q'_i$, where Q'_i is a linearized polynomial with a q-degree $\tau - sl - i(k-1) - 1$. We choose $\tau = ml$, and let $Q(\beta_{i,0}^{[j]}, \beta_{i,1}^{[j]}, \ldots, \beta_{i,L}^{[j]}) = 0$ for $i = 1, 2, \ldots, n$ with $n \leq t + l$, and $j = 0, 1, \ldots, m - s - 1$, where $(\beta_{i,0}, \beta_{i,1}, \ldots, \beta_{i,L})$ are the points to be interpolated (obtained as described in Section 3.2). Then there are a total of no more than $(m-s)(t+l)$ equations, while the number of unknowns is $\sum_{i=0}^{L}(m-s)l - i(k-1)$. Hence when

$$t < lL - \frac{k-1}{2(m-s)} L(L+1), \tag{3.5}$$

a nonzero solution of Q' can be obtained. In addition, the q-degree of $E'(x) = Q'(x, u(x), \ldots, u^{\otimes L}(x))$ is $(m-s)l - 1$, while it has $(m-s)l$ linearly independent roots, implying that $E'(x)$ is identically zero. Hence (3.5) ensures decodability when the multiplicity of each point is q^{sl}.

Comparing the decodability condition in (3.5) to that in (3.2), we note that the introduction of multiplicity actually slightly reduces the decoding radius. This is due to the unique properties of linearized polynomials. As mentioned above, for multivariate linearized polynomials, the multiplicities of all the points are not independent, and they have to be the same. Further, Lemma 8 and Definition 2 indicate that a multiplicity of q^s at each point defines a same set of $s(L+1)$ extra constraints on the unknowns. Hence unlike the case for RS codes, the same multiplicity on each

interpolated point does not produce a number of extra linear constraints that is proportional to the number of points to be interpolated.

3.5 Decoder Error Probability

Motivated by classic coding theory, given the list of possible codewords returned by the list decoding algorithm, we propose to choose a codeword with the minimum subspace distance to the received subspace from the list. In classic coding theory, this approach is attractive since it ensures that, when the returned list contains the transmitted codeword, a list decoding algorithm performs no worse than a maximum likelihood (ML) decoding algorithm, provided that a nearest neighbor decoder is equivalent to an ML decoder. With this additional step, we obtain a nearest neighbor decoder up to the decoding radius of the list decoding algorithm in [10].

A decoder error happens when this nearest neighbor decoder produces an incorrect codeword, and we analyze its decoder error probability (DEP) with and without erasures. We assume that all possible received subspaces of the same dimension and at the same subspace distance from the transmitted subspace are equiprobable.

3.5.1 DEP without Erasures

We first consider the case where only errors happen over the operator channel. Under this assumption, the transmitted codeword is actually a subspace of the received subspace. For an l-dimensional MV code $\mathcal{C}$, suppose $V \in \mathcal{C}$ is transmitted, and an error of dimension t occurs, resulting in a $(t+l)$-dimensional received subspace U. We use a nearest neighbor decoder with a decoding radius t^*, where $t^* < lL - \frac{L(L+1)}{2}\frac{k-1}{m}$

based on (3.2). If $t > lL - \frac{L(L+1)}{2}\frac{k-1}{m}$, there is no nonzero solution for (3.1), and the list decoding algorithm simply fails. Hence we consider the case with $t \leq t^*$, where the list decoding algorithm will generate a decoding list that includes the transmitted codeword. Given that $d_s(U,V) = t$, a decoder error occurs only when $d_s(U,V') \leq t$, where $V' \neq V \in \mathcal{C}$. Then $d_s(U,V') = l+t+l-2\dim(U\cap V') \leq d_s(U,V)$ only when $\dim(U \cap V') = l$, i.e., the received subspace U contains other codeword V' as its subspace.

Let $p(s|V)$ denote the probability that the list decoding algorithm returns a list with $s+1$ codewords in $\mathcal{C}$, where $\mathcal{C}$ has a minimum subspace distance of $2d$. Then $p(s|V)$ is the number of $(t+l)$-dimensional subspaces that only contain V and s other codewords in $\mathcal{C}$ divided by the number of $(t+l)$-dimensional subspaces that contain V, which can be upper bounded by

$$p(s|V) \leq \sum_{l_s=l+d}^{t+l} \frac{N_{s,l_s} B_{s,l_s}}{\prod_{i=0}^{l-1}(q^l - q^i)\prod_{j=0}^{t-1}(q^{Lm+l} - q^{l+j})}, \tag{3.6}$$

where N_{s,l_s} is the number of sets of s codewords $V_1, V_2, \ldots, V_s \in \mathcal{C}$ such that $V_i \neq V$ for $i = 1, 2, \ldots, s$ and $U_v = V + V_1 + V_2 + \cdots + V_s$ is an l_s-dimensional subspace that does not contain any other codeword in $\mathcal{C}$, $B_{s,l_s} = \prod_{i=0}^{l_s-1}(q^{l_s} - q^i)\prod_{j=0}^{t+l-l_s-1}(q^{Lm+l} - q^{l_\mathcal{C}+j})$, and $l_\mathcal{C}$ is the dimension of $V_\mathcal{C}$, the smallest subspace that contains all the codewords in $\mathcal{C}$. For further references, we denote $p(s,l_s) = B_{s,l_s}/(\prod_{i=0}^{l-1}(q^l - q^i)\prod_{j=0}^{t-1}(q^{Lm+l} - q^{l+j}))$.

It can be easily shown that $p(s,l_s+1) < p(s,l_s)$, then $p(s|V) \leq \sum_{l_s=l+d}^{t+l} N_{s,l_s} p(s, l_s = l+d) < \sum_{l_s=l+d}^{t+l} N_{s,l_s} q^{-(Lm-(l+d))d}$ from $p(s, l_s = l+d) < q^{-(Lm-(l+d))d}$. Furthermore, an upper bound on N_{s,l_s} can be given by $\binom{q^k-1}{s} \leq (q^k-1)^s < q^{ks}$. Then the DEP is

bounded by

$$p_e \leq \sum_{s=1}^{L-1} \frac{s}{s+1} p(s|V) < (t-d)q^{-(L(md-k)-(l+d)d)}, \tag{3.7}$$

which decreases exponentially with the list size L, the degree of extension m over the base field, and the minimum subspace distance of the code.

3.5.2 DEP with Erasures

Now suppose an erasure of dimension ρ happens aside from the error of dimension t. We use a nearest neighbor decoder after the list decoding algorithm in [10], with a decoding radius $t^* < lL - \frac{L(L+1)}{2}\frac{k-1}{m}$ by (3.4). If $L\rho + t > lL - \frac{L(L+1)}{2}\frac{k-1}{m}$, no nonzero solution for (3.3) can be found, and the list decoding algorithm fails. Now suppose $L\rho + t \leq t^*$, then the decoder will produce an error if $d_s(U, V') \leq \rho + t$ for some $V' \neq V \in \mathcal{C}$. Suppose $A_w(V)$ [43] is the distance distribution of $\mathcal{C}$ with respect to V. Then based on the results of [43], the DEP is given by

$$p_e(V, \rho, t, d) = \frac{1}{N(\rho, t)} \sum_{w=d}^{l} A_w(V) \sum_{s=0}^{\rho+t} J(\rho, t, s, w), \tag{3.8}$$

when $N(\rho, t) > 0$, and $\rho + 2t \geq d$; and $p_e(V, \rho, t, d) = 0$ otherwise. In (3.8), $N(\rho, t)$ is the number of $(l - \rho + t)$-dimensional subspaces at subspace distance $\rho + t$ from V, and $J(\rho, t, s, w)$ is the number of $(l - \rho + t)$-dimensional subspaces that are at a subspace distance $\rho + t$ from V and a subspace distance s from another l-dimensional subspace at a distance w from V [43].

3.6 Conclusion

This chapter addresses three problems about the list decoding of MV codes: correction of erasures, effects of multiplicities, and decoder error probability for a nearest neighbor decoder based on the list decoding algorithm. We derive the condition of decodability for the list decoding algorithm assuming erasures. We also attempt to achieve a greater decoding radius by introducing multiplicity to the interpolated points. But our results show that the decoding radius is slightly reduced. Finally, by forming a nearest neighbor decoder up to the decoding radius of the list decoding algorithm in [10], we evaluate the decoder error probability of this nearest neighbor decoder.

Chapter 4

Rank Deficient Decoding of Linear Network Coding

4.1 Introduction

Due to its promise of significant throughput gains as well as other advantages, network coding [2, 3, 5] is already used or considered for a wide variety of wired and wireless networks (see, for example, [13–17]). One significant drawback of network coding is that a full rank of received packets at the receiver nodes of a multicast (or a unicast) is needed before decoding can start, leading to long delays and low throughputs, especially when the number of packets of a session is large. This is particularly undesirable for applications with stringent delay requirements.

Aiming to solve this problem, we propose rank deficient decoding for linear network coding, which can start even when the received packets are not full rank. By reformulating the decoding problem of network coding in a different fashion, the

decoding problem reduces to a collection of syndrome decoding problems. Solving these syndrome decoding problems, rank deficient decoding leads to smaller delays and higher throughputs, at the expense of possible decoding errors. Specifically, we propose two classes of rank deficient decoders with different complexities. The decoders of the first class, called Hamming norm (HN) decoders, take advantage of the sparsity inherent in data and produce the data vectors with the smallest Hamming weight. Since the HN decoders have high complexities for large size systems, we propose a class of decoders based on linear programming, referred to as linear programming (LP) decoders. Considering linear programming relaxation of the Hamming norm decoders and solving them by using standard linear programming procedures, the linear programming decoders have polynomial complexities and are much more affordable. Both classes of decoders recover data from fewer received packets and hence achieve higher throughputs and shorter delays than the full rank decoder. Since these decoders could produce erroneous outputs, within each class several different decoding strategies have been proposed for different tradeoffs between delay/throughput and data accuracy, and they include the full rank decoder of network coding as a special case.

In the literature, there are two related different approaches to dealing with the synergy of network coding and compressive sensing, and they also aim for different applications. Our work is quite different from both existing approaches. Above all, our reformulation of the decoding problem in network coding is novel, and this reformulation was not considered in the open literature to the best of our knowledge. One approach was proposed in [44], where statistical property of data blocks are taken advantage of to alleviate the "all-or-nothing" drawback of network coding

in distributed storage systems. In this approach, random linear network coding is used to encode coded blocks in distributed storage networks. Hence, this approach is not directly comparable to our work, which focuses on the decoding issue of linear network coding in general and applies to a wide variety of applications. The other approach [45, 46] aims to take advantage of the statistical correlation of data generated by distributed sensor networks. A salient feature of this approach is that in theory data are real values and linear combinations are now performed over the real (or complex) field. The rationale for this is that the real representation of data is a more natural one for sensor networks [45, 46]. In practice, data are represented in a finite precision system. It has been shown that information loss due to finite precision grows with the network size [47]. In contrast, in our work network coding remains over some finite fields, and hence our scheme does not suffer the information loss due to finite precision as the approach in [45, 46]. Thus, the full rank decoder remains the most relevant previous work, and henceforth we compare our rank deficient decoders with the full rank decoder only.

The rest of this chapter is organized as follows.

4.2 Rank Deficient Decoding

4.2.1 System Model

In this work, we treat all packets as N-dimensional row vectors over some finite field GF(q), where q is a prime power. Also, we focus on linear network coding (LNC) only, which was shown to be optimal in most cases [5]. Finally, we assume that the network is error-free, and error control (see, for example, [6, 9, 10, 48]) is not

embedded in network coding.

Suppose a source node of a unicast or multicast injects a collection of n data packets (or row vectors over GF(q)), $\boldsymbol{X}_0, \boldsymbol{X}_1, \cdots, \boldsymbol{X}_{n-1}$, into the network. At any sink node, m packets (or row vectors over GF(q)), $\boldsymbol{Y}_0, \boldsymbol{Y}_1, \cdots, \boldsymbol{Y}_{m-1}$, are received, where $\boldsymbol{Y}_i = \sum_{j=0}^{n-1} a_{i,j}\boldsymbol{X}_j$ for $i = 0, 1, \cdots, m-1$ and $a_{i,j} \in$ GF(q). Since the sink node can locally generate more linear combinations of $\boldsymbol{Y}_0, \boldsymbol{Y}_1, \cdots, \boldsymbol{Y}_{m-1}$, it is assumed that $\boldsymbol{Y}_0, \boldsymbol{Y}_1, \cdots, \boldsymbol{Y}_{m-1}$ are linearly independent, which implies that $m \leq n$. That is, the $m \times n$ matrix $\boldsymbol{A} = [a_{i,j}]$, often called the global coding kernel matrix, has a rank of m.

4.2.2 Full Rank Decoder

Let us further denote the matrices $\left[\boldsymbol{X}_0^T\, \boldsymbol{X}_1^T\, \cdots\, \boldsymbol{X}_{n-1}^T\right]^T$ and $\left[\boldsymbol{Y}_0^T\, \boldsymbol{Y}_1^T\, \cdots\, \boldsymbol{Y}_{m-1}^T\right]^T$ as $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively, where T is the matrix transpose operator. Since $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X}$, the sink node can recover the transmitted data packets by reversing the encoding of the data packets by the network. This is easily achievable when $m = n$, as the sink node can recover the data packets by computing $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{Y}$. Thus, the decoding in network coding starts only after the sink node has received n linearly independent combinations of the transmitted data packets. The required number of linearly independent packets received by the sink node leads to longer delays and lower throughputs, which may be undesirable for some applications.

4.2.3 Rank Deficient Decoding

We can formulate the data recovery problem at the sink node in a different way. Let us consider symbol l of $\boldsymbol{Y}_i$, and we have $Y_{i,l} = \sum_{j=0}^{n-1} a_{i,j}X_{j,l}$ for $i = 0, 1, \cdots, m-1$

and $l = 0, 1, \cdots, N-1$. Let us denote the column vectors $(Y_{0,l}\, Y_{1,l}\, \cdots\, Y_{m-1,l})^T$ and $(X_{0,l}\, X_{1,l}\, \cdots\, X_{n-1,l})^T$ as $\boldsymbol{V}_l$ and $\boldsymbol{W}_l$, respectively. Clearly, we have $\boldsymbol{V}_l = \boldsymbol{A}\boldsymbol{W}_l$ for $l = 0, 1, \cdots, N-1$. The sink node can recover the data packets if it can obtain $\boldsymbol{W}_l$ from

$$\boldsymbol{V}_l = \boldsymbol{A}\boldsymbol{W}_l \text{ for } l = 0, 1, \cdots, N-1. \tag{4.1}$$

Eq. (4.1) shows that the data recovery problem at the sink node can be viewed as N parallel decoding problems in Eq. (4.1), each corresponding to one symbol in the packet (or row vector). These N parallel decoding problems are equivalent to the decoding problem of linear network coding.

This reformulated problem is related to two well known decoding problems. First, if we treat the $m \times n$ matrix $\boldsymbol{A}$ as a parity check matrix for a linear block code of length n and dimension $n - m$, the decoding problem in Eq. (4.1) is closely related to a syndrome decoding problem. That is, the sink node needs to recover $\boldsymbol{W}_l$ based on the syndrome $\boldsymbol{V}_l$. Second, if we treat $\boldsymbol{W}_l$ as a data vector and $\boldsymbol{A}$ a measurement matrix, this is analogous to the decoding problem in compressive sensing.

4.2.4 Hamming Norm Decoders

Since the data recovery problem at any sink node is equivalent to a collection of parallel problems in Eq. (4.1), we focus on one such problem. In other words, we try to solve $\boldsymbol{V} = \boldsymbol{A}\boldsymbol{W}$ for $\boldsymbol{W}$, where $\boldsymbol{V}$ and $\boldsymbol{W}$ are m- and n-dimensional column vectors, respectively, and $\boldsymbol{A}$ remains an $m \times n$ matrix with full rank ($m \leq n$).

For a linear block code of length n and dimension $n-m$ with a parity check matrix

$\boldsymbol{A}$, $\boldsymbol{V} = \boldsymbol{AW}$ can be viewed as a syndrome of the received vector $\boldsymbol{W}$. It is well known that for a linear block code, the syndromes have a one-to-one correspondence with its cosets, each of which is of size q^{n-m}. In other words, all vectors in a coset lead to the same syndrome. Thus, solving $\boldsymbol{V} = \boldsymbol{AW}$ for $\boldsymbol{W}$ is equivalent to finding a vector within a coset.

If no side information is available, we can make a decision within the coset by taking advantage of some inherent properties of the data vector. In this work, we proceed by relying on the sparsity of the data vector, which is well justified in many applications. That is, the proposed Hamming norm decoders produce the vector with the smallest Hamming weight in the coset.

As is common in the compressive sensing literature, we consider two possible scenarios for sparsity. First, when $\boldsymbol{W}$ is sparse, we use a vector with the smallest Hamming weight in the coset corresponding to $\boldsymbol{V}$ as the estimate of $\boldsymbol{W}$. Second, suppose that $\boldsymbol{\Phi W}$ is sparse for a known nonsingular $n \times n$ matrix $\boldsymbol{\Phi}$. Since $\boldsymbol{V} = \boldsymbol{AW} = \boldsymbol{A\Phi^{-1}\Phi W}$, we can treat $\boldsymbol{V}$ as a syndrome for the linear block code defined by $\boldsymbol{A\Phi^{-1}}$. Thus, in this scenario, we first select a vector with the smallest Hamming weight in the coset of the code defined by $\boldsymbol{A\Phi^{-1}}$ corresponding to $\boldsymbol{V}$, and then produce an estimate of $\boldsymbol{W}$ by multiplying the selected vector with $\boldsymbol{\Phi^{-1}}$. In both scenarios, the key step is to select a vector with the smallest Hamming weight in the coset corresponding to the given syndrome. Thus, we assume $\boldsymbol{W}$ is sparse without loss of generality.

In coding theory terminology, a vector with the smallest Hamming weight among a coset is called a leader of the coset. Note that some coset leaders may not be unique, when more than one vector in the coset has the smallest Hamming weight.

In this case, either the coset leader is selected among these vectors at random or a list of all potential leaders is the output.

We remark that this problem is closely related to but different from the syndrome decoding problem in classic coding theory. In our decoding, a vector or a list of vectors with the smallest Hamming weight in the coset corresponding to the given syndrome is the estimate of the data vector. In the syndrome decoding problem, a coset leader is often considered as an estimate of the error vector. However, the key step in both problems is to select a vector or a list of vectors with the smallest Hamming weight in the coset corresponding to the given syndrome.

Thus, we have the following sufficient condition for successful decoding:

Lemma 9. *The minimum Hamming distance of the linear block code defined by* $\boldsymbol{A}$*, denoted by* $d_H(\boldsymbol{A})$*, satisfies* $d_H(\boldsymbol{A}) \leq m+1$*. When the Hamming weight of* $\boldsymbol{W}$*, denoted by* $w_H(\boldsymbol{W})$*, is less than half of the minimum Hamming distance of the linear block code defined by* $\boldsymbol{A}$*, that is* $w_H(\boldsymbol{W}) < \frac{d_H(\boldsymbol{A})}{2}$*,* $\boldsymbol{W}$ *can be recovered by syndrome decoding.*

Proof. The first part is due to the Singleton bound on the minimum Hamming distance of linear block codes. The second part holds because it is well known that a coset leader with Hamming weight less than $\frac{d_H(\boldsymbol{A})}{2}$ is unique. □

When $\boldsymbol{W}$ is not a unique coset leader, there are two possibilities. First, when the Hamming weight of $\boldsymbol{W}$ is minimal in its coset, either $\boldsymbol{W}$ has a probability to be selected when coset leaders are chosen at random or $\boldsymbol{W}$ is one of the possible vectors produced by the decoder, depending on whether the decoder needs to generate only one vector or a list of vectors. Second, when the Hamming weight of $\boldsymbol{W}$ is not

minimal, a wrong vector will be produced by the Hamming norm decoder.

4.2.5 Decoding Strategies

Possible outcomes of the full rank decoder are failure or success. In contrast, the proposed Hamming norm decoders may produce wrong decisions. Analogous to classical error control coding, the preference between decoding failures and decoding errors varies from one application to another. For instance, for applications with stringent delay constraints, partially correct data packets may be more desirable than decoding failures. For other applications such as cloud storage, data integrity may be a top priority than delays, especially when packet retransmission is possible. Hence, it is necessary to consider a wide range of decoding strategies so as to offer different tradeoffs between delay/throughput and accuracy.

Two extreme strategies are natural and straightforward. One extreme, called the error-free (EF) decoder, is similar to the full rank decoder in the sense that it decodes only if decoding success is guaranteed by Lemma 9. The other extreme, referred to as the best-effort (BE) decoder, always tries to decode with available received packets. The error-free and best-effort decoders represent the most conservative and the most aggressive strategies, respectively.

We also devise a family of decoding strategies that fills the gap between these two extremes based on one observation about error control codes. For an (n, k) perfect code over GF(2), we have $\sum_{i=0}^{t} \binom{n}{i} = 2^{n-k}$, where $t = \left\lfloor \frac{d_H(\boldsymbol{A})-1}{2} \right\rfloor$. In other words, all coset leaders are unique and have Hamming weight up to t. However, since most codes are not perfect and some allowance needs to be made. Hence, we devise a greedy-l decoding strategy: decodes only if $\sum_{i=0}^{cw-l} \binom{n}{i} = 2^{n-k}$, where cw

is the maximal possible Hamming weight of $\boldsymbol{W}$. The parameter l represents how aggressive the decoder is: for the same code defined by $\boldsymbol{A}$, the greater l is, the more aggressive the decoder. In fact, one can use different l values to approach the two extremes, the best-effort and error-free strategies.

4.2.6 Linear Programming Decoders

Since both the computational complexity and the memory requirement of the Hamming norm decoders grow exponentially with the size of $\boldsymbol{A}$, we also adopt a linear programming (LP) approach. Since $\boldsymbol{A}$ is not necessarily sparse, we formulate the problem based on that for binary linear block code with high-density polytopes in [49].

Let $x_0, x_1, \ldots, x_{n-1}$ be the variables representing the code bits of $\boldsymbol{W}$, and $\boldsymbol{V} = (v_0, v_1, \ldots, v_{m-1})^T$ be the syndrome received. For each check node $j \in \mathcal{J}$, let $T_j^E = \{0, 2, 4, \ldots, 2\lfloor |N(j)|/2 \rfloor\}$ for $v_j = 0$, and $T_j^O = \{1, 3, 5, \ldots, 2\lfloor (|N(j)| - 1)/2 \rfloor + 1\}$ for $v_j = 1$. Then the linear programming formulation for the syndrome decoding is to minimize $\sum_{i=0}^{n-1} f_i$ subject to the linear constraints in [49](14)–(19) except that $T_j = T_j^E$ if $v_j = 0$, and $T_j = T_j^O$ if $v_j = 1$. In contrast, $T_j = T_j^E$ in [49](14)–(19). In addition, we add a linear constraint to narrow down the optimal solutions:

$$\sum_{i=0}^{n-1} x_i \leq cw.$$

Linear programming may produce non-integral results, in which case two approaches are considered. The first is to round off the real values into integers, which are compared with the original data to compute decoding error or success rate, and

Table 4.1: Average packets for HN decoders ($N = 8$ over GF(2))

Strategy	FR	EF	greedy-(-1) HN	greedy-0 HN	greedy-1 HN	BE HN
100% PSR	9.60	8.84	8.12	7.57	7.44	7.44
95% BSR	9.60	8.84	8.05	7.40	7.17	7.17

Table 4.2: Average packets for LP decoders ($N = 8$ over GF(2))

Strategy	greedy-(-1)		greedy-0		greedy-1		BE	
	LP I	LP II	LP I	LP II	LP I	LP II	LP I	LP II
100% PSR	8.44	8.45	8.19	8.22	8.17	8.21	8.17	8.21
95% BSR	8.15	8.18	7.66	7.74	7.58	7.67	7.58	7.67

we call this approach LP I. The other, referred to as LP II, is to declare decoding failure. Both LP I and LP II are applicable to all greedy as well as the BE strategies.

4.3 Simulation Results

To illustrate the advantages of the proposed rank deficient decoders, we present some numerical simulation results with the following settings. Network coding is carried out over GF(2). We assume each session (or generation) consists of $n = 8$ packets of length $N = 8$ bits such that the transmission matrix has a constant column weight of $cw = 2$. The matrix $\boldsymbol{A}$ is generated randomly, with each element being 0 or 1 with equal probability. For each iteration, as the number of (linearly independent) received packets m increases from 1 to 15, the proposed decoders as well as the full rank decoder are used to decode, and their decoding success, failure, or error on both packet and bit levels are recorded. For each decoder, its packet- and bit-level success, failure, or error rate is obtained by averaging over 100,000 generations.

We note that such small values for n and N are chosen so that the complexities of

the Hamming norm decoders are manageable. We also note that in this setting, the data sparsity is manifested as an upper bound on the column weights in the transmitted data packets. We also have simulation results assuming other deterministic or stochastic manifestations of data sparsity, such as an upper bound on the row weights in the transmitted data packets, or the bits in the transmitted data packets being i.i.d. binary Bernoulli random variables with probability p ($p < 1/2$). Due to limited space, the simulation results for these other manifestations are omitted, but the proposed rank deficient decoders demonstrate similar advantages regardless of the manifestation of data sparsity.

In Fig. 4.1 and Fig. 4.2, respectively, the packet- and bit-wise fraction of decoding success, failures, and errors of Hamming norm decoders are represented by green, yellow, and red bars. Similarly, Fig. 4.3 and Fig. 4.4, respectively, compare the packet- and bit-wise fraction of decoding success, failures, and errors of linear programming decoders. In all figures, for each value of m, the six bars represent, from left to right, the full rank, error-free, greedy-(-1), greedy-0, greedy-1, and best-effort strategies, respectively. In order to measure and compare the throughput and delay of linear network coding with these decoders, the average minimum numbers of packets required to achieve a packet success rate (PSR) of 1 or a bit success rate (BSR) of 0.95 are compared in Table 4.1 and 4.2.

The simulation results confirm our claims about rank deficient decoders. The full rank decoder can recover data packets only when $m \geq n = 8$ and recovers no packet when $m < 8$. In contrast, our rank deficient decoders recover a greater fraction of data packets when $m \geq n = 8$, and recover a substantial fraction of data packets even when $m < 8$.

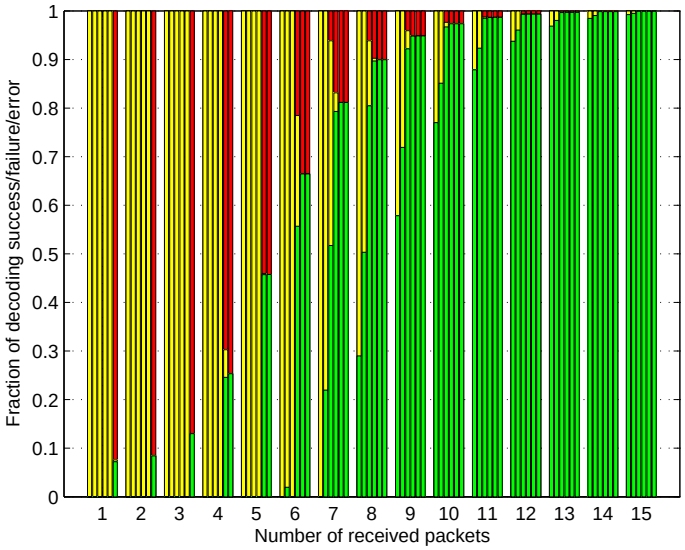

Figure 4.1: Packet-level performance of HN decoders ($N = 8$ over GF(2))

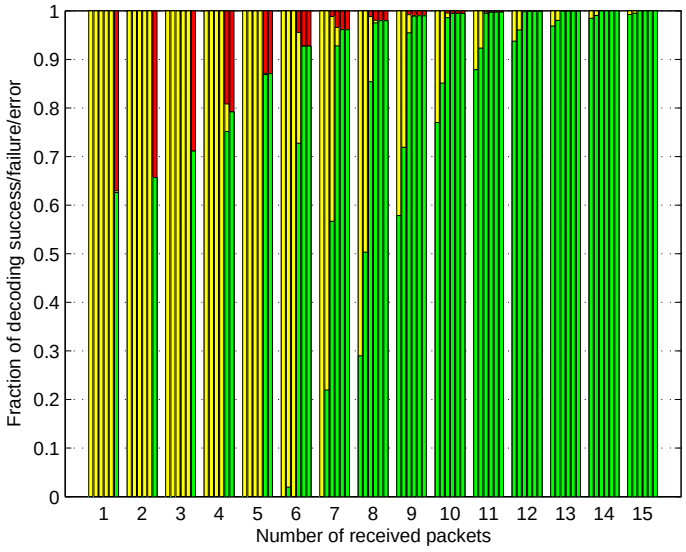

Figure 4.2: Bit-level performance of HN decoders ($N = 8$ over GF(2))

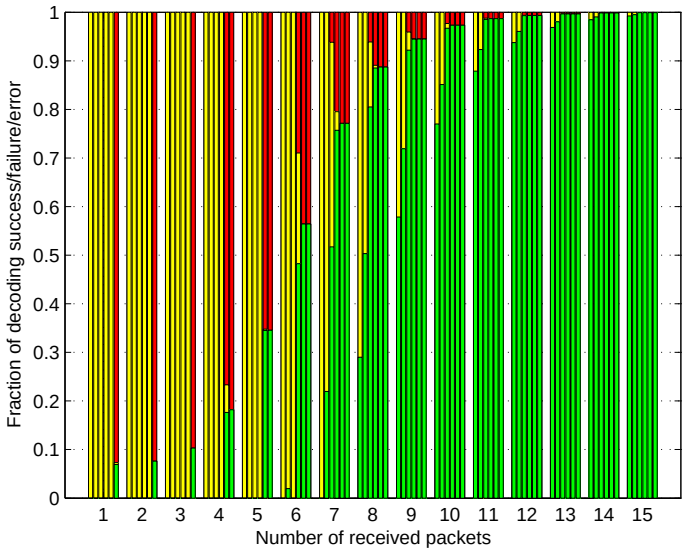

Figure 4.3: Packet-level performance of LP I decoder($N = 8$ over GF(2))

The proposed decoders provide a wide range of tradeoffs between delay/throughput and decoding errors. Just like the full rank decoder, the error-free strategy does not produce any decoding errors. Nevertheless, it outperforms the full rank decoder significantly for $m < n$. For instance, when $m = 7$, the error-free strategy recovers over 20% of the packets, while the full rank decoder cannot recover anything. At the other extreme, the performance of the best-effort strategy improves when m grows. For instance, when $m = 1$, it recovers around 10% of the packets and 70% of the bits. However, when $m = 7$, it recovers over 80% of the packets and 96% of the bits in the session. The greedy-l strategies fill the gap between the two extremes.

There is a difference between packet- and bit-level performances. For the full rank and error-free strategies, their packet- and bit-level performances are the same, because their decoding strategies depend on $\boldsymbol{A}$ only, and are the same for all l's in Eq. (4.1). For the other four strategies, since their decoding strategies depend on

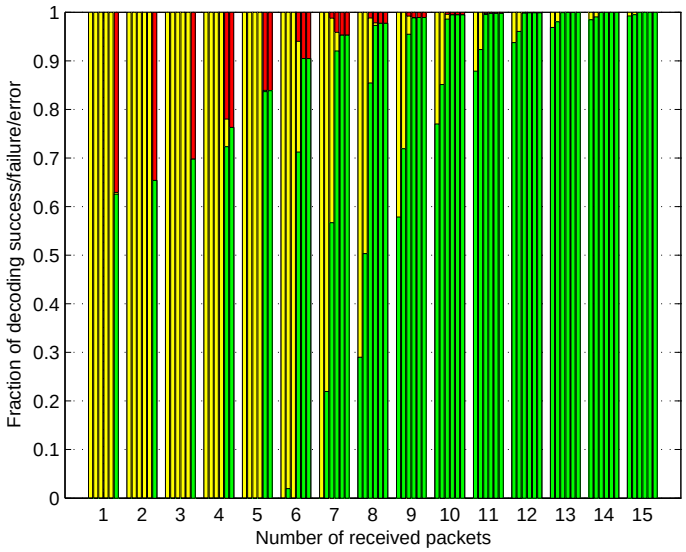

Figure 4.4: Bit-level performance of LP I decoder($N = 8$ over GF(2))

$\boldsymbol{A}$ as well as $\boldsymbol{V}_l$, their packet- and bit-level performances are different. Of course, their bit-level decoding success fractions are better than their respective packet-level decoding success fractions. This is because a packet-level decoding success requires bit-level decoding successes for all l's in Eq. (4.1).

Compared with the full rank decoder, the average minimum numbers of packets required for success decoding for the error-free and best-effort strategies are approximately 10% and 20% smaller, respectively. Assuming that the received packets arrive in a uniform interval, this means that throughputs achieved by the error-free and best-effort strategies are roughly 10% and 20%, respectively, higher than the full rank decoder. The actual advantage may be more significant, because it takes longer to receive a linearly independent packet when more received packets already exist.

As expected, the linear programming decoders perform slightly worse than the

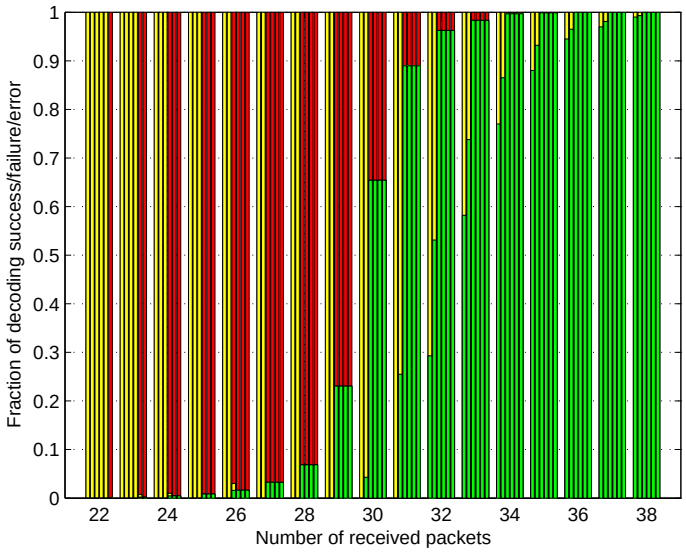

Figure 4.5: Packet-level performance of LP I decoder ($N = 32$ over GF(2))

Hamming norm decoders. However, the performance difference is negligible when the number of received packets is large.

As noted earlier, the computational complexity of Hamming norm strategy grows exponentially with code parameters, hence we adopt linear programming for different decoding strategies for simulations with larger parameters. Fig. 4.5 and 4.6 show the packet- and bit-wise simulation results with LP I for $n = N = 32$ and m ranges from 22 to 38, averaged over 1,000 sessions. Decoding success, failures, and errors are represented by green, yellow, and red bars, respectively. For each m value, the six strategies are FR, EF, greedy-(-1), greedy-0, greedy-1, and BE from left to right. We note that simulation results with m from 1 to 50 are obtained, but are truncated to between 22 and 38 to show results of interest.

Similar to the small parameter case, the FR strategy does not produce any decoding results when $m < 32$, while it doesn't produce any decoding errors, as the

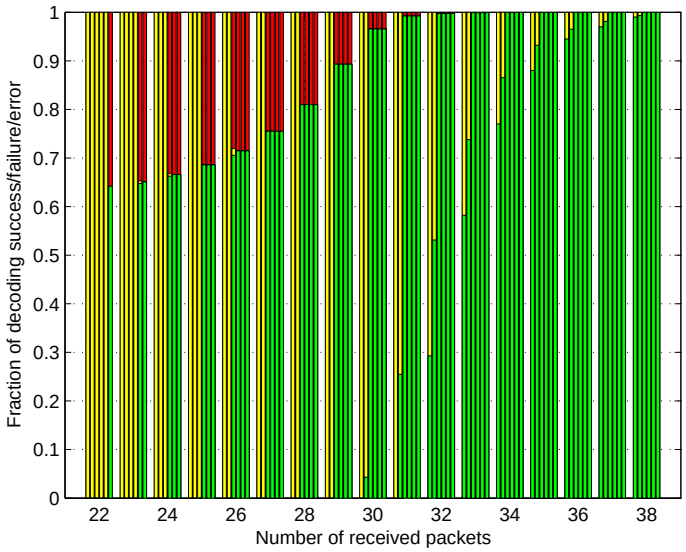

Figure 4.6: Bit-level performance of LP I decoder ($N = 32$ over GF(2))

Table 4.3: Average packets for different decoders ($N = 32$ over GF(2))

Strategy	FR	EF	greedy-(-1)		greedy-0		greedy-1		BE	
			LP I	LP II	LP I	LP II	LP I	LP II	LP I	LP II
100% PSR	33.59	32.71	30.83	32.04	30.83	32.04	30.83	32.04	30.83	32.04
95% BSR	33.59	32.71	30.11	31.96	30.11	31.96	30.11	31.96	30.11	31.96

EF strategy. The last four decoding strategies, though bring errors, recovers 96% of the packets and 99% of the bits at $m = 31$. In general, better BSR results are obtained compared to the PSR results. However, the PSR results of the last four strategies grows rapidly with m around 29 to 34. For example, though only about 25% of the packets are recovered successfully when $m = 29$, the values reaches about 65% for $m = 30$, which further grows to about 90% with $m = 31$. Note that the results are almost the same for the last four decoding strategies when $m \geq 27$, as the conditions that trigger the decoding of the four strategies are all satisfied.

Table 4.4: Average packets for different decoders ($N = 100$ over GF(2))

Strategy	FR	EF	BE	
			LP I	LP II
100% PSR	101.65	100.25	98.86	101.65
95% BSR	101.65	99.76	98.29	101.55

The average minimum numbers of packets required to achieve a PSR of 1 or a BSR of 0.95 are compared in Table 4.3. Compared to the FR strategy, the EF strategy requires about 0.8 less packets to achieve the same target PSR and BSR. However, the other four strategies with LP I saves about 2.7 packets to reach the same PSR and about 3 packets for a same BSR. The improvement with LP II is reduced, but the last four strategies still requires about 1.5 and 1.6 less packets for a same PSR and BSR, respectively.

Table 4.4 shows the average minimum numbers of packets for $N = 8192$ and a maximum of m being 100. Since the LP solver works column wisely, the performance is expected to be worse following increasing number of packet length N. Note that with this parameter settings, all the three greedy-(-1), greedy-0, and greedy-1 strategies have the same performance as the BE strategy, hence only the BE results are shown in Table 4.4. Also, to avoid large complexity from the HN decoder, the EF strategy here adopts an LP solver to estimate the minimum Hamming distance, where fractional numbers count for the Hamming weight of the codeword. As demonstrated in Table 4.4, the advantage of BE strategy over FR shrinks, where only about 3.3 less packets are required for BE to reach the same level of BSR, and even smaller advantage for the same PSR.

4.4 General LP Formulation over GF(2)

4.4.1 General LP Formulation with Arbitrary Parities

In [50], a new LP decoding algorithm was proposed by Yang *etc.* to decode binary codes, where both the number of linear constraints and variables are linear with respect to the maximum check node degree. Here, we generalize the approach such that linear codes with both even and odd parity checks can be decoded. A binary equation with odd parity check can be formulated into linear constraints as follows.

Suppose there is an odd parity check equation with d variables:

$$f_0 + f_1 + \cdots + f_{d-1} = 1,\ f_i \in \{0,1\},\ \text{for } i = 0, 1, \ldots, d-1. \tag{4.2}$$

We want to decompose this equation into a groups of equations with smaller number of variables to facilitate the LP formulation. The goal is to reach an odd parity check equation with only two variables, hence we decompose the equations in a recursive manner. We denote by $f_i^{(j)}$ the auxiliary variable x_i in the jth decomposition step. Naturally we set $f_i^{(0)} = f_i$ for $i = 0, 1, \ldots, d-1$ in Eq. (4.2).

If $d^{(0)} = d$ is an even number, i.e., $d^{(0)} = 2d^{(1)}$ for some nonzero positive integer $d^{(1)}$, we decompose Eq. (4.2) in the first step into

$$\begin{aligned} f_{2k}^{(0)} + f_{2k+1}^{(0)} + f_k^{(1)} = 0, \qquad & \sum_{k=0}^{d^{(1)}-1} f_k^{(1)} = 1 \\ k = 0, 1, \ldots, d^{(1)} - 1, \qquad & f_{2k}^{(0)}, f_{2k+1}^{(0)}, f_k^{(1)} \in \{0,1\}. \end{aligned} \tag{4.3}$$

If d is an odd number, i.e., $d^{(0)} = 2d^{(1)}+1$ for some nonnegative integer $d^{(1)}$, Eq. (4.2)

will be decomposed into

$$\begin{aligned} f_{2k}^{(0)} + f_{2k+1}^{(0)} + f_k^{(1)} = 0, \qquad & f_{d^{(0)}-1}^{(0)} + \sum_{k=0}^{d^{(1)}-1} f_k^{(1)} = 1 \\ k = 0, 1, \ldots, d^{(1)} - 1, \qquad & f_{2k}^{(0)}, f_{2k+1}^{(0)}, f_k^{(1)} \in \{0, 1\}. \end{aligned} \tag{4.4}$$

If the odd parity check equation in the current step contains more than two variables, the decomposition continues as in Eq. (4.3) or Eq. (4.4). Suppose $d^{(\ell-1)} > 2$ in step $\ell - 1$, then we obtain $d^{(\ell)} = \lfloor d^{(\ell-1)}/2 \rfloor$, and the decomposition in the ℓth step is

$$\begin{aligned} f_{2k}^{(\ell-1)} + f_{2k+1}^{(\ell-1)} + f_k^{(\ell)} = 0, \qquad & \sum_{k=0}^{d^{(\ell)}-1} f_k^{(\ell)} = 1, \\ k = 0, 1, \ldots, d^{(\ell)} - 1, \qquad & f_{2k}^{(\ell-1)}, f_{2k+1}^{(\ell-1)}, f_k^{(\ell)} \in \{0, 1\}, \end{aligned} \tag{4.5}$$

if $d^{(\ell-1)}$ is even, and is

$$\begin{aligned} f_{2k}^{(\ell-1)} + f_{2k+1}^{(\ell-1)} + f_k^{(\ell)} = 0, \qquad & f_{d^{(\ell-1)}-1}^{(\ell-1)} + \sum_{k=0}^{d^{(\ell)}-1} f_k^{(\ell)} = 1, \\ k = 0, 1, \ldots, d^{(\ell)} - 1, \qquad & f_{2k}^{(\ell-1)}, f_{2k+1}^{(\ell-1)}, f_k^{(\ell)} \in \{0, 1\}, \end{aligned} \tag{4.6}$$

if $d^{(\ell-1)}$ is odd.

The total number of recursions needed to reach an odd parity check equation with two variables is $\ell^* = \lceil \log_2(d/2) \rceil$. Hence after the ℓ^*th step, we replace the odd parity check equation with $0 < f_0^{(\ell^*)} + f_1^{(\ell^*)} < 1$, all other even parity check equations with linear constraints as that in [50], relax the integer constraints $f_k^{(\ell)} \in \{0, 1\}$ to $f_k^{(\ell)} \in [0, 1]$, and form a representation of Eq. (4.2) by linear constraints. Then given

an objective function, the decoding of a linear block code with both even and odd parity checks can be performed by linear programming.

4.4.2 Analysis

For an odd parity check equation with d variables, let $A(d)$ be the number of auxiliary variables introduced into the original nonlinear constraint, and $C(d)$ the number of groups of linear constraints with two or three variables. Then in the tree representation [50], there will be one constraint with two variables, and $C(d) - 1$ with three variables. Hence the total number of edges in the tree is

$$3(C(d)) - 1 = d + 2A(d), \tag{4.7}$$

while the total number of nodes is

$$3(C(d)) = d + A(d) + C(d). \tag{4.8}$$

Solving these two equations, we obtain

$$A(d) = d - 2, \qquad C(d) = d - 1, \tag{4.9}$$

linear with respect to the number of variables involved in the original equation.

Since each even parity check equation with three variables is represented by four linear constraints in [50], while only one is needed for the one with an odd parity, the total number of linear constraints in the LP formulation will be $4(C(d) - 1) + 1 = 4d - 7$. The total number of variables will be $d + A(d) = 2d - 2$. Compared

to the formulation with even parity checks in [50], one additional constraint and one additional auxiliary variable are needed for the odd parity case given the same number of original variables. linear network coding, our proposed decoders require fewer received packets to decoder and hence achieve higher throughputs and shorter delays.

4.5 LP Decoding of Nonbinary Linear Block Codes

4.5.1 Preliminaries and Notations

Suppose an (N, K, M) nonbinary linear block code $\mathcal{C}$ is defined over GF(q), where N is the code length, K the dimension of the code, and M the number of rows in the parity check matrix $\boldsymbol{H}$. Denote by $\mathcal{I} = \{0, 1, \ldots, N-1\}$ and $\mathcal{J} = \{0, 1, \ldots, M-1\}$ the column and row indicies of $\boldsymbol{H}$, respectively, and $\mathcal{I}_j$ the supporting set of the row vector $\boldsymbol{H}_j$. For each $j \in \mathcal{J}$, a single parity check code $\mathcal{C}_j$ is defined by $\mathcal{C}_j = \{\boldsymbol{b} = (b_i)_{i\in\mathcal{I}_j} : \sum_{i\in\mathcal{I}_j} b_i \boldsymbol{H}_{j,i} = 0\}$. Then for each codeword $\boldsymbol{c} \in \mathcal{C}$, we have $\boldsymbol{x}_j(\boldsymbol{c}) \in \mathcal{C}_j$, where $\boldsymbol{x}_j(\boldsymbol{c}) = (c_i)_{i\in\mathcal{I}_j}$.

For $\alpha \in$ GF(q), define a vector $\boldsymbol{\xi}(\alpha) = (\alpha(0), \alpha(1), \ldots, \alpha(q-1))$, where $\alpha(\beta) = 1$ if $\beta = \alpha$ and 0 otherwise, for $\beta \in$ GF(q). Hence $\boldsymbol{\xi}(\alpha)$ is a length-q binary vector with Hamming weight one, "pointing" to the value of α. Let us denote the set of images of the function $\boldsymbol{\xi}(\cdot)$ by Ω, hence $\boldsymbol{\xi} : \text{GF}(q) \to \Omega \subset \{0, 1\}^q$. Note that different from the mapping denoted by the same symbol in [21], the vector $\boldsymbol{\xi}(\alpha)$ always has a Hamming weight of one for any $\alpha \in$ GF(q), i.e., $\sum_{\beta\in\text{GF}(q)} \alpha(\beta) = 1$. Further for a vector $\boldsymbol{c} = (c_0, c_1, \ldots, c_{N-1}) \in \text{GF}(q)^N$, define $\boldsymbol{\Xi} : \text{GF}(q)^N \to \Omega^N \subset \{0, 1\}^{Nq}$ by $\boldsymbol{\Xi}(\boldsymbol{c}) = (\boldsymbol{\xi}(c_0)|\boldsymbol{\xi}(c_1)|\cdots|\boldsymbol{\xi}(c_{N-1}))$. Both mappings of $\boldsymbol{\xi}$ and $\boldsymbol{\Xi}$ are one-to-one

correspondence, hence their inverse exist respectively.

Example 3. *Let γ be a primitive element over GF(2^2) and a root of $\gamma^2+\gamma+1=0$. Hence elements in GF(2^2) can be represented by $0,1,\gamma,\gamma^2$, or binary vectors $(0,0),(1,0),(0,1),(1,1)$ given $(1,\gamma)$ as a basis set. Denote these elements by $0,1,2,3$, respectively. Then $\boldsymbol{\xi}(3)=(0,0,0,1)$, $\boldsymbol{\xi}^{-1}=(0,1,0,0)=1$, and $\boldsymbol{\Xi}(2,0,3)=(0,0,1,0|1,0,0,0|0,0,0,1)$.*

As an extension, for $\boldsymbol{f}=(\boldsymbol{f}_0|\boldsymbol{f}_1|\cdots|\boldsymbol{f}_{N-1})\in\mathbb{R}^{Nq}$, if $\boldsymbol{f}\in\Omega^N$, define $\boldsymbol{\Xi}^{-1}(\boldsymbol{f})=(\boldsymbol{\xi}^{-1}(\boldsymbol{f}_0),\boldsymbol{\xi}^{-1}(\boldsymbol{f}_1),\ldots,\boldsymbol{\xi}^{-1}(\boldsymbol{f}_{N-1}))$, and $\boldsymbol{\Xi}^{-1}(\boldsymbol{f})=\bar{\boldsymbol{c}}$ otherwise, where $\bar{\boldsymbol{c}}\in\mathrm{GF}(q)^N$ but $\bar{\boldsymbol{c}}\notin\mathcal{C}$. $\bar{\boldsymbol{c}}\in$ indicates a decoding failure.

Given a q-ary input memoryless channel, suppose the transition probability is $p(y|x)$, where $x\in\mathrm{GF}(q)$ ($q=2^m$), and $y\in\mathcal{Y}$. Define vector

$$\boldsymbol{\lambda}(y_i)=\left(0,\log\frac{p(y_i|0)}{p(y_i|1)},\log\frac{p(y_i|0)}{p(y_i|2)},\ldots,\log\frac{p(y_i|0)}{p(y_i|q-1)}\right),$$

and the ML decoding is to find

$$\begin{aligned}\hat{\boldsymbol{c}} &= \arg\max_{\boldsymbol{c}\in\mathcal{C}}\prod_{i=0}^{N-1}p(y_i|c_i)\\ &= \arg\min_{\boldsymbol{c}\in\mathcal{C}}\sum_{i=0}^{N-1}\boldsymbol{\lambda}(y_i)\boldsymbol{\xi}(c_i)^T\\ &= \arg\min_{\boldsymbol{c}\in\mathcal{C}}\boldsymbol{\Lambda}(\boldsymbol{y})\boldsymbol{\Xi}(\mathbf{c})^T. \qquad (4.10)\end{aligned}$$

Equivalently in our case, this is to find $\hat{\boldsymbol{c}}=\boldsymbol{\Xi}^{-1}(\hat{\boldsymbol{f}})$, where

$$\hat{\boldsymbol{f}}=\arg\min_{\boldsymbol{f}\in\mathcal{K}(\mathcal{C})}\boldsymbol{\Lambda}(\boldsymbol{y})\boldsymbol{f}^T, \qquad (4.11)$$

and $\mathcal{K}(\mathcal{C})$ is the convex hull of all points $\boldsymbol{f} \in \mathbb{R}^{Nq}$. Hence the ML decoding reduces to the minimization of a linear objective function over a polytope in $\mathbb{R}^{Nq}$. However, the number of variables and constraints for this LP problem is exponential with respect to the code length, hence relaxed LP formulations are proposed in [21] and [51] for nonbinary linear codes and that over GF(2^m), respectively.

4.5.2 LP Decoding of Nonbinary Linear Block Codes

In [21], linear programming was used to decode nonbinary linear codes with zero parity. We generalize this approach to decoding of nonbinary linear block codes with arbitrary parity checks. Suppose the nonbiary linear block code is defined by $\mathcal{C} = \{\boldsymbol{c} \in \mathrm{GF}(q)^n : \boldsymbol{Ac} = \beta\}$, where GF($q$) is the underlying field, and β is an arbitrary element in GF(q).

To solve the linear programming problem in 4.11, auxiliary variables $w_{j,\boldsymbol{b}}$ are introduced for $j \in \mathcal{J}, \boldsymbol{b} \in \mathcal{C}_j$ with constraints

$$\sum_{\boldsymbol{b} \in \mathcal{C}_j} w_{j,\boldsymbol{b}} = 1, \qquad \forall j \in \mathcal{J}, \tag{4.12}$$

$$f_i^{(\alpha)} = \sum_{\boldsymbol{b} \in \mathcal{C}_j, b_i = \alpha} w_{j,\boldsymbol{b}}, \ \forall j \in \mathcal{J}, \forall i \in \mathcal{I}_j, \forall \alpha \in \mathrm{GF}(q) \backslash \{0\}, \tag{4.13}$$

and

$$\sum_{\alpha \in \mathrm{GF}(q) \backslash \{0\}} f_i^{(\alpha)} \leq 1, \forall i \in \mathcal{I}, \tag{4.14}$$

with

$$0 \leq w_{j,\boldsymbol{b}} \leq 1, \forall j \in \mathcal{J}, \forall \boldsymbol{b} \in \mathcal{C}_j, \tag{4.15}$$

and

$$0 \leq f_i^{(\alpha)} \leq 1, \forall i \in \mathcal{I}, \forall \alpha \in \mathrm{GF}(q) \backslash \{0\}. \tag{4.16}$$

4.5.3 LP Decoding of Nonbinary Linear Codes over GF(2^m)

In [51], LP decoding of LDPC codes over GF(2^m) is proposed, where the number of constraints is linear in the size of the field $q = 2^m$. However, the LP relaxation becomes loose when $q \geq 8$.

Each element $z \in \mathrm{GF}(q) = \{0, 1, \ldots, q-1\}$ is mapped into a binary string of length q by $G : \mathrm{GF}(q) \rightarrow \mathcal{F}$ as $G(z) = (0, ..., \ldots, 0, 1, 0, \ldots, 0)$, where the only nonzero element is located as position z for $z = 0, 1, \ldots, q-1$, and $\mathcal{F} = \{(f^0, ..., f^{q-1}) \in [0,1]^q : \sum_{z=0}^{q-1} f^z = 1\}$. Define $G^n(\boldsymbol{z}) = (G(z_1), \ldots, G(z_n))$ for $\boldsymbol{z} \in GF(q)^n$ correspondingly. Then the objective function of the relaxed problem is

$$\tilde{L}(\{f_i\}, \boldsymbol{y}) = \sum_{i=1}^{n} \sum_{a=0}^{q-1} f_i^a \log W^m(y_i|a),$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ with $y_i = ((y_i)_1, \ldots, (y_i)_m)$ is the received vector, and $W^m(y_i|a) = \prod_{l=1}^{m} W((y_i)_l|(a)_l)$ is the transition probability of the channel.

Following similar notations as previous sections, let $i(j,l)$ denote the l-th element of $N(j)$ for $l = 1, 2, \ldots, |N(j)|$. For $r \in \mathbb{N}$, define $\mathcal{S}_r^e = \{\boldsymbol{s} = (s_1, s_2, \ldots, s_r) \in \mathrm{GF}(2)^r : \sum_{l=1}^{r} s_l = 1\}$, and $\mathcal{S}_r^o = \{\boldsymbol{s} = (s_1, s_2, \ldots, s_r) \in \mathrm{GF}(2)^r : \sum_{l=1}^{r} s_l = 0\}$, where the summation is over GF(2). Then the feasible region of the original LP relaxation problem in [49] is the set of points in $\mathcal{F}^n$ such that

$$\sum_{l=1}^{|N(j)|} f_{i(j,l)}^{s_l} \leq |N(j)| - 1$$

Table 4.5: Average packets for HN decoders ($N = 8$ over GF(2^2))

Strategy	FR	EF	greedy-(-1)	greedy-0	greedy-1	BE
100% PSR	8.52	7.02	8.52	8.52	8.52	8.52
95% BSR	8.52	7.02	8.52	8.52	8.52	8.52

is satisfied for all $j \in \mathcal{J}$ and $\boldsymbol{s} \in \mathcal{S}^e_{|N(j)|}$ for binary linear codes with even parity checks, while $\boldsymbol{s} \in \mathcal{S}^o_{|N(j)|}$ for codes with odd parity checks.

For codes over GF(2^m), define $(z)_B = \sum_{l \in B}(z)_l$ for $z \in$ GF(2^m) and $B \in \mathcal{B}$ $\emptyset$, where $\mathcal{B}$ is the set of nonempty sets of $\{1, 2, \ldots, m\}$. Without loss of generality, let us suppose the parity check corresponding to the j-th check node is 1, the multiplication identity, with binary representation $(1, 0)$, i.e., $(1)_1 = 1$ and $(1)_2 = 0$. It can be shown from the uniqueness of the inverse that equations with other nonzero parity checks can be transformed into equations with parity check 1 without changing the solutions. Then the feasible region of the LP relaxation is

$$\mathcal{P} = \bigcap_{j \in \mathcal{J}, \boldsymbol{s} \in \mathcal{S}_{|N(j)|}, B \in \mathcal{B}} \mathcal{P}_j^{B,\boldsymbol{s}},$$

where $\mathcal{P}_j^{B,\boldsymbol{s}}$ is the set of $\{f_i\} \in \mathcal{F}^n$ satisfying

$$\sum_{l=1}^{|N(j)|} \sum_{z:(H_{j,i(j,l)} \times_q z)_B = s_l} f^z_{i(j,l)} \leq |N(j)| - 1.$$

If the parity check is 0, then $\mathcal{S}_{|N(j)|} = \mathcal{S}^e_{|N(j)|}$ for any $B \in \mathcal{B}$. If the parity check is 1, then $\mathcal{S}_{|N(j)|} = \mathcal{S}^o_{|N(j)|}$ for $B = \{1\}$ and $B = \{1, 2\}$, and $\mathcal{S}_{|N(j)|} = \mathcal{S}^e_{|N(j)|}$ for $B = \{2\}$.

Table 4.6: Average packets for LP decoders ($N = 8$ over GF(2^2))

Strategy	EF		greedy-(-1)		greedy-0		greedy-1		BE	
	FLP	XLP	FLP	XLP	FLP	XLP	FLP	XLP	FLP	XLP
100% PSR	8.52	8.52	6.67	7.32	6.60	7.32	6.60	7.32	6.60	7.32
95% BSR	8.52	8.52	6.28	7.04	6.00	7.02	5.97	7.02	5.97	7.02

4.5.4 New LP Formulation for Nonbinary Linear Codes over GF(2^m)

Given a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{N-1}) \in \mathcal{C}$, each symbol $c_i \in \mathrm{GF}(2^m)$ can also be viewed as a length-m vector over GF(2), represented by $c_i = (c_{i,0}, c_{i,1}, \ldots, c_{i,m-1})$ for each $i \in \{0, 1, \ldots, N-1\}$. Hence the notation c_i can be a variable over GF(2^m) , or a vector over the base field throughout this chapter. Also, we write $(c_i)_\ell$ to indicate the ℓth bit of c_i when treated as a binary vector, i.e., $(c_i)_\ell = c_{i,\ell}$. Also, we fix $q = 2^m$, and will use GF(q) and GF(2^m) alternatively. Without loss of generality, we can denote elements in GF(q) by $0, 1, \ldots, q-1$, with 0 the addition identity. Let $\mathcal{B} = \{B : B \subseteq \{0, 1, \ldots, m-1\}\}$. Given an element $\alpha \in \mathrm{GF}(2^m)$, define $(\alpha)_B = \sum_{j \in B} (\alpha)_j$. Conversely, $(\alpha)_j = (\alpha)_B$ with $B = \{j\}$.

For nonbinary codes over GF(2^m), each parity check equation defined by each row of the parity check matrix can be represented by m equations over the base field. We aim to express the variables over the extension field by that over the base field, and then solve the corresponding binary constraints by existing LP formulations.

Suppose the check node degree of the parity check matrix $\boldsymbol{H}$ is d, and the d

nonzero elements in the jth row of $\boldsymbol{H}$ are $(h_{j,i_0}, h_{j,i_1}, \ldots, h_{j,i_{d-1}})$. Then the corresponding parity check equation is

$$h_{j,i_0}x_{i_0} + h_{j,i_1}x_{i_1} + \cdots + h_{j,i_{d-1}}x_{i_{d-1}} = \mathbf{0}, \tag{4.17}$$

which corresponds to m binary constraints

$$\begin{aligned}&(h_{j,i_0}x_{i_0} + h_{j,i_1}x_{i_1} + \cdots + h_{j,i_{d-1}}x_{i_{d-1}})_\ell = \\ &(h_{j,i_0}x_{i_0})_\ell + (h_{j,i_1}x_{i_1})_\ell + \cdots + (h_{j,i_{d-1}}x_{i_{d-1}})_\ell = 0,\end{aligned} \tag{4.18}$$

for $0 \leq \ell \leq m-1$. In addition, it also holds

$$\begin{aligned}&(h_{j,i_0}x_{i_0} + h_{j,i_1}x_{i_1} + \cdots + h_{j,i_{d-1}}x_{i_{d-1}})_B = \\ &(h_{j,i_0}x_{i_0})_B + (h_{j,i_1}x_{i_1})_B + \cdots + (h_{j,i_{d-1}}x_{i_{d-1}})_B = 0,\end{aligned} \tag{4.19}$$

for $B \in \mathcal{B}$, with a total of $2^m - 1$ binary constraints.

Next, we represent the constraints in (4.19) in terms of $f_i(\alpha)$'s. Define $t_{j,\ell,k} \stackrel{\text{def}}{=} (h_{j,i_k}x_{i_k})_{B_\ell}$, the k-th $0 \leq k \leq d-1$ bit participating in the ℓth binary constraint of Eq. (4.19), defined by the jth parity check equation Eq. (4.17).

Then

$$\begin{aligned}t_{j,\ell,k} \stackrel{\text{def}}{=} (h_{j,i_k}x_{i_k})_{B_\ell} = &\sum_{\alpha:(h_{j,i_k}\alpha)_{B_\ell}=1} f_i(\alpha), \\ &\forall j \in \{0, 1, \ldots, M-1\},\end{aligned} \tag{4.20}$$

where

$$\sum_{\alpha \in \mathrm{GF}(q)} f_i(\alpha) = 1, 0 \le f_i(\alpha) \le 1, \tag{4.21}$$
$$\forall i \in \{i_0, i_1, \ldots, i_{d-1}\}, \forall j \in \{0, 1, \ldots, M-1\}.$$

Hence the ℓth ($\ell \in \{0, 1, \ldots, 2^m - 1\}$) constraint of Eq. (4.19) is

$$\forall j, \forall \ell, \qquad \sum_{k=0}^{d-1} t_{j,\ell,k} \equiv 0, \qquad 0 \le t_{j,\ell,k} \le 1, \tag{4.22}$$

and the total number of $t_{j,\ell,k}$ is $M(2^m - 1)d$.

Eq. (4.22) is a binary parity check constraint, and can be solved by any of the linear programming approaches introduced in [49] and [50], where the number of linear equations or variables are in the order of 2^d (in [49]) or $O(d)$ (in [50]), compared to $O(q^d)$ in [21].

The variables $\boldsymbol{f}$ and $\boldsymbol{t}$'s together with their constraints form a polytope denoted by $\mathcal{Q}$. Depending on the specific binary linear programming used to solve (4.22), $\mathcal{Q}$ can be different.

Example 4. *Using the same GF(2^2) as in Example 3. Suppose a code over GF(2^2) is* $\mathcal{C} = \{(0,0,0), (1,0,\gamma), (\gamma,0,\gamma^2), (\gamma^2,0,1)\}$*, defined by*

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_0 \\ \boldsymbol{H}_1 \end{pmatrix} = \begin{pmatrix} 1 & \gamma & \gamma^2 \\ \gamma & 1 & 1 \end{pmatrix}.$$

Based on Eq. (4.19)-(4.22), we have

$$\sum_{\alpha:(\alpha)_{B_\ell}=1} f_0(\alpha) + \sum_{\alpha:(\gamma\alpha)_{B_\ell}=1} f_1(\alpha) + \sum_{\alpha:(\gamma^2\alpha)_{B_\ell}=1} f_2(\alpha) = 0,$$

corresponding to $\boldsymbol{H}_0$*, where* $B_0 = \{0\}, B_1 = \{1\}, B_2 = \{0,1\}$*. Specifically,*

$$\begin{aligned}&(f_0(1) + f_0(\gamma^2)) + (f_1(\gamma) + f_1(\gamma^2)) + (f_2(1) + f_2(\gamma))\\ &= t_{0,0,0} + t_{0,0,1} + t_{0,0,2} = 0\end{aligned}$$

for $\ell = 0$ *(*$B_\ell = \{0\}$*),*

$$\begin{aligned}&(f_0(\gamma) + f_0(\gamma^2)) + (f_1(1) + f_1(\gamma)) + (f_2(1) + f_2(\gamma^2))\\ &= t_{0,1,0} + t_{0,1,1} + t_{0,1,2} = 0\end{aligned}$$

for $\ell = 1$ *(*$B_\ell = \{1\}$*), and*

$$\begin{aligned}&(f_0(1) + f_0(\gamma)) + (f_1(1) + f_1(\gamma^2)) + (f_2(\gamma) + f_2(\gamma^2))\\ &= t_{0,2,0} + t_{0,2,1} + t_{0,2,2} = 0\end{aligned}$$

for $\ell = 2$ *(*$B_\ell = \{0,1\}$*).*

For codeword $\boldsymbol{c} = (1, 0, \gamma)$*,* $\boldsymbol{f} = \Xi(\boldsymbol{c}) = (\boldsymbol{f_0}, \boldsymbol{f_1}, \boldsymbol{f_2}) = (0,1,0,0|1,0,0,0|0,0,1,0)$*,* $t_{0,0,0} = 1, t_{0,0,1} = 0, t_{0,0,2} = 1, t_{0,1,0} = 0, t_{0,1,1} = 0, t_{0,1,2} = 0, t_{0,2,0} = 1, t_{0,2,1} = 0, t_{0,2,2} = 1$*, and the previous binary equations are satisfied. Similarly,* $\boldsymbol{c}$ *satisfies the other two binary constraints corresponding to* $\boldsymbol{H}_1$*, and so are other three codewords.*

Remark 1. *In our approach, $(\alpha)_{B_\ell} = 1$ works exactly the same as the function of $(\alpha)_B = s$ with $s = 1$ in [51]. Further, if the equivalent polytope $\bar{Q}$ represented in Eq. (9) of Section III.C in [49] is used to get linear constraints for binary equations in Eq. (4.19), our approach leads to linear constraints as that in [51]. Hence [51] can be viewed as a special case of our formulation.*

On the other hand, in our approach, the introduction of $t_{j,\ell,k}$'s enable explicit expression of the value of each bit participating in the binary constraints as demonstrated in (4.19). As a result, any LP decoding algorithm for binary codes can be adopted to formulate nonbinary linear codes decoding into a linear programming problem.

The new formulation also features the ML certificate property, as shown below.

Proposition 1. *There is a one-to-one correspondence between each codeword and each integral point of $(\boldsymbol{t})$ in $\mathcal{Q}$.*

Proof. Based on previous results on binary LP formulation, it only needs to show the correspondence between codewords and integral $(\boldsymbol{f}, \boldsymbol{t})$'s in $\mathcal{Q}$.

First consider a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{N-1})$. For each $i \in \{0, 1, \ldots, N-1\}$, we have $f_i(\alpha) = 1$ if $\alpha = c_i$, and $f(\beta) = 0$ for all other $\beta \neq \alpha$, hence constraint (4.21) is satisfied. Then, $t_{j,\ell,k} = \sum_{\alpha:(h_{j,i_k}\alpha)_{B_\ell}=1} f_i(\alpha) = f_i(c_i)1_{(h_{j,i_k}c_i)_{B_\ell}=1} = (h_{j,i_k}c_i)_{B_\ell}$, where $1_E = 1$ if the event E is true, and 0 other wise.

As a result, $\sum_{k=0}^{d-1} t_{j,\ell,k} = \sum_{k=0}^{d-1}(h_{j,i_k}c_i)_{B_\ell} \equiv 0$ given $\boldsymbol{c}$ a codeword, and Eq. (4.20) and Eq. (4.22) are satisfied. Once Eq. (4.22) holds, we can find an integral point in the polytope corresponding to each LP formulation as that shown in [49] and [50].

Conversely, given an integral vector of $\boldsymbol{f} = (\boldsymbol{f}_i)$ with $f_i(\alpha) \in \{0, 1\}$ that satisfies $\sum_{\alpha \in \mathrm{GF}(q)} f_i(\alpha) = 1$, let $\boldsymbol{c} = \Xi^{-1}(\boldsymbol{f})$, hence $c_i = f_i(\alpha)$ with $f_i(\alpha) = 1$. Then $t_{j,\ell,k} =$

$\sum_{\alpha:(h_{j,i_k}\alpha)_{B_\ell}=1} f_i(\alpha) = f_i(c_i)1_{(h_{j,i_k}c_i)_{B_\ell}=1} = (h_{j,i_k}c_i)_{B_\ell}$, leading to $\sum_{k=0}^{d-1}(h_{j,i_k}c_i)_{B_\ell} = 0$ for every $B \in \mathcal{B}$. Specifically, it is true for $B = \{j\}$ where $j = 0, 1, \ldots, m-1$. Equivalently we have $\sum_{i\in\mathcal{I}_j} h_{j,i}c_i = \mathbf{0}$, and $\boldsymbol{c}$ is a local codeword of $\mathcal{C}_j$ for every $j \in \mathcal{J}$, hence a codeword of $\mathcal{C}$. □

Proposition 2. *Whenever the cross LP decoder outputs a codeword $\boldsymbol{c} \in \mathcal{C}$, $\boldsymbol{c}$ is the ML codeword.*

The proof is straightforward as that in [49].

4.5.5 Simulation Results

We performed simulations of network coding over GF(2^2) for the six different strategies, while linear programming is performed by both extended Flanagan's approach (FLP) in Section 4.5.2 and our new algorithm (XLP) in Section 4.5.4. Again, we adopt the toy model with $n = 8, N = 8, cw = 2$, and m ranging from 1 to 15. $\boldsymbol{A}$ is generated randomly in the sense that each element takes one of the four symbols in GF(2^2) with equal probability. For each decoder, its packet- and bit-level success, failure, or error rate is calculated over 100,000 generations. It should be noted that because of the mapping $\boldsymbol{\Xi}$ in Section 4.5.2, decoding failure will be reported in our simulations when the two LP decoders output fractional results.

In Fig. 4.7 and Fig. 4.8, respectively, the packet- and bit-wise fraction of decoding success, failures, and errors of Hamming norm decoders are represented by green, yellow, and red bars. Similarly, Fig. 4.9 and Fig. 4.10, respectively, compare the packet- and bit-wise fraction of decoding success, failures, and errors of the FLP decoder. Due to similarities, bar figures of the XLP results are not displayed here. In

all figures, for each value of m, the six bars represent, from left to right, the full rank, error-free, greedy-(-1), greedy-0, greedy-1, and best-effort strategies, respectively. In order to measure and compare the throughput and delay of linear network coding with these decoders, the average minimum numbers of packets required to achieve a packet success rate (PSR) of 1 or a bit success rate (BSR) of 0.95 are compared in Table 4.5 and 4.6, where the EF, greedy-ℓ, and BE decoders use HN decoding algorithm in Table 4.5, and their LP results including both the FLP and the XLP are shown in Table 4.6.

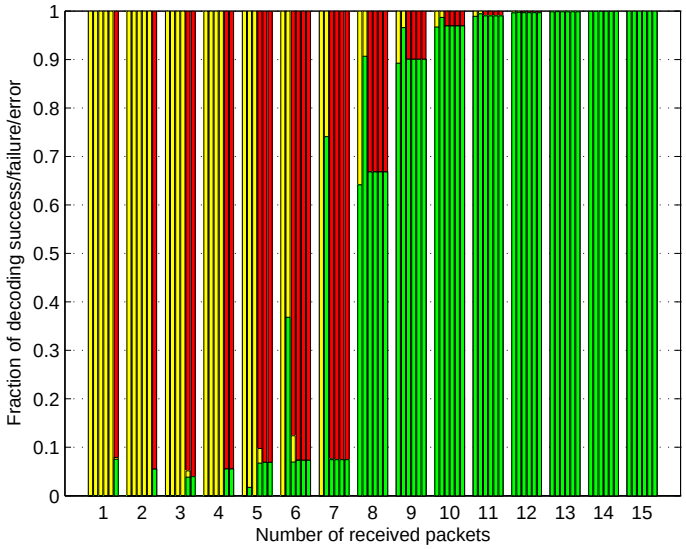

Figure 4.7: Packet-level performance of HN decoders ($N = 8$ over GF(2^2))

Generally speaking, better results are obtained at bit-level compared to that in the packet-level, especially for the HN decoders at Fig. 4.7 and Fig. 4.8. The LP decoder significantly reduces both the PER and the BER as shown in Fig. 4.9 and Fig. 4.10, compared to the HN decoders, but the cost is larger failure rates. However, LP failure decreases as m increases, leading to better PSR and BSR results than

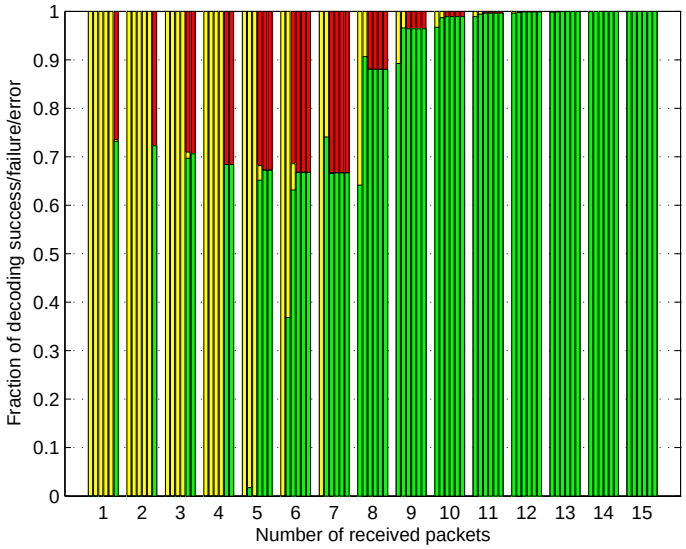

Figure 4.8: Bit-level performance of HN decoders ($N = 8$ over GF(2^2))

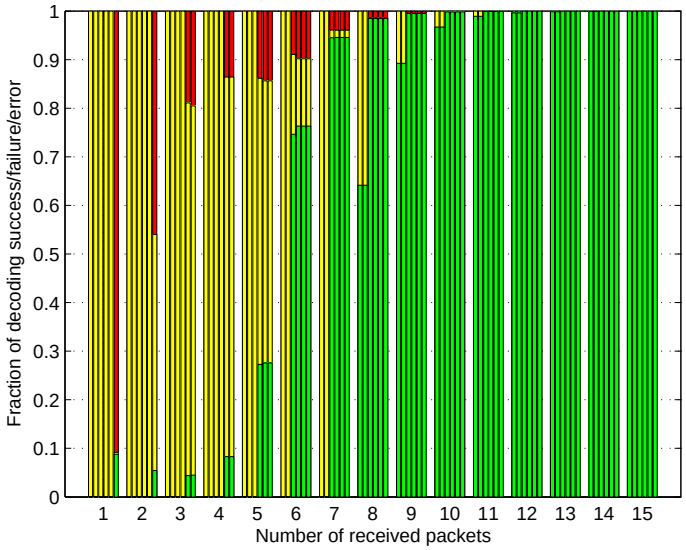

Figure 4.9: Packet-level performance of FLP decoder ($N = 8$ over GF(2^2))

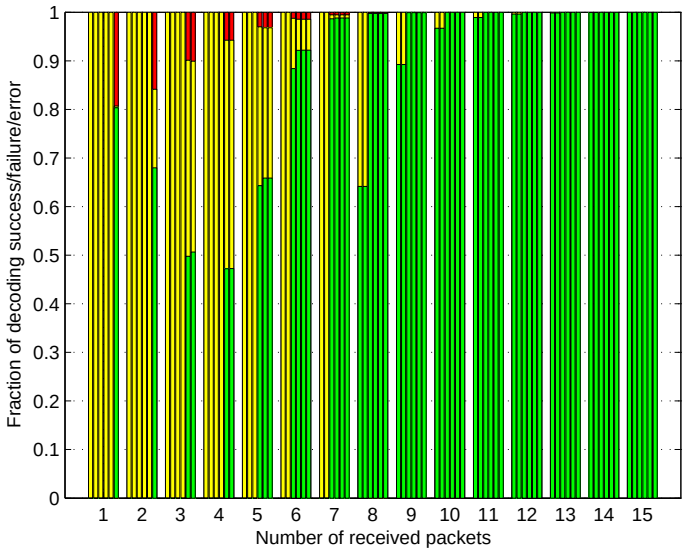

Figure 4.10: Bit-level performance of FLP decoder ($N = 8$ over GF(2^2)

the HN decoders for the last four strategies. Hence the average numbers of packets required by the LP decoder to achieve the same PSR and BSR are smaller than that by the HN decoder. As shown in Table 4.5 and 4.6, the BE strategy, when using the FLP decoder, only requires less than 6 packets to achieve a BSR of 0.95, which is 2.5 less than using HN decoder. On the other hand, improvement from the XLP decoder is about 1.5, due to larger fractional outputs compared to the FLP decoder. Also because of larger failure rate, the EF strategy does not benefit from the LP solver, as the minimum distances are underestimated.

Fig. 4.11 and Fig. 4.12 show packet- and bit- level performance of the XLP decoder, respectively. Compared to the corresponding results from the FLP decoder as shown in Fig. 4.9 and Fig. 4.10, the XLP suffers from a higher rate of decoding failure from fractional results, especially when m is small. On the other hand, the advantage is a tradeoff of a lower decoding error rate. As m increases, the difference

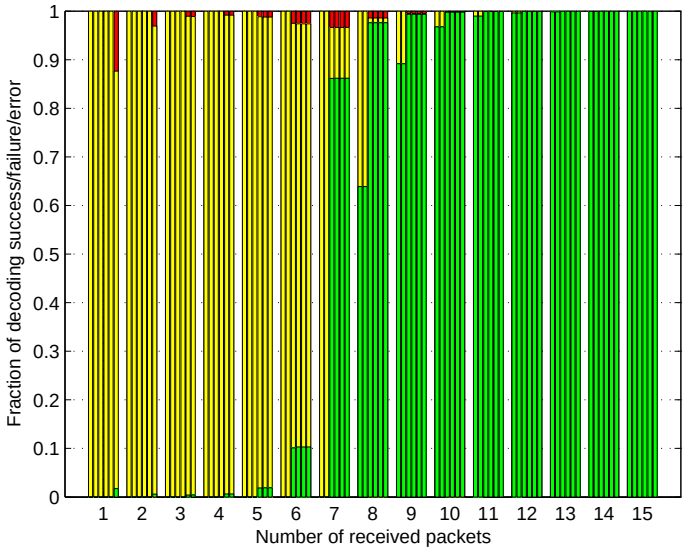

Figure 4.11: Packet-level performance of XLP decoder ($N = 8$ over GF(2^2))

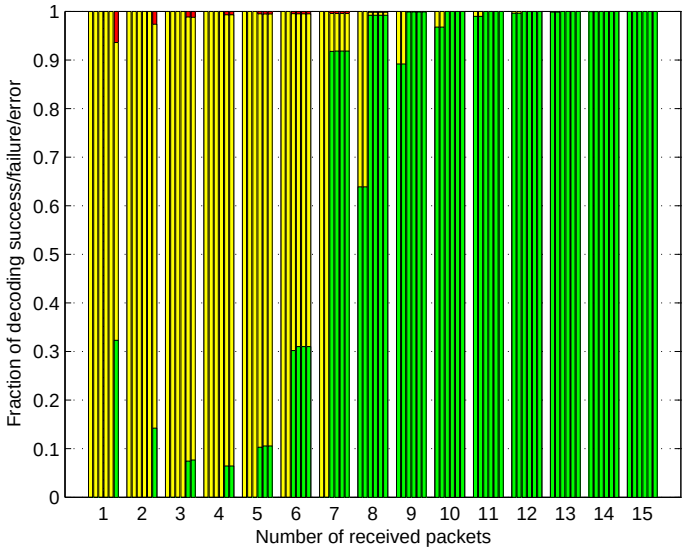

Figure 4.12: Bit-level performance of XLP decoder ($N = 8$ over GF(2^2))

between the two LP decoders becomes negligible.

4.6 Conclusion

In this chapter, we propose two classes of rank deficient decoders for network coding, where the receiver can start decoding even if the received packets are not of full rank. As a result, reduced delays and improved throughput can be achieved. Taking advantage of data sparsity, we first use Hamming norm decoder for rank deficient decoding, and then introduce linear programming decoders to handle bigger data. We also propose LP formulations with reduced complexity when network coding is performed over a large field. Simulation results show that rank deficient decoders require less received packets than the traditional full rank decoders to achieve the same packet/bit success rate.

Chapter 5

Distributed Storage Code Constructions from A Vector Space Approach

5.1 Introduction

In distributed storage systems (DSSs), data is spread across nodes in the network, while users are geographically dispersed. To avoid data loss from node failures, coding techniques such as replication and erasure codes [52] are employed to create redundancy for two types of data recovery [19]. First, a user (data collector) must be able to retrieve the original data by connecting to a certain number of storage nodes in the system, called *data reconstruction*. Second, data stored in a failed node should be recovered and stored in a new replacement node by contacting other nodes in the network, called *data repair*.

Compared to repetition schemes, erasure codes such as maximum distance separable (MDS) codes can be used to maximize protection against data reconstruction [19], which is equivalent to correction of erasures by traditional MDS codes. However, data repair is a new challenge, as node failure is a routine rather than an exception for larger scale DSSs such as cloud and peer-to-peer storage systems. Also, nodes carrying partial information may join or leave the network dynamically. Hence new MDS codes should be devised for data repair with low cost.

To measure the performance of data repair, different metrics have been proposed, such as *repair bandwidth*, the total number of symbols transmitted in the network, and *repair locality*, the number of nodes to be contacted, during the repair process. Codes aiming at optimizing these two metrics are regenerating codes [4] and locally repairable codes [53], respectively. In addition, repair of low computational complexity is also desirable for the information to be recovered locally in an efficient manner at a new replacement node.

Regenerating codes reduce the bandwidth consumed to repair a failed node, by contacting more nodes but only downloading part of information stored in each node. It was shown in [4] that there exists a tradeoff between repair bandwidth and storage capacity of each node, and codes that achieve the tradeoff curve are called *regenerating codes*. In particular, codes that achieve the minimal storage are called *minimum storage regenerating* (MSR) codes. Various repair models have been proposed in literature, including functional repair (see, for example, [4]), exact repair [54][55], and the hybrid repair [56]. *Exact repair* recovers the original symbols in failed nodes, and is preferred in some scenarios. MSR codes achieving this goal are called *exact-MSR* codes [54] [55] [57]. Recently, exact-MSR codes have been

proposed based on interference alignment [54, 58] and a product-matrix approach [59].

Both traditional MDS codes and MSR codes are optimal in terms of storage assumption. However, existing constructions of both codes require decoding of the original message before data repair. Hence the computational complexity of data repair has the same magnitude as that of data reconstruction, which usually involves matrix inversion (see, for example, [54, 58, 59]).

For local repair, as pointed out in [60], the number of nodes involved is closely related to the disk input/output (I/O) overhead, which is the main performance bottleneck in the repair problem. Hence various codes have also been proposed to reduce repair locality, such as scalar linear codes [53][61] and vector codes [60] [62] [63]. In [53][61], extra parity constraints are introduced into encoded symbols of an MDS code to enhance repair locality, and a trade-off is demonstrated between the minimum distance and the repair locality of the resulting code. The trade-off is extended in [60] to accommodate vector codes for local repair of one failed node, and explicit code construction based on a two-layer encoding structure is proposed for a specific set of parameters. Vector codes capable of repairing more than one failed node locally at the same time are proposed in [63]. Those vector codes with local repair property are called locally repairable codes (LRCs), and LRCs achieving the optimal minimum distance are said to be optimal. Optimal LRCs are also constructed in [63] featuring a two-layer encoding structure, where Gabidulin codes [12] are used in the first layer encoding. However, the adoption of Gabidulin codes leads to a finite field size growing exponentially with the number of nodes in the DSS.

In this dissertation,we propose three code constructions for DSSs with different repair features. We view a code for DSSs as an linear array or vector code, where each coded symbol stored in the nodes is a linear combination of the message symbols. All the coefficient vectors form a linear transformation matrix that maps the message space into the code space. Hence we call the vector space spanned by the coefficient vectors a linear transformation (LT) space, and those corresponding to encoded symbols in each node span a subspace of the LT space. For simplicity, we also say the subspace is spanned by a node. Under this setting, data reconstruction (recovery of the message space) from a set of nodes is equivalent to retrieving the LT space from the union of subspaces spanned by the these nodes. Data repair is to recover the subspace spanned by a node from union and/or intersections of subspaces spanned by other nodes.

We first propose a new family of MDS code that has low computational complexities for both data repair and reconstruction. Encoding can be performed from evaluation of linearized polynomials, whose linear mapping property leads to low-complexity data repair by simple linear combinations of participating symbols. Meanwhile, data reconstruction, or decoding of our new MDS codes can be conducted efficiently by the interpolation algorithm over linearized polynomial ring [64][65]. Hence data reconstruction has a quadratic complexity with respect to code parameters, compared to a cubic complexity of some traditional MDS codes and existing MSR codes [54, 58, 59].

Second, we propose optimal LRC codes with a two-layer encoding structure, and present a special property for an MDS code to be used in the first layer to ensure data reconstruction. In particular, we prove that Gabidulin codes satisfy this property,

and obtain explicit optimal LRC codes. It turns out that this LRC code is the same as that in [63], hence it is a special case of our construction. Our construction has flexible structures, and can also lead to comparable codes proposed in [60] given the same set of parameters. However, compared to the construction in [60], our codes are more storage efficient, as they are constructed based on array codes approach instead of traditional scalar codes. Meanwhile, our codes have smaller penalty when successive reads are performed in the scenario of degraded reads.

Finally, we construct MSR codes from our vector space analysis. Since the original message space is uniquely defined by the LT space, data reconstruction is nothing but collecting basis vectors of the LT space from a subset of nodes. Inversely, the coding procedure is to distribute different sets of basis vectors into subsets of nodes. Since intersection properties of subspaces quickly become intractable as dimension grows, we only consider MSR code construction with small parameters. An interesting outcome of this construction is that data repair is automatically guaranteed, though we do not apply special rules for data repair during the construction process.

The rest of this chapter is organized as follows. Section 5.2 provides some preliminaries on distributed storage coding schemes such as MSR and LRC codes. Section 5.4, 5.5 and 5.6 present our new MDS, optimal LRC, and MSR codes, respectively. Section 5.7 provides some concluding remarks and possible extensions of our work.

5.2 Preliminary

Suppose there are n nodes in a DSS, each having a storage capacity of α. We also say there are α storage units in each node. A message file of M symbols over GF(q) (q is a prime power) will be encoded by a distributed storage code into $n\alpha$ coded symbols, stored in the $n\alpha$ storage units, α encoded symbols in each node. Data reconstruction requires that the original message file should be recovered from the $k\alpha$ encoded symbols stored in any k nodes, where $0 < k < n$ and $k\alpha \geq M$. Data repair is to recover the α encoded symbols in any node from any other r nodes, where $0 < r < n$.

Depending on different values of r, different metrics have been proposed in the literature to measure the repair performance, repair bandwidth and repair locality, termed as locally repairable codes [53] and regenerating codes [4], to be addressed in Section 5.5 and Section 5.6, respectively. In this dissertation, we also consider computational complexity for data repair, and propose new MDS codes with low complexity in Section 5.4.

5.2.1 Maximum Distance Separable (MDS) Codes

Intuitively, an array codes (or vector codes, we will use these two terms alternatively in this chapter) can be used to provide protection again both kinds of data loss. Each codeword of a linear array code is a two-dimensional array, instead of a vector as for linear block codes. Suppose an array code $\mathcal{C}$ encodes the M symbols into a codeword $\boldsymbol{c} = (c_{i,j})$, with each coded symbol $c_{i,j} \in$ GF(q) a linear combination of the message symbols, and stored in unit j of node i for $i \in [n]$ and $j \in [\alpha - 1]$, as shown in Fig.

5.1. Throughout this chapter, we use $[n]$ to denote the set $\{0, 1, \ldots, n-1\}$.

Array codes can also be obtained from linear block codes over the extension field GF(q^α). As shown in Fig. 5.1, we denote $\boldsymbol{c}_i^\perp = (c_{i,0}, c_{i,1}, \ldots, c_{i,\alpha-1})$ for $i \in [n]$, and write $\boldsymbol{c} = (\boldsymbol{c}_0^\perp, \boldsymbol{c}_1^\perp, \ldots, \boldsymbol{c}_{n-1}^\perp)$. Then an (n, k) linear block code over GF(q^α) can be used to generate coded symbols $\boldsymbol{c}_i^\perp$, a length-α vector over GF(q), whose elements are then stored in node i. The minimum distance d of the array code is then defined to be the Hamming distance of this (n, k) code over GF(q^α), see [66] and the references therein.

Equivalently, the minimum distance d of $\mathcal{C}$ can also be defined to be the minimum number of erased nodes so that the entropy of the non-erased coded nodes is strictly less than M [60][63]. That is,

$$d = n - \max_{S:H(\boldsymbol{c}_S^\perp)<M} |S|, \tag{5.1}$$

where $S \subseteq [n]$ and $\boldsymbol{c}_S^\perp = \{\boldsymbol{c}_i^\perp : i \in S\}$. The code $\mathcal{C}$ is referred to as an (n, α, M, d) array code. An (n, α, M, d) array code is called an maximum distance separable (MDS) array code if $d = n - k + 1$ [66][63].

MDS array codes correct up to $n - k$ erasures, hence can be used in DSS to ensure data reconstruction from up to $n - k$ failed nodes. Based on the previous analysis, we can always use traditional MDS codes such as an (n, k) Reed-Solomon codes over GF(q^α) to get MDS array codes.

In fact, the construction can be further simplified into the base field GF(q). We denote $\boldsymbol{c}_j = (c_{0,j}, c_{1,j}, \ldots, c_{n-1,j})$, a length n vector over GF(q) for $j \in [\alpha]$, as shown in Fig. 5.1. If each $\boldsymbol{c}_j$ is a codeword of an (n, k) MDS code over GF(q), we can also

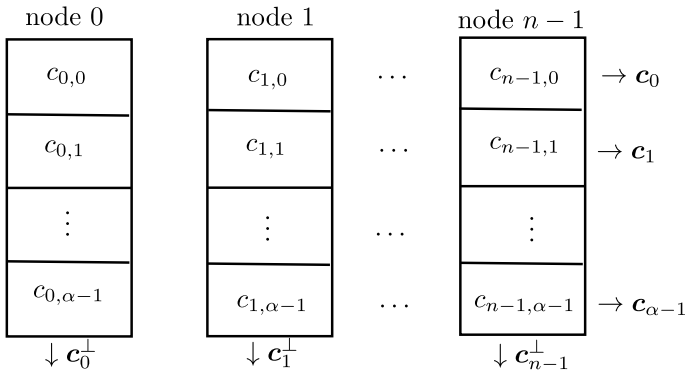

Figure 5.1: MDS codes in DSS

recover the original data from up to $n-k$ node failures. Note that in this case, we don't even have to ask each $\boldsymbol{c}_j$ is from the same MDS code, though it is preferred for simplicity.

As a result, the design of MDS array code over GF(q^α) is reduced to design of (n,k) MDS code over GF(q), and we will construct new MDS codes with low computational complexity for both data reconstruction and data repair in Section 5.4.

5.2.2 Locally Repairable Codes (LRC)

Locally repairable codes [53] aims at a small number of nodes to be contacted in data repair, to reduce system I/O overhead [60]. This type of data repair is called local repair, and the number of node contacted, $0 < r < k$, is *repair locality*.

In [53], local repair was originally considered for scalar codes, i.e., traditional linear block code, which was later extended to vector codes [60] suitable for DSSs. Since scalar codes can be viewed as vector codes with dimension one for each encoded elements, we use vector codes uniformly in this chapter.

Using same notations in Section 5.2.1, if for each coded symbol $\boldsymbol{c}_i^{\perp}$ with $i \in [n]$

of a codeword $\boldsymbol{c} \in \mathcal{C}$, there exists a set of nodes $\Gamma(i) \subseteq [n]$ such that 1) $i \in \Gamma(i)$; 2) $|\Gamma(i)| \leq r + \delta - 1$; and 3) minimum distance of $\mathcal{C}|_{\Gamma(i)}$ is at least δ, where r, δ are positive integers and $\mathcal{C}|_{\Gamma(i)}$ is the code obtained by restricting $\mathcal{C}$ over $\Gamma(i)$, then $\mathcal{C}$ is said to have (r, δ) locality [63]. Note that the (r, δ) locality indicates that each node $i \in [n]$ can be expressed as a function of at most r other elements $j \in \Gamma(i) \backslash \{i\}$, a property called *locally repairable*, and $\Gamma(i)$ is referred to as a *local repair group*. The (n, α, M, d) vector code $\mathcal{C}$ is then called a *locally repairable code* (LRC) [60][63], denoted as $(n, \alpha, M, d; r, \delta)$ LRC $\mathcal{C}$.

It is established in [67] that the minimum distance of an $(n, \alpha, M, d; r, \delta)$ LRC code is bounded by

$$d \leq n - \left\lceil \frac{M}{\alpha} \right\rceil + 1 - \left(\left\lceil \frac{M}{r\alpha} \right\rceil - 1 \right) (\delta - 1), \tag{5.2}$$

and codes attaining this bound are said to be optimal. When δ is fixed at 2, the bound in Eq. (5.2) reduces to $d \leq n - \lceil \frac{M}{\alpha} \rceil - \lceil \frac{M}{r\alpha} \rceil + 2$, which was first proved in [60].

Note that if we let $k = \lceil \frac{M}{\alpha} \rceil$ and $\delta = 2$, we'll obtain

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2,$$

which is exactly the same bound for scalar codes in [53], hence the results of scalar and vector codes are consistent.

The upper bound in Eq. (5.2) is proved in [67], based on an iterative algorithm that finds a set S in Eq. (5.1) in a fast way, and bounds the minimum distance d accordingly. Generally speaking, in each iteration, the algorithm picks a node and adds its local repair group into the current set S. If this group has at least $\delta - 1$

nodes outside the current S, then S is updated to accommodate the newly added nodes. The iteration carries on till the set S provides entropy $\lceil \frac{M}{\alpha} \rceil \alpha - \alpha$.

A two-layer encoding scheme is used in [63] to construct LRC codes that reach the optimal minimum distance in Eq. (5.2), based on Gabidulin codes [12] and MDS array codes. When $\delta = 2$, a similar two-layer encoding approach is also proposed to obtain optimal LRC codes with parameters that satisfy $(r+1)|n$ and $r+1 = \alpha$ in [60].

5.2.3 Minimum Storage Regenerating (MSR) Codes

If β encoded symbols are downloaded from each of the r supporting nodes in the repair process, a repair bandwidth of $\gamma = r\beta$ will be consumed in the network. Clearly, if we can reconstruct the entire message file from any k nodes, we can always re-encode and repair any node in the DSS. However, this approach costs a total bandwidth of $k\alpha \geq M$ in order to repair one node of capacity α. It is shown in [4] that by connecting to more nodes ($r \geq k$), less bandwidth will be needed to conduct data repair, and there exists an optimal tradeoff between α, the storage per node, and γ, the bandwidth to repair one node. Codes that attain the optimal tradeoff curve are called *regenerating codes*.

In particular, the extreme point with the smallest α in this curve corresponds to a *minimum-storage regenerating* (MSR) code, with

$$(\alpha_{\text{MSR}}, \gamma_{\text{MSR}}) = \left(\frac{M}{k}, \frac{Mr}{k(r-k+1)} \right). \tag{5.3}$$

As pointed out in [59], one can always construct an MSR code with $\beta \geq 1$ if one can

construct one with $\beta = 1$. Hence we will assume $\beta = 1$ in the rest of this chapter, resulting in $\alpha = r - k + 1$ and $M = k\alpha$, denoted an $(n, \alpha, M; k, r)$ MSR code. Note that according to Eq. 5.2, MSR codes are also MDS array codes.

Table 5.1: A $(4, 2, 4; 2, 3)$ MSR Code

node 0	node 1	node 2	node 3
m_0	m_2	$m_0 + m_2$	$m_1 + m_2$
m_1	m_3	$m_1 + m_3$	$m_0 + m_2 + m_3$

Table 5.1 presents a $(4, 2, 4; 2, 3)$ MSR code over GF(2), where m_0, m_1, m_2, m_3 are the message symbols. It can be verified that any $k = 2$ nodes suffice to recover the four message symbols, and any $r = 3$ nodes repair the other node. For example, when node 3 fails, the other three nodes will be contacted, and m_1 from node 0 and m_2 from node 1 will recover the first symbol, while m_1 from node 0 and $m_0 + m_1 + m_2 + m_3$ from node 2 recovers the second symbol in node 3. Note that node 2 passes $m_0 + m_1 + m_2 + m_3$, a linear combination of its encoded symbols for the repair. Hence even only β symbols are to be transmitted in the network, up to α symbols may have to be accessed from each helping node.

5.3 DSS Coding from Vector Space Approach

In this dissertation, we construct linear array codes for DSSs, viewed from a vector space's perspective. Since each coded symbol of a linear array code is a linear combination of the message symbols, it can be represented by a vector of coefficients. All the coefficient vectors compose of a matrix that defines the linear transformation from the message space into the code space. Accordingly, we call the space spanned

by the coefficient vectors the linear transformation (LT) space, and vectors from one node span a subspace of the LT space. To recover the message space from a subset of nodes, submatrix of the LT matrix must be invertible, which is equivalent to the retrieving of the LT space from summation of subspaces spanned by the corresponding nodes.

To facilitate the illustration of this vector space approach, we first represent the array code by that of traditional linear block codes, by aligning the coded symbols $c_{i,j}$'s into a one-dimensional vector. For example, we group the encoded symbols within one node together following the order of the storage unit, and write $\boldsymbol{c} = (c_{0,0}, c_{0,1}, \ldots, c_{0,\alpha-1}, c_{1,0}, \ldots, c_{1,\alpha-1}, \ldots, c_{n-1,\alpha-1})^T$, where T is the matrix transpose operation. This is an $(n\alpha, M)$ linear block code, and the codewords span an M-dimensional subspace of GF($q^{n\alpha}$) over GF(q).

Let $\boldsymbol{m} = (m_0 m_1 \ldots m_{M-1})^T$ be the message vector. Suppose $\boldsymbol{c} = (c_{i,j})^T = (\boldsymbol{g}_{i,j}^T \boldsymbol{m})^T$ is the corresponding codeword, where $\boldsymbol{g}_{i,j}$ an M-dimensional column vector called a *generator vector* for $i \in [n]$ and $j \in [\alpha]$. We can write a generator matrix G of size $M \times n\alpha$ for $\mathcal{C}$ to be

$$G_{M \times n\alpha} = (\boldsymbol{g}_{0,0} \boldsymbol{g}_{0,1} \cdots \boldsymbol{g}_{0,\alpha-1} \boldsymbol{g}_{1,0} \cdots \boldsymbol{g}_{1,\alpha-1} \cdots \boldsymbol{g}_{n-1,\alpha-1}),$$

and define $G_i = (\boldsymbol{g}_{i,0} \boldsymbol{g}_{i,1} \ldots \boldsymbol{g}_{i,\alpha-1})$, called a *node generator* for $i \in [n]$.

From $\boldsymbol{c} = G^T \boldsymbol{m}$, we know that G^T is the matrix that defines the linear transformation from message space $\mathcal{M} = \{\boldsymbol{m}\}$ into the code space $\mathcal{C}$. In order to recover the original message space from a subset of nodes, the corresponding submatrix of G^T

should be invertible. Consequently, we call the vector space spanned by the generator vectors the linear transformation (LT) space V, i.e., $V = \langle \boldsymbol{g}_{0,0}, \boldsymbol{g}_{0,1}, \ldots, \boldsymbol{g}_{n-1,\alpha-1} \rangle$ and $\dim(V) = M$, where $\langle \cdot \rangle$ is the span of vectors in it, and $\dim(V)$ is the dimension of V. The α generator vectors from node i span a subspace V_i of V, that is, $V_i = \langle \boldsymbol{g}_{i,0} \boldsymbol{g}_{i,1} \cdots \boldsymbol{g}_{i,\alpha-1} \rangle \subseteq V$. Then data reconstruction from k nodes $i_0, i_1, \ldots, i_k \in [n]$ is equivalent to $V \subseteq \sum_{j=0}^{k-1} V_{i_j}$, where $V_i + V_j = \{x + y : x \in V_i, y \in V_j\}$ is the smallest subspace that contains both V_i and V_j. Further, repair of one node i from any other r nodes $i_0, i_1, \ldots, i_{r-1}$ means $V_i \subseteq \sum_{j=0}^{r-1} V'_{i_j}$, where V'_{i_j} is a subspace of V_{i_j} spanned by the β vectors from node i_j.

Basically, the construction from vector space is how to choose the generator vectors such that the spanned subspaces satisfy data reconstruction and repair constraints. In the following sections, we will demonstrate three different constructions with specific data recovery advantages.

5.4 New MDS Codes with Low Complexity

As explained in Section 5.2.1, construction of MDS array codes for DSSs can be reduced to the scalar case as in traditional linear block codes, and we can use the same code for different storage units. Hence we will simply use $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ to denote the n encoded symbols stored across n nodes, instead of $\boldsymbol{c_j} = (c_{0,j}, c_{1,j}, \ldots, c_{n-1,j})$, for an arbitrary storage unit $j \in [\alpha]$. Correspondingly, a generator matrix is $G_{k \times n} = (\boldsymbol{g}_0, \boldsymbol{g}_1, \ldots, \boldsymbol{g}_{n-1})$, and we only have to consider generator vectors $\boldsymbol{g}_i$ for $i \in [n]$.

Since for traditional MDS codes, data repair is through decoding first and then

re-encode to get the coded symbols in the failed node, code construction is to ensure data reconstruction. For MDS codes, we have $\dim(V) = \dim(\sum_{j=0}^{k-1} V_{i_j}) = \sum_{j=0}^{k-1}\dim(V_{i_j})$, i.e., V_{i_j}'s have trivial intersections, where $i_0, i_1, \ldots, i_k \in [n]$ is a subset of $[n]$ of size k. Note that in this case $V_i = \langle \boldsymbol{g}_i \rangle$, hence we should choose $\boldsymbol{g}_i$'s to be linearly independent over the underlying field for any $i \in \{i_0, i_1, \ldots, i_k\} \subseteq [n]$.

For our new MDS codes, two constructions over GF(q^m) with $q > 2$ and over GF(2^m), respectively, are proposed. In both constructions, the encoded symbols are obtained by first treating the message vector as a linearized polynomial and then evaluating it at a set of evaluator points. This turns the linear dependency properties of generator vectors into corresponding requirements of evaluator points. To construct an (n, k) MDS code, the evaluator points are chosen so as to satisfy two conditions: (1) all the evaluator points form a k-dimensional subspace over GF(q), and (2) any k of them are linearly independent over GF(q). A Cauchy matrix is used to construct elements satisfying these two conditions.

Computational complexities benefit from the usage of linearized polynomials in two key aspects. First, the decoding procedure for data reconstruction can be performed by an interpolation algorithm for linearized polynomials, which has a quadratic complexity with respect to code parameters. Second, data repair has only a **linear** complexity, since only linear combinations of the encoded symbols in supporting nodes are required.

Note that GF(q^m) here is equivalent to GF(q) in Section 5.2, not an extension over it. We use GF(q^m) here for easier explanation of our ideas in this section.

The focus of this work is on computational complexity of MDS codes for distributed storage networks, and our current constructions are not optimal in terms of

repair bandwidth. However, our codes are suitable for applications where bandwidth is not a primary concern but a large amount of data repair is required.

5.4.1 Linearized Polynomials

A linearized polynomial $f(x)$ [31] over GF(q^m), where q is a prime power and m a positive integer, can be written as $f(x) = \sum_{i=0}^{l} a_i x^{[i]}$, where $a_i \in \text{GF}(q^m)$ and $[i] \stackrel{\text{def}}{=} q^i$. Suppose $\mathcal{K}$ is an extension field of GF(q^m), then $f(x)$ has the following properties:

$$f(\delta + \rho) = f(\delta) + f(\rho) \text{ for any } \delta, \rho \in \mathcal{K}, \text{ and} \tag{5.4}$$

$$f(c\delta) = cf(\delta) \text{ for any } c \in \text{ GF}(q) \text{ and } \delta \in \mathcal{K}. \tag{5.5}$$

In other words, $f(x)$ can be viewed as a linear mapping from $\delta \in \mathcal{K}$ to $f(\delta) \in \mathcal{K}$ with respect to GF(q) [31].

Encoding by evaluation of linearized polynomials is not unique in our work. Gabidulin codes [12] can be constructed from evaluation of linearized polynomials, as are a family of MDS codes based on Gabidulin codes in [68]. If we select the evaluator points such that they form an n-dimensional subspace over GF(q), an (n, k) Gabidulin code over GF(q^m)($n \leq m$) is obtained. However, similar to traditional MDS code, data repair for Gabidulin codes requires decoding of the message file.

A similar usage of linearized polynomial is also adopted for code constructions in distributed storage in [69], and data repair is also performed by linear dependencies incurred from properties of linearized polynomials. However, codes constructed in [69] work in a probabilistic way, i.e., neither data reconstruction nor data repair is guaranteed from an **arbitrary** subset of k nodes. That is, the construction in [69]

does not produce MDS codes.

5.4.2 Construction I over GF(q^m) with $q > 2$

In this section, we propose a construction of MDS codes over GF(q^m), where q is a prime power greater than two. Our (n, k) MDS code over GF(q^m) requires a base field size of $q \geq n$, and the code dimension satisfies $k \leq m$.

In our construction, each encoded symbol over GF(q^m) is evaluation of a linearized polynomial, whose coefficients come from the message vector, at one evaluator point. The evaluator points and the linearized polynomial are chosen to ensure linear and nonlinear constraints of the encoded symbols, so that data repair of our MDS codes can be simply performed by linear combinations of encoded symbols from the supporting nodes, while data reconstruction benefits from an efficient interpolation algorithm for linearized polynomials.

We start from an (n, k) linear block code defined by evaluation of linearized polynomials, and then propose specific constructions on the evaluator points that lead to an (n, k) MDS code over GF(q^m). Suppose the message vector is $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$, where $u_i \in \text{GF}(q^m)$ for $i = 0, 1, \ldots, k-1$. A linearized polynomial is defined to be $u(x) = \sum_{i=0}^{k-1} u_i x^{[i]}$. Then the corresponding codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ is given by $c_i = u(v_i)$, where $v_i \in \text{GF}(q^m)$ is an evaluator point for $i = 0, 1, \ldots, n-1$.

Observe that in this case a generator vector $\boldsymbol{g}_i = (v_i^{[0]}, v_i^{[1]}, \ldots, v_i^{[k-1]})^T$. It was proved in [31] that choosing $\boldsymbol{g}_i$'s to be linearly independent over GF(q^m) can be achieved by choosing v_i's to be linearly independent over GF(q). Hence the code is determined by the evaluator points, and we call the length-n vector $\boldsymbol{v} =$

$(v_0, v_1, \ldots, v_{n-1})$ an *evaluator vector*. The evaluator points are chosen such that the following two conditions are satisfied:

1. The n evaluator points form a k-dimensional subspace over GF(q).
2. Any k elements among $v_0, v_1, \ldots, v_{n-1}$ are linearly independent over GF(q).

Note that Condition 1) requires $k \leq m$. Condition 2) indicates that any k elements in $\boldsymbol{v}$ constitute a basis of the k-dimensional subspace formed by the n elements. In particular, we can fix the first k elements, and write the evaluator vector as follows.

$$\begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{pmatrix} = \left(\begin{array}{c} \boldsymbol{I}_k \\ \hline \boldsymbol{E}_c \end{array}\right) \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{k-1} \end{pmatrix}, \tag{5.6}$$

where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix and $\boldsymbol{E_c}$ is an $(n-k) \times k$ matrix over GF(q). Let us denote the coefficient matrix on the right hand side (RHS) by $\boldsymbol{E} = (\boldsymbol{I}_k | \boldsymbol{E_c}^T)^T$. Hence to satisfy the two conditions, we can first pick any k linearly independent elements in GF(q^m) to be $v_0, v_1, \ldots, v_{k-1}$, and then select $\boldsymbol{E_c}$ such that any k rows of $\boldsymbol{E}$ are linearly independent over GF(q). To achieve this, henceforth we assume $\boldsymbol{E_c}$ to be a Cauchy matrix over GF(q).

Lemma 10. *Any k rows in $\boldsymbol{E}$ are linearly independent.*

Proof. Let us denote the $k \times k$ matrix formed by any k rows of $\boldsymbol{E}$ to be $\boldsymbol{E}_k$, and assume that $\boldsymbol{E}_k$ has k_1 rows from $\boldsymbol{I}_k$ and $k_2 = k - k_1$ rows from $\boldsymbol{E_c}$. $\boldsymbol{E}_k$ is non-singular when $k_2 = 0$. When $k_1 = 0$, $\boldsymbol{E}_k$ is a square submatrix of $\boldsymbol{E_c}$, and hence

is nonsingular. Now for $k_1 \neq 0$ and $k_2 \neq 0$, we permute the columns of $\boldsymbol{E}_k$ to obtain $\boldsymbol{E}'_k = \left(\begin{array}{c|c} \boldsymbol{I}_{k_1} & \boldsymbol{0} \\ \hline \boldsymbol{A} & \boldsymbol{B} \end{array}\right)$, where $\boldsymbol{I}_{k_1}$ is the $k_1 \times k_1$ identity matrix, $\boldsymbol{0}$ is the $k_1 \times k_2$ all zero matrix, and $\boldsymbol{A}$ and $\boldsymbol{B}$ are submatrices of $\boldsymbol{E_c}$ of size $k_2 \times k_1$ and $k_2 \times k_2$, respectively. We have $\det(\boldsymbol{E}'_k) = \det(\boldsymbol{I}_{k_1}) \times \det(\boldsymbol{B}) = \det(\boldsymbol{B}) \neq 0$, where $\det(\cdot)$ is the determinant of a matrix. Since the permutation does not change the singularity of a matrix, $\boldsymbol{E}_k$ is nonsingular. □

Note that in order for $\boldsymbol{E_c}$ in Lemma 10 to be well defined, we need $q \geq n$, while the message symbols in this construction are over GF(q^m). This constraint on the base field size does not cause any significant difficulty in practice. In practice, messages are in bits, and hence we can choose q to be a power of 2 and divide the message bits into groups of size $\log_2(q^m)$ to be mapped into symbols over GF(q^m).

Once the evaluator vector $\boldsymbol{v}$ is fixed, the linear block code is determined. A generator matrix $\boldsymbol{G}$ for this linear block code can be found based on the following lemma.

Lemma 11. *For any $l \in \{0, 1, \ldots, n-1\}$, we have $v_l = \sum_{i=0}^{k-1} e_i v_i$ for some $e_i \in$ GF(q), then $v_l^{[j]} = \sum_{i=0}^{k-1} e_i v_i^{[j]}$ for any $j \in \{0, 1, \ldots, k-1\}$.*

Proof. In the first part, e_i's are elements of the coefficient matrix $\boldsymbol{E}$ in Eq. (5.6). Then we calculate $v_l^{[j]} = (\sum_{i=0}^{k-1} e_i v_i)^{[j]} = \sum_{i=0}^{k-1} e_i v_i^{[j]}$ since $e_i \in$ GF(q). □

According to the encoding procedure and Lemma 11, we have

$$\boldsymbol{G} = \begin{pmatrix} v_0 & v_1 & \dots & v_{k-1} \\ v_0^{[1]} & v_1^{[1]} & \dots & v_{k-1}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ v_0^{[k-1]} & v_1^{[k-1]} & \dots & v_{k-1}^{[k-1]} \end{pmatrix} \left(\begin{array}{c|c} \boldsymbol{I}_k & \boldsymbol{E_c}^T \end{array} \right). \tag{5.7}$$

It is easily verified that $\boldsymbol{H} = (-\boldsymbol{E_c}|\boldsymbol{I}_{n-k})$ is a parity check matrix for this code.

Lemma 12. *Any $n-k$ columns of the parity check matrix $\boldsymbol{H}$ are linearly independent over GF(q^m).*

By definition, the Cauchy matrix $\boldsymbol{E_c} = \{(x_i - y_j)^{-1}\}$ in Eq. (5.6), where $x_i, y_j \in$ GF(q) for $i = 0, 1, \dots, n-k-1$ and $j = 0, 1, \dots, k-1$. Since elements over GF(q) are also elements over GF(q^m), $\boldsymbol{E_c}$ is also a Cauchy matrix over GF(q^m). Following similar arguments to those in the proof of Lemma 10, we can show that any $n-k$ columns of $\boldsymbol{H}$ are linearly independent over GF(q^m).

Theorem 1. *The (n, k) linear block code constructed from Construction I is an (n, k) MDS code.*

Proof. Following Lemma 12, the minimum Hamming distance of this linear block code is $n-k+1$, and hence it is an (n, k) MDS code. □

Remark 2. *The MDS codes in [68] also pick n evaluator points so that any k of them are linearly independent, similar to Condition 2) of our approach. However, there is a key difference between the MDS codes in [68] and our codes: for our codes, any $k+1$ evaluator points need to be linearly **dependent**, while for codes in [68], there exist $k+1$ evaluator points that are linearly **independent**.*

Remark 3. *Cauchy matrices have been used to construct MDS codes such as generalized Reed-Solomon (GRS) codes [70] and Cauchy Reed-Solomon (CRS) codes [71]. GRS codes are obtained via evaluation of polynomials, not linearized polynomials as in this dissertation. Both GRS codes and CRS codes insert a Cauchy matrix explicitly in the generator matrix to ensure that the encoding leads to MDS codes. However, both codes are still traditional MDS codes in the sense that data repair still has to decode the original message first. In our constructions, aside from ensuring MDS codes, Cauchy matrices are also used to introduce linear dependency among evaluator points, hence data repair requires no decoding and is done by forming linear combinations of the encoded symbols from supporting nodes.*

5.4.3 Data Reconstruction and Data Repair

The two conditions on the evaluator vector bring linear dependency among the evaluator points, while keeping certain independency among them. After encoding, these properties are maintained due to the linear mapping properties of linearized polynomials. As a result, data reconstruction is guaranteed, while data repair can be conducted by linear combinations of the encoded symbols from supporting nodes.

Theorem 2. *(Data Reconstruction) The MDS code obtained from Construction I can reconstruct the original message vector from any k nodes.*

Proof. Suppose $\{i_0, i_1, \ldots, i_{k-1}\}$ is an arbitrary subset of size k of $\{0, 1, \ldots, n-1\}$. Then given encoded symbols $c_{i_0}, c_{i_1}, \ldots, c_{i_{k-1}}$ from node $i_0, i_1, \ldots, i_{k-1}$, we have the

following linear equations:

$$\begin{pmatrix} v_{i_0} & v_{i_0}^{[1]} & \dots & v_{i_0}^{[k-1]} \\ v_{i_1} & v_{i_1}^{[1]} & \dots & v_{i_1}^{[k-1]} \\ \vdots & \vdots & \ddots & \vdots \\ v_{i_{k-1}} & v_{i_{k-1}}^{[1]} & \dots & v_{i_{k-1}}^{[k-1]} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{k-1} \end{pmatrix} = \begin{pmatrix} c_{i_0} \\ c_{i_1} \\ \vdots \\ c_{i_{k-1}} \end{pmatrix}. \tag{5.8}$$

Condition 2) implies that $v_{i_0}, v_{i_1}, \dots, v_{i_{k-1}}$ are linearly independent, hence the coefficient matrix on the left hand side (LHS) is nonsingular [31]. As a result, Eq. (5.8) can be solved by multiplying both sides with the inverse of the coefficient matrix, i.e, the message vector $\boldsymbol{u}$ can be recovered from any k nodes. □

Theorem 3. *(Data Repair) The MDS code obtained from Construction I can repair the encoded symbol in any node by linear combining the encoded symbols from any other k nodes.*

Proof. Suppose we want to repair node i, where $i \in \{0, 1, \dots, n-1\}$, from nodes $l_0, l_1, \dots, l_{k-1}$, where $\{l_0, l_1, \dots, l_{k-1}\} = L_k$ is a subset of size k of $\{0, 1, \dots, n-1\} \backslash \{i\}$. Condition 2) satisfied by $\boldsymbol{v}$ implies that $v_i = \sum_{l \in L_k} e_l v_l$, where e_l's are elements in the coefficient matrix $\boldsymbol{E}$ in Eq. (5.6). Then $c_i = u(v_i) = u(\sum_{l \in L_k} e_l v_l) = \sum_{l \in L_k} e_l u(v_l) = \sum_{l \in L_k} e_l c_l$ from the linear mapping properties of linearized polynomials in Eq. (5.4) and Eq. (5.5). □

Hence data repair by our MDS codes can be performed by additions and multiplications over GF(q), instead of a matrix inversion over GF(q^m), which is required in traditional MDS codes from decoding the message vector in the first place.

5.4.4 Construction II over GF(2^m)

Since the MDS construction in Section 5.4.2 employs Cauchy matrix, q has to be greater than 2. Hence, these codes require multiplications and additions over GF(q) for data repair. In extension fields of GF(2), these operations reduce to simple AND and XOR operations, respectively. But if we set $q = 2$ in the construction in Section 5.4.2, only trivial repetition codes are obtained. In this section, we propose constructions to obtain nontrivial MDS codes over GF(2^m) with low repair complexity. Note that this construction fixes $k = n - 1$, leading to $(n, n-1)$ linear block codes.

We pick an evaluator vector $\boldsymbol{v} = \{v_0, v_1, \ldots, v_{n-1}\}$ over GF(2^m) such that it also satisfies the two conditions in Section 5.4.2. Note that Condition 2) reduces to $v_{n-1} = \sum_{i=0}^{k-1} v_i$, i.e., any evaluator point is the XOR of all other $n-1$ points. Then given a message vector, the formation of a linearized polynomial and the encoding follow a similar manner as that in Section 5.4.2.

Similarly, we can write a generator matrix for this linear block code as in Eq. (5.7). Note that $\boldsymbol{E_c}$ is $(\underbrace{1, 1, \ldots, 1}_{k})$ here, and a parity check matrix can be found to be $\boldsymbol{H} = (\underbrace{1, 1, \ldots, 1}_{n})$, in which any one column is linearly independent, leading to minimum Hamming distance of two. Hence this linear block code is also an $(n, n-1)$ MDS code over GF(2^m). Then following similar arguments in Section 5.4.2, we can show that data reconstruction and data repair can be performed correctly as in Theorems 2 and 3, respectively. Note that repairing one node now is simply XORing the encoded symbols from all other nodes.

5.4.5 Complexity Analysis

To solve Eq. (5.8), a general interpolation algorithm by linearized polynomials in [64] can be used, which requires $O(k^2)$ multiplications and $O(k^2)$ additions over GF(q^m). Hence, data reconstruction has a quadratic complexity with respect to the code dimension. Data repair for our MDS codes needs only k multiplications and $k-1$ additions over GF(q^m) to form linear combinations of encoded symbols from k supporting nodes as shown in Theorem 3, and hence has a linear complexity with respect to k.

We also make a comparison of the computational complexities between our MDS codes and a family of MSR codes. As pointed out in Section 5.2.3, MSR codes are also MDS array codes, and both codes store the same amount of message given the same k and n. Hence it is feasible to make a comparison between the two types of codes. Current exact-repair MSR codes have been proposed based on interference alignment [54, 58] or a product-matrix approach [59], all of which require matrix inversions for data repair. Hence without loss of generality, we compare the data repair complexities between our MDS codes and codes obtained from the product-matrix approach in [59].

Since our constructions are determined by n and k, we express the complexities in terms of these two parameters. Since the construction in [59] requires $d \geq 2k-2$, we fix $d = 2k-2$ to obtain $\alpha = k+1$, and have α expressed as $O(k)$. To make a fair comparison, we assume all operations are over characteristic two fields, and map all the multiplications and additions into equivalent operations over GF(2). For the product-matrix MSR code [59] constructed over a finite field of size at least n, message and encoded symbols are mapped into a binary vector of length

at least $\lceil \log_2(n) \rceil$. Similarly, for our codes constructed over GF(q^m), we assume the base field has a size of at least n, and map each element into a binary vector of length at least $m\lceil \log_2(n) \rceil$. For codes constructed over GF(2^m), the mapping is straightforward. Under these assumptions, the complexities of the three codes are listed in Table 5.2, where REC stands for data reconstruction, REP data repair, $\otimes$ binary multiplication, $\oplus$ binary addition, and all logarithms are base-2.

Table 5.2: Complexity of data reconstruction and data repair

Construction		[59]	Construction I	Construction II
Base Field		$2^{\lceil \log(nk) \rceil}$	$2^{\lceil \log(n) \rceil}$	2
REC	$\otimes$	$O(k^3\lceil \log(nk) \rceil^2)$	$O(k^5\lceil \log(n) \rceil^2)$	$O(k^5)$
	$\oplus$	$O(k^3\lceil \log(nk) \rceil^2)$	$O(k^5\lceil \log(n) \rceil^2)$	$O(k^5)$
REP	$\otimes$	$O(k^3\lceil \log(nk) \rceil^2)$	$O(k^2\lceil \log(n) \rceil^2)$	0
	$\oplus$	$O(k^3\lceil \log(nk) \rceil^2)$	$O(k^2\lceil \log(n) \rceil^2)$	$O(k^2)$

Table 5.2 shows that our MDS codes may have higher complexity for data reconstruction. But for data repair, the computational complexity of our MDS codes is much lower than the bandwidth-optimal distributed code in [59]. Hence our constructions are suitable for applications that need a small amount of data reconstruction but a large amount of data repair.

Remark 4. *As pointed out earlier, our new MDS codes are not optimal in terms of repair bandwidth. However, both data reconstruction and repair feature low computational complexity, from the usage of linearized polynomials. In particular, only linear combination operations will be carried out locally in new replacement nodes to recover failed ones, hence it suits applications where bandwidth is not a big concern but data repair is frequent, such as nodes leaving or joining the system dynamically. Furthermore, this construction provides some clue to construct optimal LRC codes*

in Section 5.5.

5.5 LRC Code Construction from Vector Space

In this section, we propose optimal LRC codes based on a two-layer encoding structure, viewed from a vector space perspective. We present a sufficient condition for the first layer MDS code to achieve the optimal minimum distance of the resulted LRC codes. We also prove that Gabidulin codes satisfy this condition, leading to the code proposed in [63]. Hence it is a special case of our construction. Though the condition is not necessary, which requires a bigger field than a current approach [60] for a special set of parameters, it is derived based on array codes analysis, hence is more efficient in terms of storage. Furthermore, efficient degraded reads [1] can be performed in DSS from our code structure.

It should be pointed out that the two-layer encoding (or concatenated encoding) is not a new technique. It has been widely used in the literature for different desired advantages in the DSS aside from [60][63]. For example, fractional repetition codes are proposed in [72] from concatenation of an outer MDS code and an inner repetition code for uncoded repair process. An outer MDS code and an inner fractional repetition code are employed in [73] to construct regenerating codes with local, exact and uncoded repair. Scalar linear codes such as Pyramid codes investigated in [53][74] and that proposed in [61] can also be viewed as examples of two-layer encoding.

First we discuss the achievability of the optimal distance in Eq. (5.2) under different parameter settings, and then present our code construction accordingly.

5.5.1 Achievability of Optimal Distance

As mentioned in Section 5.5, an iterative algorithm is used in [67] to prove the upper bound in Eq. (5.2). Based on the work in [67], we further claim the achievability of the optimal distance in Lemma 13.

Lemma 13. *The optimal distance in Eq. (5.2) is reachable if and only if* $(r+\delta-1)|n$, *or* $n(mod\ r+\delta-1)-(\delta-1) \geq \lceil \frac{M}{\alpha} \rceil (mod\ r) > 0$ *when* $(r+\delta-1) \nmid n$.

Proof. The optimal distance d in Eq. (5.2) can be obtained in two cases by using the algorithm in [67].

- The algorithm in [67] terminates in $\lceil \frac{M}{r\alpha} \rceil - 1$ steps, with each step adding exactly $r+\delta-1$ nodes with entropy $r\alpha$. Hence local repair groups should be non-overlapping in the first place. Further, the algorithm should reach the same result regardless of which $\lceil \frac{M}{r\alpha} \rceil - 1$ local repair groups are selected. Hence $(r+\delta-1)|n$. Simply speaking, the optimal distance is achievable only if nodes in the DSS can be divided into non-overlapping local repair groups of the same size. Conversely, if we can divide the nodes into non-overlapping groups of size $r+\delta-1$, any (r,δ) LRC codes can be used to accommodate entropy of $r\alpha$ within each group, which satisfies the termination of the algorithm.

- The algorithm in [67] terminates in $\lceil \frac{M}{r\alpha} \rceil$ steps, where $r+\delta-1$ nodes with entropy $r\alpha$ are added in each of the first $\lceil \frac{M}{r\alpha} \rceil - 1$ steps, and a other nodes with entropy $(a-\delta+1)\alpha$ are added in the last step, where a is a positive integer. As in the first case, non-overlapping local repair groups are also required. The last portion of entropy indicates that $\lceil \frac{M}{\alpha} \rceil (\text{mod } r) > 0$, and $n(\text{mod } r+\delta-1)-(\delta-1) \geq \lceil \frac{M}{\alpha} \rceil (\text{mod } r)$ for the last group added to provide

this portion of entropy. In other words, when $(r+1) \nmid n$, the optimal d is reachable only if $n(\text{mod } r+\delta-1)-(\delta-1) \geq \lceil \frac{M}{\alpha} \rceil (\text{mod } r) > 0$. The converse can be proved similarly as in the first case.

□

5.5.2 Code Structure

We use the same two-layer encoding structure as that in [63] to construct optimal LRCs for the two cases in Lemma 13. A scalar MDS code is used in the first layer, whose encoded symbols are partitioned into sets of size $r\alpha$, to be stored in non-overlapping local repair groups. Then a second layer encoding is performed within each local repair group by an $(r+\delta-1, r)$ MDS array code to ensure (r, δ) locality. We will show that as long as the first layer MDS code satisfies a special property, the overall code reaches the desired repair locality as well as the optimal minimum distance.

Suppose a message file of M symbols over GF(q) is to be encoded and stored in a DSS with n nodes, each of which has a storage capacity of α symbols. We construct $(n, \alpha, M, d; r, \delta)$ LRC codes such that the system has (r, δ) locality and minimum distance $d = n - \left\lceil \frac{M}{\alpha} \right\rceil + 1 - \left(\left\lceil \frac{M}{r\alpha} \right\rceil - 1 \right)(\delta - 1)$, or any $k^* = \left\lceil \frac{M}{\alpha} \right\rceil + \left(\left\lceil \frac{M}{r\alpha} \right\rceil - 1 \right)(\delta - 1)$ nodes suffice for data reconstruction. Note that if $M \leq r\alpha$, each node will be repaired locally by r other nodes, while the traditional repair through data reconstruction needs $\lceil \frac{M}{\alpha} \rceil \leq r$ nodes, which is not the purpose of LRC codes. Hence we assume $M > r\alpha$ in the rest of this chapter. We introduce $M^* = \lceil \frac{M}{\alpha} \rceil \alpha$ for simplicity. It can be easily shown that $\lceil \frac{M}{\alpha} \rceil = \lceil \frac{M^*}{\alpha} \rceil$ and $\lceil \frac{M}{r\alpha} \rceil = \lceil \frac{M^*}{r\alpha} \rceil$, hence $k^* = \lceil \frac{M^*}{\alpha} \rceil + \left(\lceil \frac{M^*}{r\alpha} \rceil - 1 \right)(\delta - 1)$.

As discussed in Section 5.5.1, we assume non-overlapping local repair groups to obtain optimal distance. Suppose the n nodes of storage capacity α are labeled as $0, 1, \ldots, n-1$, and divided into Δ non-overlapping groups, denoted by $\mathcal{G}_j$, where Δ is a positive integer greater than 1, and $j \in [\Delta]$. Depending on whether $r+\delta-1$ divides n or not, we construct the optimal LRC code for the two cases accordingly.

1) $n = \Delta(r+\delta-1)$. The optimal distance achievability analysis in Section 5.5.1 suggests that $\delta-1$ nodes in each local repair group will solely be used for local repair, while the other r nodes ensures reliability of the overall code. Hence we need to embed M information symbols in $\Delta r\alpha$ encoded symbols. First we pad M^*-M zeros into the message, and use a $(\Delta r\alpha, M^*)$ MDS code $\mathcal{C}^{(1)}$ to obtain $\Delta r\alpha$ encoded symbols, and store them into the first r nodes of each repair group, shown as the blank areas in Fig. 5.2, with $r_{\Delta-1} = r$. Based on Fig. 5.2, we may refer to a node as a *column*, and another dimension a *row* of the DSS. Next, within each group $\mathcal{G}_j$, an $(r+\delta-1, r)$ systematic MDS array code $\mathcal{C}^{(2)}$ over GF(q) is used to encode the $r\alpha$ encoded symbols of $\mathcal{C}^{(1)}$ and store the parity checks in the $\delta-1$ shaded columns.

2) $n = (\Delta-1)(r+\delta-1) + r_{\Delta-1} + \delta - 1$, where $r_{\Delta-1} \geq \lceil \frac{M}{\alpha} \rceil (\text{mod } r) > 0$. Similarly, we pad zeros to the message file if necessary, and encode M^* symbols into $(\Delta-1)r\alpha + r_{\Delta-1}\alpha$ symbols using a $((\Delta-1)r\alpha + r_{\Delta-1}\alpha, M^*)$ MDS code $\mathcal{C}^{(1)}$, and store them in the blank columns of Fig. 5.2. Next, each of the first $\Delta-1$ groups employs an $(r+\delta-1, r)$ systematic MDS array code $\mathcal{C}^{(2)}$ over GF(q), and store the parity checks in its shaded columns respectively, as in the first case. For $G_{\Delta-1}$, an $(r_{\Delta-1}+\delta-1, r_{\Delta-1})$ systematic MDS array code $\mathcal{C}^{(3)}$ over GF(q) is used to obtain coded symbols in the last $r_{\Delta-1}+\delta-1$ nodes.

In both cases, we obtain an (n, α, M^*) vector code $\mathcal{C}$. Let $\boldsymbol{m} = (m_0 m_1 \ldots m_{M^*-1})^T$

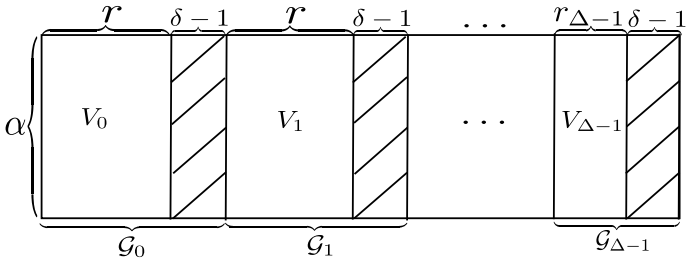

Figure 5.2: Two-layer encoding structure.

be the message vector after padding of zeros (if necessary), and $\boldsymbol{g}_{i,j}$ the generator vectors for $i \in [n]$ and $j \in [\alpha]$ in a generator matrix G. Note that $\mathcal{C}^{(2)}$ (and $\mathcal{C}^{(3)}$ in case 2) is an $(r+\delta-1, r)$ systematic MDS array code over GF(q), which can be obtained by employing a systematic $(r+\delta-1, r)$ MDS scalar code over GF(q) in each row of the local repair group for simplicity. In this case, we get $G^{(1)} = (\boldsymbol{g}_{i',j})$ where $j \in [\alpha]$ and $i' \in [n] \backslash \{t(r+\delta-1)+r_0, t(r+\delta-1)+r_0+1, \ldots, t(r+\delta-1)+r_0+\delta-2 : r_0 \in \{r, r_{\Delta-1}\}, t \in [\Delta-1]\}$. Then $G^{(1)}$ is a generator matrix of $\mathcal{C}^{(1)}$. Let $G^{(2)} = (\boldsymbol{I}_0, \boldsymbol{I}_1, \ldots, \boldsymbol{I}_{r_0-1} | \boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_{\delta-2})$ be a generator matrix of $\mathcal{C}^{(2)}$ (or $\mathcal{C}^{(3)}$), where $\boldsymbol{I}_j, j \in [r_0]$ is the jth column of the $r_0 \times r_0$ identity matrix I_{r_0}. Then

$$\boldsymbol{g}_{i+r_0+\epsilon,\ell} = [\boldsymbol{g}_{i,\ell} \boldsymbol{g}_{i+1,\ell} \cdots \boldsymbol{g}_{i+r_0-1,\ell}] \boldsymbol{\eta}_\epsilon, \tag{5.9}$$

where $i = j(r+\delta-1)$ for $j \in [\Delta-1], \epsilon \in [\delta-1], \ell \in [\alpha]$, and $r_0 \in \{r, r_{\Delta-1}\}$. Eq. (5.9) establishes the linear dependency of generator vectors of $\mathcal{C}^{(2)}$ and $\mathcal{C}^{(3)}$ on that of $\mathcal{C}^{(1)}$. A different set of equations will be obtained if we use arbitrary $(r+\delta-1, r)$ systematic MDS array code over GF(q) for $\mathcal{C}^{(2)}$ ($\mathcal{C}^{(3)}$), but the dependency between generator vectors of $\mathcal{C}^{(2)}$ and $\mathcal{C}^{(3)}$ on that of $\mathcal{C}^{(1)}$ will not be changed, which is the basis of our proof of the data reconstruction.

5.5.3 Local Repair and Data Reconstruction

As described in the previous section, the local repair property is solely determined by the second layer encoding, which guarantees the desired locality for the DSS.

Theorem 4. *The LRC code constructed in Section 5.5.2 has a repair locality of r.*

Proof. Each node participates in a $(r+\delta-1, r)$ or $(r_{\Delta-1}+\delta-1, r_{\Delta-1})$ MDS array code, which can be repaired by at most r other nodes. Hence the LRC code has a repair locality of r. □

The data reconstruction, on the other hand, depends on the erasure correction capability of the overall code $\mathcal{C}$. Note that generator vectors from blank columns of Fig. 5.2 are exactly the same ones from $\mathcal{C}^{(1)}$, hence any subset of them have full rank as long as the set size is no greater than M^*. In particular, those from the same local repair group $\mathcal{G}_j$ span a vector space V_j of dimension $r_0\alpha$ over GF(q) given that $r\alpha < M \leq M^*$, where $j \in [\Delta], r_0 \in \{r, r_{\Delta-1}\}$.

Meanwhile, generator vectors in the shaded columns are linear combinations of that from the blank columns, as shown in Eq. (5.9). Hence all the $r_0+\delta-1$ nodes in $\mathcal{G}_j$ span the same vector space V_j as obtained from the first r_0 nodes for $j \in [\Delta], r_0 \in \{r, r_{\Delta-1}\}$, as shown in Fig. 5.2. Now we are ready to prove the data reconstruction of $\mathcal{C}$.

The data reconstruction problem requires the original message be recovered based on any k^* nodes, or equivalently, $k^*\alpha$ generator vectors from any k^* node generators have rank M^*. We will show that this condition is satisfied by employing some special MDS codes as $\mathcal{C}^{(1)}$ that have the following property.

Subspaces of V_j's have trivial intersections if the summation of their dimensions is no greater than M^*.

In the next section, we will prove that Gabidulin codes satisfy Property 5.5.3. Here, we consider the data reconstruction of the code derived from employing MDS codes with Property 5.5.3.

Theorem 5. *Data reconstruction can be performed by any* $k^* = \lceil \frac{M}{\alpha} \rceil + \left(\lceil \frac{M}{r\alpha} \rceil - 1 \right) (\delta - 1)$ *columns of code* $\mathcal{C}$.

Proof. The theorem can be proved by showing that the $k^*\alpha$ generator vectors corresponding to any k^* columns of $\mathcal{C}$ span a vector space of dimension M^* over GF(q). We prove this for the two cases of construction in Section 5.5.2 respectively.

1) $n = \Delta(r + \delta - 1)$. From previous analysis, subspaces with smaller dimensions can be obtained by picking nodes from the same local repair group as many as possible. Since $\alpha | M^*$, we can write $M^* = \lambda r\alpha + r_1\alpha$, where $0 \leq r_1 \leq r - 1$, and have two different k^*'s:

- $k^* = \lambda r + (\lambda - 1)(\delta - 1) = (\lambda - 1)(r + \delta - 1) + r$ if $r_1 = 0$ or $r\alpha | M^*$. In this case, we pick the k^* nodes by first choosing $(\lambda - 1)(r + \delta - 1)$ nodes from $\lambda - 1$ different local repair groups, and then selecting another r nodes randomly from the remaining repair groups. Based on Property 5.5.3, the last r nodes span a subspace U_0 of dimension $r\alpha$, while the first $(\lambda - 1)(r + \delta - 1)$ nodes span a subspace U_1 of dimension $(\lambda - 1)r\alpha$, given that $r\alpha < M^*$ and $(\lambda - 1)r\alpha < M^*$. Using Property 5.5.3 again, U_0 and U_1 have only trivial intersection since the summation of their dimensions is exactly M^*. Hence any k^* nodes will span a subspace with dimension at least M^*.

- $k^* = \lambda(r+\delta-1)+r_1$ if $r_1 > 0$ or $r\alpha \nmid M^*$. Similarly, we compose the k^* worst-case nodes with $\lambda(r+\delta-1)$ elements from λ different local groups, and r_1 others randomly selected from the remaining groups. Following a similar argument as the first case, those nodes span a vector space of dimension $\lambda r\alpha + r_1\alpha = M^*$.

In either setting, any k^* nodes will span a subspace with dimension at least M^*. Conversely, the largest possible dimension spanned by any k^* nodes is also M^*. Hence generator vectors from any $k^* = \lceil \frac{M^*}{\alpha} \rceil + \left(\lceil \frac{M^*}{r\alpha} \rceil - 1 \right)(\delta-1) = \lceil \frac{M}{\alpha} \rceil + \left(\lceil \frac{M}{r\alpha} \rceil - 1 \right)(\delta-1)$ nodes span an M^*-dimensional vector space over GF(q), and the original message file can be reconstructed.

2) $n = (\Delta-1)(r+\delta-1)+r_{\Delta-1}+\delta-1$, where $r_{\Delta-1} \geq \lceil \frac{M}{\alpha} \rceil (\text{mod } r) > 0$. In this case, $M^* = \lambda r\alpha + r_1\alpha$, where $0 < r_1 \leq r_{\Delta-1} \leq r-1$, and $k^* = \lambda(r+\delta-1)+r_1$. Similarly, we pick $\lambda(r+\delta-1)$ elements from λ different local groups, and r_1 randomly from other remaining groups. Note that every group suffice to provide a $r_1\alpha$ dimensional subspace given that $r_{\Delta-1} \geq r_1$. Following a similar argument as the first case, those nodes span a vector space of dimension $\lambda r\alpha + r_1\alpha = M^*$, and data construction can be performed correctly. □

5.5.4 Relation to Other Works

The general construction in Section 5.5.2 requires $\mathcal{C}^{(1)}$ to be an MDS code that satisfies Property 5.5.3, and we will show that Gabidulin codes [12] have this desirable property, leading to an explicit construction of our approach.

Claim 1. *Gabidulin codes can satisfy Property 5.5.3.*

Proof. Consider an $(\Delta r\alpha, M^*)$ Gabidulin code over GF(q^m) with $m \geq \Delta r\alpha$, determined by generator vectors $\boldsymbol{g}_0, \boldsymbol{g}_1, \ldots, \boldsymbol{g}_{\Delta r\alpha-1}$, and corresponding evaluator points $v_0, v_1, \ldots, v_{\Delta r\alpha-1}$. Note that the $\Delta r\alpha$ evaluator points are linearly independent over GF(q), and will span a vector space W of dimension $\Delta r\alpha$ over GF(q). Based on our code construction, those evaluator points will be divided into Δ non-overlapping groups, and the $r\alpha$ evaluator points of group j will span a subspace $W_j \subseteq W$ of dimension $r\alpha$ over GF(q). Note that W_j's have trivial intersections, as otherwise we will have $\dim(W) = \dim(\sum_{j=0}^{\Delta-1} W_j) < \dim(\sum_{j=0}^{\Delta-1} \oplus W_j) = \Delta r\alpha$. In particular, subspaces of W_j's of dimension d_j have trivial intersections if the summation of their dimensions is no greater than M^*, as $M^* \leq \Delta r\alpha$, where $j \in [\Delta'] \subseteq [\Delta]$. Equivalently, all the basis vectors $v'_{j,t}$ for $j \in [\Delta'], t \in [d_j]$ from those subspaces are linearly independent over GF(q). Now we use $v'_{j,t}$ as an evaluation point, and construct a generator vector $\boldsymbol{\eta}_{j,t} = ((v'_{j,t})^{[0]}, (v'_{j,t})^{[1]}, \ldots, (v'_{j,t})^{[M^*-1]})$, and those $\sum_{j\in[\Delta']} d_j$ vectors are linearly independent over GF(q^m), i.e., they span a $(\sum_{j\in[\Delta']} d_j)$-dimensional subspace of V. Naturally, subspace V'_j spanned by $\boldsymbol{\eta}_{j,t}$ with a fixed j must have trivial intersections with those from groups $j' \in [\Delta']$ and $j' \neq j$. Hence for our code construction, we choose $\boldsymbol{\eta}_{j,t} = \sum_{\ell=0}^{r\alpha-1} a_{t,j,\ell}\boldsymbol{g}_{j,\ell}$, where $a_{t,j,\ell} \in$ GF(q) for $j \in [\Delta']$ and $t \in [d_j]$. Then their evaluation points are $v'_{j,t} = \sum_{\ell=0}^{r\alpha-1} a_{t,j,\ell}v_{j,\ell}$ from Eq. (5.4) and (5.4), and we have shown that they will guarantee that $\boldsymbol{\eta}_{j,t}$'s span subspaces of V_j's with trivial intersections. □

When using Gabidulin codes as in the proof of Claim 1, we obtain codes proposed in [63]. Hence the codes in [63] can be viewed as a special case of our construction.

It should be pointed out that Property 5.5.3 is a sufficient but not a necessary condition for optimal LRC code. By requiring certain subset of V_j subspaces have

trivial intersections, it is guaranteed that any set of their basis vectors are linearly independent over some field, including the $k^*\alpha$ generator vectors from any k^* nodes. Hence as long as we have generator matrix of $\mathcal{C}^{(2)}$ over the coefficient field, the original data can be recovered correctly. Since these $k^*\alpha$ generator vectors only compose one particular basis set, Property 5.5.3 requires more than necessary, which is reflected by the large field size.

Also note that although we use the same two-layer encoding structure as that in [63], and reach to the same code when adopting Gabidulin codes, we tackle the problem from a totally different perspective. In [63], Gabidulin codes are considered so that $d-1$ node erasures can be turned into $d-1$ rank erasures, which are then proved to be correctable by the Gabudulin code. In our approach, we treat $d-1$ node erasures as $(d-1)\alpha$ erasures of coded symbols, and consider the subspace spanned by the corresponding generator vectors.

Our vector space approach also has a flexible structure. The proof of Theorem 5 is based on subspaces spanned by generator vectors of $\mathcal{C}^{(1)}$. Note that the same V_j will be obtained if we rearrange coded symbols (generator vectors) of $\mathcal{C}^{(1)}$ within each local repair group in Fig. 5.2. Hence there's actually no strict rule of how the encoded symbols should be placed in the storage units within the same local group, as far as data reconstruction is concerned. Of course, repair locality must be guaranteed by the placing of encoded symbols. Though it doesn't seem to provide meaning benefits for now, it will facilitate our comparison with another piece of work.

For some particular set of parameters, optimal LRC codes have been proposed over smaller fields than our vector space approach, such as those with $\delta = 2, (r+1)|n$

and $r+1=\alpha$ in [60]. However, we show that our construction, though requires a bigger field, stores more information given the same set of parameters, and also induces efficient degraded reads in practice.

As pointed out earlier, we may store the encoded symbols in each local group arbitrarily. In particular, for $r+1=\alpha$, we can store the $r\alpha$ encoded symbols of $\mathcal{C}^{(1)}$ in the first r rows, and the α parity checks from $\mathcal{C}^{(2)}$ in the last row, instead of the last column of $\mathcal{G}_j$.

Note that encoded symbols of the same codeword of $\mathcal{C}^{(2)}$ should come from different columns (nodes) in order to obtain the desired repair locality, which can be implemented by simple permutations. Let $\boldsymbol{\pi}=(r,r-1,\ldots,0)$, and $\boldsymbol{\pi}_\ell$ the ℓ-th right circulant of $\boldsymbol{\pi}$, that is, $\boldsymbol{\pi}_\ell(t)=\boldsymbol{\pi}((t+\ell) \bmod (r+1))$ for $\ell,t\in[r+1]$. Then the ℓ-th codeword of $\mathcal{C}^{(2)}$ is obtained by $\sum_{t=0}^{r} c_{i,t}=0$, where $i=\boldsymbol{\pi}_\ell(t)$. For example, $\boldsymbol{\pi}_r=(r-1,r-2,\ldots,0,r)$, and $c_{r-1,0}+c_{r-2,1}+\cdots+c_{r,r}=0$, from which $c_{r,r}$ can be calculated and stored into row r of node r.

Example 5. *We construct an example with $n=6,\alpha=3,M=8,r=2,\delta=2$, leading to $M^*=9,k^*=4$ and a designed distance $d=3$. A $(12,9)$ Gabidulin code over GF(q^m) with $m\geq 12$ is adopted as $\mathcal{C}^{(1)}$, and $\mathcal{C}^{(2)}$ a $(3,2)$ single parity check code (an MDS code). A length-8 message is first padded with a 0. Suppose $\boldsymbol{c}=(c_{i,j})^T\in\mathcal{C}^{(1)}$ is obtained in the first layer encoding, where $i\in\{0,1,3,4\},j\in\{0,1,2\}$. The corresponding codeword of $\mathcal{C}$ is shown in Table 5.3, where n_i is storage node i with $i\in[6]$, and $c_{i,j}=c_{i-1,j}+c_{i-2,j}$ for $i\in\{2,5\},j\in\{0,1,2\}$. It can be verified that any node has a repair locality of 2, and any $k^*=4$ nodes suffice to reconstruct $\boldsymbol{m}$. Hence the maximums minimum distance $d=3$ is reached, and the code in Table 5.3 is a $(6,3,8,3;2,3)$ optimal LRC code.*

Table 5.3: A $(6, 3, 8, 3; 2, 3)$ optimal LRC code

n_0	n_1	n_2	n_3	n_4	n_5
$c_{0,0}$	$c_{1,0}$	$c_{0,0} + c_{1,0}$	$c_{3,0}$	$c_{4,0}$	$c_{3,0} + c_{4,0}$
$c_{0,1}$	$c_{1,1}$	$c_{0,1} + c_{1,1}$	$c_{3,1}$	$c_{4,1}$	$c_{3,1} + c_{4,1}$
$c_{0,2}$	$c_{1,2}$	$c_{0,2} + c_{1,2}$	$c_{3,2}$	$c_{4,2}$	$c_{3,2} + c_{4,2}$

The code in Table 5.3 has the original structure as in Fig. 5.2. Table 5.4 gives another $(6, 3, 8, 3; 2, 3)$ optimal LRC code with exactly the same parameters but a different structure, as that in [60]. Note that given the same input message, codeword $\boldsymbol{b} = (b_0, b_1, \ldots, b_{11})^T$ in Table 5.4 is the same as $\boldsymbol{c} = (c_{i,j})^T$ for $i \in \{0, 1, 3, 4\}, j \in \{0, 1, 2\}$ in Table 5.3. However, parity check symbols from $\mathcal{C}^{(2)}$ are formed according to the permutation approach above to ensure a repair locality of 2.

Table 5.4: Another $(6, 3, 8, 3; 2, 3)$ optimal LRC code

n_0	n_1	n_2	n_3	n_4	n_5
b_0	b_2	b_4	b_6	b_8	b_{10}
b_1	b_3	b_5	b_7	b_9	b_{11}
$b_3 + b_4$	$b_0 + b_5$	$b_1 + b_2$	b_9+$_{10}$	$b_6 + b_{11}$	$b_7 + b_8$

Though both of our code and that in [60] have a message size of $M = 8$, our code construction can actually accommodate $M^* = 9$ message symbols given the same DSS and the same data reconstruction and repair requirement. In the worst case, up to $\alpha - 1$ storage units will be wasted by the structure in [60]. Hence our approach is more efficient, as it is constructed based on array codes, instead of dividing the DSS into units to suit for scalar codes, which is performed in [60].

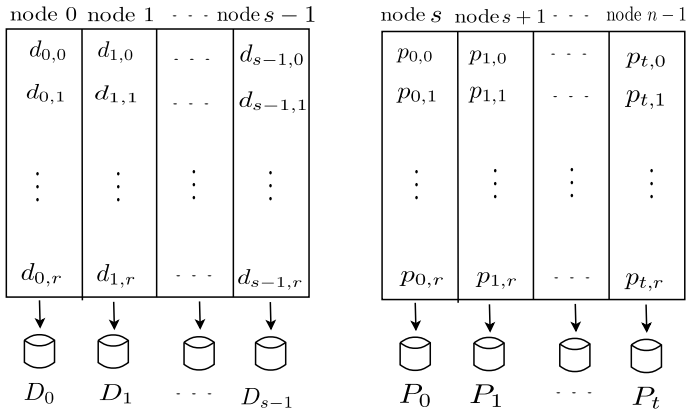

Figure 5.3: A Systematic Code for Degraded Reads (reproduced from [1])

5.5.5 Degraded Reads

Both schemes in Table 5.3 and 5.4 achieve the desired data repair and reconstruction parameters. However, compared to [60], our codes in Section 5.5.2 feature other merits such as efficient degraded reads [1].

As pointed out in [1] and the references therein, disk failures are dominated by temporary unavailability due to network partitions, software updates and so on. In the period between failure and recovery, reads are *degraded* because data from failed nodes must be recovered to complete the read process. For single disk failures, a *penalty* is defined to be the number of symbols required to perform the read minus the number of symbols desired to be read.

If random reads do not cause extra cost (e.g., delay), both codes from our construction and [60] induce a penalty of at most $r-1$ in degraded reads. Given the same repair locality of r, reading one symbol from a failed node can be performed by reading at most r other symbols in other nodes. In practice, however, reading from random positions of a disk could be time consuming, and successive reads are

preferred. In this case, degraded reads performed by our construction have less penalty than that in [60].

Suppose a systematic code $\mathcal{C}$ is used for simplicity, and the message symbols are stored in data disks $D_0, D_1, \ldots, D_{s-1}$, and parity symbols are stored in parity disks $P_0, P_1, \ldots, P_t$, respectively, as shown in Fig. 5.3 (reproduced from [1, Fig. 1]), where $s = \lceil \frac{M}{\alpha} \rceil$ and $t = n - s - 1$. Without loss of generality, let us assume the first Δ parity disks stores the single parity check symbols of $\mathcal{C}^{(2)}$ for local group $0, 1, \ldots, \Delta - 1$, respectively, and the rest stores that from $\mathcal{C}^{(1)}$.

As in [1], we assume contiguous data symbols are stored in successive disks to take better advantage of parallel I/O, i.e., successive reads are performed from the starting point to the end in a row by row manner. For our construction, at most r extra symbols in the same row are required to be read if one node fails, as local repair constraints are conducted row wisely. Hence a penalty of at most $r - 1$ is necessary. On the other hand, the structure in [60] stores the $\alpha = r + 1$ symbols participating in the same parity check equation of $\mathcal{C}^{(2)}$ in different rows. To be specific, node t stores a symbol of the ℓ-th codeword in row $\boldsymbol{\pi}_\ell(t)$, hence up to $\boldsymbol{\pi}_\ell(t) + 1$ symbols are to be read from node t to repair some other symbol participating in the same codeword. Given that $\boldsymbol{\pi}_\ell(t) \in [r + 1]$, in the worst case, reading of $r + 1, r, \ldots, 2$ symbols (rows) from r nodes respectively is necessary to repair one symbol in a failed node. Hence a penalty of $\frac{(r+1)(r+2)}{2} - 2$ is resulted, in the oder of $O(r^2)$.

For example, suppose node 0 fails in Table 5.4, and b_0 is to be read. If using successive reads, we have to read 3 symbols in node n_1 till $b_0 + b_5$ is reached and 2 symbols in node n_2 till b_5 is obtained to repair b_0. Therefore a total of 5 reads and a penalty of 4 is necessary, reaching the upper-bound of $\frac{(r+1)(r+2)}{2} - 2$ above. On the

other hand, if $c_{0,0}$ is to be read in Table 5.3 while node 0 fails, we only need to read $c_{1,0}$ from node 1 and $c_{0,0} + c_{1,0}$ from node 2 to recover $c_{0,0}$, and the penalty is 1.

5.5.6 Code Rate

We define the code rate R of $\mathcal{C}$ to be the ratio of the number of original message symbols over the total number of storage units required to store the encoded symbols, that is

$$\begin{aligned} R &\stackrel{\text{def}}{=} \frac{M}{n\alpha} \leq \frac{M^*}{n\alpha} = \frac{M^*}{((\Delta-1)(r+\delta-1)+r_{\Delta-1}+\delta-1)\alpha} \\ &= R^{(1)} \frac{(\Delta-1)r+r_{\Delta-1}}{(\Delta-1)(r+\delta-1)+r_{\Delta-1}+\delta-1}, \end{aligned}$$

where $R^{(1)} = \frac{M^*}{(\Delta-1)r\alpha+r_{\Delta-1}\alpha}$ is the code rate of code $\mathcal{C}^{(1)}$. Hence the code rate of $\mathcal{C}$ is bounded by that of $\mathcal{C}^{(1)}$, and the factor $\frac{(\Delta-1)r+r_{\Delta-1}}{(\Delta-1)(r+\delta-1)+r_{\Delta-1}+\delta-1}$ reflects the cost of extra storage to obtain the (r, δ) repair locality. Note when we set $\delta = 2$ and $(r+1)|n$, we have $R = R^{(1)}\frac{r}{r+1}$, the same as that presented in [60].

5.6 MSR Code Construction from Vector Space

In this section, we consider deterministic linear constructions of distributed storage coding from a perspective of linear vector space. We present explicit MSR codes with small field size when the code parameters are small. Though our construction process focuses only on data reconstruction, the subspaces under the set of parameters considered display some very nice intersection properties, which also lead to the desired data repair results.

No explicit constructions for lager code parameters are found, as certain key properties of subspaces soon become too complicated to be tracked when their dimension increases. However, we want to point out that our current construction is quite "passive", in the sense that no constraints are imposed during the construction process to ensure data repair. If both data reconstruction and data repair requirements are taken into account, our subspace approach may produce MSR codes with general parameters.

5.6.1 MSR Codes from Vector Space Approach

We now use the same vector space approach and notations presented in Section 5.5 to interpret MSR codes. For MSR codes, we have $M = k\alpha$, i.e., $\dim(V) = M$ and $\dim(V_i) = \alpha$. Hence $V = \sum_{j=0}^{k-1} V_{i_j} = \sum_{j=0}^{k-1} \oplus V_{i_j}$, where $\oplus$ is the direct sum of two subspaces. This means the $k\alpha$ generator vectors from any k nodes should be linearly independent, and span the LT space V. On the other hand, data repair imposes linear dependencies among any $r+1$ node generators.

Table 5.5: A $(5, 2, 4; 2, 3)$ MSR Code

node i	0	1	2	3	4
x_i	0 0 0 1	0 1 0 0	1 0 1 0	0 1 1 0	1 1 1 0
y_i	0 0 1 0	1 0 0 0	0 1 0 1	1 0 1 1	0 1 1 1
z_i	0 0 1 1	1 1 0 0	1 1 1 1	1 1 0 1	1 0 0 1

We illustrate our idea first with a simple example. Table 5.5 shows a $(n = 5, \alpha = 2, M = 4; k = 2, r = 3)$ MSR code, where $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are the generator vectors for node i over GF(2). Here V is the 4-dimensional vector space spanned by all the

length $M = 4$ vectors. A generator matrix G for this code is

$$G^T = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Here V is the 4-dimensional vector space spanned by all the length $M = 4$ vectors. It can be verified that $V_i = \langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \subseteq V$ has dimension 2, where $i \in [5]$. Further, the 4 generator vectors from any two V_i and V_j are linearly independent for $i, j \in [5]$ and $i \neq j$, and will span the entire vector space V. Hence the original message can be recovered by multiplying the coded symbol with the inverse of the corresponding matrix. Meanwhile, some three vectors from any $r = 3$ nodes, one from each, can repair the rest nodes by simple linear combinations (XOR's in this example). For example, to repair node 3 from node 0, 1, and 2, we have $\boldsymbol{x}_3 = (0110) = (0010) + (0100) = \boldsymbol{y}_0 + \boldsymbol{x}_1$ and $\boldsymbol{y}_3 = (1011) = (0100) + (1111) = \boldsymbol{x}_1 + (\boldsymbol{x}_2 + \boldsymbol{y}_2)$. Hence three symbols corresponding to $\boldsymbol{y}_0, \boldsymbol{x}_1, \boldsymbol{x}_2 + \boldsymbol{y}_2$ from node 0, 1, 2, respectively, repair the two symbols $\boldsymbol{x}_3, \boldsymbol{y}_3$ in node 3. Note that $\boldsymbol{x}_2 + \boldsymbol{y}_2$ is used in node 2, meaning that a helping node can do linear combinations locally before sending out the symbol to repair the failed node, which is the essence of reducing bandwidth in regenerating codes, compared to erasure correction of traditional codes.

Note there is an extra row of vectors $\boldsymbol{z}_i = \boldsymbol{x}_i + \boldsymbol{y}_i$ in Table 5.5, displayed for better illustration of our code construction, though they are not to be stored in node i. It can be seen that $V_i = \{\mathbf{0}, \boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i\}$, where $\mathbf{0}$ is the all-zero vector, and V_i and

V_j's are mutually exclusive, i.e., with trivial intersections, for $i, j \in [5]$ and $i \neq j$. As a result, data reconstruction property is guaranteed.

5.6.2 MSR Codes Construction from Vector Spaces

Our code construction is based on the idea shown in Table 5.5. We pick up generator vectors for each node one by one, avoiding those already contained in previous subspaces to ensure data reconstruction. On the other hand, certain dependencies between the generator vectors must exist for data repair.

The first k nodes are always easy, as we can pick an arbitrary basis set and distribute the B basis vectors randomly into the k nodes. Then we consider the $(k+1)$-th node, picking $\boldsymbol{g}_{k,0}$ out of $\cup_{i=0}^{k-1} V_i$, where $\cup$ is the union the subspaces viewed as sets of vectors. Then we choose $\boldsymbol{g}_{k,1}$ out of $\cup_{i=0}^{k-1} V_{i,0}$, where $V_{i,0} = V_i + \langle \boldsymbol{g}_{k,0} \rangle$. The process continues till we reach $\boldsymbol{g}_{k,\alpha-1}$, chosen out of $\cup_{i=0}^{k-1} V_{i,\alpha-2}$, where $V_{i,\alpha-2} = V_i + \langle \boldsymbol{g}_{k,0}, \boldsymbol{g}_{k,1}, \ldots, \boldsymbol{g}_{k,\alpha-2} \rangle$. The process then moves on to node $k+1$. In general, for a node $j \in [n] \backslash [k]$, we shall choose $\boldsymbol{g}_{j,\ell}$ out of $\cup_{i \in S_k} V_{i,\ell-1}$, where S_k is any subset of $[j]$ of size k.

Clearly, to pick an element out of a union of subspaces, we have to know the intersections of the subspaces, and enumerate the elements in the union. Unfortunately, the intersections of subspaces soon become intractable when their dimensions increase. Hence we only present a construction for a set of small parameters, $(n = 5, \alpha = 2, M = 4; k = 2, r = 3)$. However, throughout the construction process, we can still show some generality of the construction. Another interesting property of this special case is that the data repair is naturally fulfilled, though the construction above only focuses on data reconstruction.

5.6.3 Data Reconstruction

It can be seen that following this construction, the data reconstruction requirement will be satisfied naturally, as the construction process guarantees that the $k\alpha$ generator vectors from any k nodes are linearly independent. But before we jump to this conclusion, we have to show that there are indeed enough vectors left to choose from, such that subspaces V_i's exist for $i \in [5]$.

Note that the construction method above does not impose any special constraint on how to choose the vectors for V_i's, as long as they are out of a certain set. In other words, we only need to count the number of vectors included in some sets, instead of enumerate them. Hence all the V_i's are equivalent for any $i \in [5]$. We will examine the intersection and union properties displayed by any two and three of them, as stated below. Since $\alpha = 2$ in this case, we still use $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ to denote the generator vectors of node i, and define $V_{i,\boldsymbol{x}_\ell} = V_i + \langle \boldsymbol{x}_\ell \rangle$ for $i, \ell \in [5]$.

Lemma 14. *dim*$(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell}) = 2$*, where* $i, j, \ell \in [5]$ *and are mutually distinct.*

Proof. $\dim(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell}) = \dim(V_{i,\boldsymbol{x}_\ell}) + \dim(V_{j,\boldsymbol{x}_\ell})$ - dim $(V_{i,\boldsymbol{x}_\ell} + V_{j,\boldsymbol{x}_\ell}) = 3+3-4 = 2$, as $V_{i,\boldsymbol{x}_\ell} + V_{j,\boldsymbol{x}_\ell} = V_i + \langle \boldsymbol{x}_\ell \rangle + V_j + \langle \boldsymbol{x}_\ell \rangle = V_i + V_j = V$, leading to $\dim(V_{i,\boldsymbol{x}_\ell} + V_{j,\boldsymbol{x}_\ell}) = 4$. □

Furthermore, we can say something more about what is contained in the intersection of the two subspaces.

Lemma 15. *There exists a nonzero vector* $\boldsymbol{x} \in (V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})$ *such that* $\boldsymbol{x} \in V_i$ *and* $\boldsymbol{x} + \boldsymbol{x}_\ell \in V_j$.

Proof. Since $\dim(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell}) = 2$, there exists a nonzero $\boldsymbol{x}^* \in V_{i,\boldsymbol{x}_\ell}$ and $\boldsymbol{x}^* \neq \boldsymbol{x}_\ell$ such that $(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell}) = \langle \boldsymbol{x}^*, \boldsymbol{x}_\ell \rangle$. If all the three nonzero elements $\boldsymbol{x}^*, \boldsymbol{x}_\ell, \boldsymbol{x}^* + \boldsymbol{x}_\ell$

are out of V_i, then V_i and $(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})$ have trivial intersection. Hence $\dim(V_i + (V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})) = \dim(V_i \oplus (V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})) = \dim(V_i) + \dim((V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})) = 2$, a contradiction. Hence there exists $\boldsymbol{x} \in (V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})$ which is also in V_i.

Similarly, there exists a nonzero $\boldsymbol{y} \in (V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})$ such that $\boldsymbol{y} \in V_j$. From our construction of $\boldsymbol{x}_\ell$, we know that $\boldsymbol{y} \neq \boldsymbol{x}_\ell$. Further, if $\boldsymbol{y} = \boldsymbol{x} \in V_i$, we would have a nonzero $\boldsymbol{y}$ in both V_i and V_j, contradicting our construction rule that $V_i \cap V_j = \emptyset$. Hence $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{x}_\ell$. □

Now consider the intersection of any three of the subspaces spanned by any three nodes, as stated in Lemma 16.

Lemma 16. *$\dim(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell} \cap V_{k,\boldsymbol{x}_\ell}) = 1$, where $i, j, k, \ell \in [5]$ and are mutually distinct.*

Proof. From Lemma 15, we have a nonzero $\boldsymbol{x} \in V_i$ such that $\langle \boldsymbol{x}, \boldsymbol{x}_\ell \rangle = (V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell})$. Similarly, we have a nonzero $\boldsymbol{z} \in V_k$ such that $\langle \boldsymbol{z}, \boldsymbol{x}_\ell \rangle = (V_{j,\boldsymbol{x}_\ell} \cap V_{k,\boldsymbol{x}_\ell})$. If $\dim(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell} \cap V_{k,\boldsymbol{x}_\ell}) = 2$, we must have $\langle \boldsymbol{x}, \boldsymbol{x}_\ell \rangle = \langle \boldsymbol{z}, \boldsymbol{x}_\ell \rangle$, as both of them have dimension 2, and are subspaces of $(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell} \cap V_{k,\boldsymbol{x}_\ell})$. Hence either $\boldsymbol{x} = \boldsymbol{z} \in V_k$ or $\boldsymbol{x} = \boldsymbol{z} + \boldsymbol{x}_\ell \in V_j$, contradicting the trivial intersection constraints among any pair of V_i, V_j, V_k. Hence $\dim(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell} \cap V_{k,\boldsymbol{x}_\ell}) \leq 1$. Given this subspace contains a nonzero vector $\boldsymbol{x}_\ell$, we have $\dim(V_{i,\boldsymbol{x}_\ell} \cap V_{j,\boldsymbol{x}_\ell} \cap V_{k,\boldsymbol{x}_\ell}) = 1$. □

After the intersection properties are clear, we can count the number of elements included in previous subspaces (nodes), and use what's left to construct the rest nodes. After selecting an arbitrary basis and distribute the basis vectors into node 0 and node 1, a total of $2 \cdot 2^2 - 1 = 7$ elements will be covered by $V_0 \cap V_1$ since the two have trivial intersections. Hence we have $2^4 - 7 = 9$ elements left for $\boldsymbol{x}_2$. After

picking one of them randomly, we will have $V_{0,\boldsymbol{x}_2} \cap V_{1,\boldsymbol{x}_2}$ containing $2 \cdot 2^3 - 2^2 = 12$ elements covered, and $\boldsymbol{y}_2$ has to be chosen from the rest of $2^4 - 12 = 4$ vectors.

Next, we choose $\boldsymbol{x}_3$ out of V_0, V_1, V_2. Since they have trivial intersections with each other, we have $2^4 - 3 \cdot 2^2 + 2 = 6$ choices. To pick $\boldsymbol{y}_3$, we consider $V_{0,\boldsymbol{x}_3} \cup V_{1,\boldsymbol{x}_3} \cup V_{2,\boldsymbol{x}_3}$, which has a total of $3 \cdot 2^3 - \binom{3}{2}2^2 + \binom{3}{3}2^1 = 14$, where $\binom{3}{2}$ is the number of combinations of $V_{i,\boldsymbol{x}_3}$ and $V_{j,\boldsymbol{x}_3}$ with intersection a 2-dimensional subspace. Eventually we still have $2^4 - 14 = 2$ possible vectors to assign to $\boldsymbol{y}_3$.

Finally, we can choose from $2^4 - 4 \cdot 2^2 + 3 = 3$ vectors not covered by the union of V_i's to form V_4, where $i \in [4]$. Note that V_4 itself is a 2-dimensional subspace with 3 nonzero vectors, with trivial intersection with any previous V_i's, hence we can choose any two out of these three vectors as $\boldsymbol{x}_4$ and $\boldsymbol{y}_4$ respectively.

Note that after V_4 is obtained, we will have $5 \cdot 2^2 - 4 = 16$ vectors occupied by the 5 subspaces, which is exactly the total number of vectors in V. Hence we cannot get a code with larger parameters over GF(2) with this parameter settings.

5.6.4 Data Repair

We will show that the code construction procedure above, which focuses on data reconstruction, also naturally satisfies data repair requirement of regenerating codes, leading to MSR codes under our parameter setting.

Lemma 17. *Any node in the DSS constructed above can be repaired by any other three nodes.*

As we stated, no specific constraints are imposed to individual nodes aside from their trivial intersection property, hence all the nodes are equivalent. Without loss of generality, we consider repairing node ℓ from node i, j, k.

Proof. Consider $V_{i,\boldsymbol{x}_j} \cap V_{\ell,\boldsymbol{x}_j}$ from V_i, V_ℓ and $\langle \boldsymbol{x}_j \rangle$. From the proof of Lemma 15, we have a nonzero $\boldsymbol{x}_i^* \in V_i$ and $\boldsymbol{x}_\ell^* \in V_\ell$ such that $\langle \boldsymbol{x}_i^*, \boldsymbol{x}_j \rangle = \langle \boldsymbol{x}_\ell^*, \boldsymbol{x}_j \rangle$ and $\boldsymbol{x}_\ell^* = \boldsymbol{x}_i^* + \boldsymbol{x}_j$. Similarly, for $V_{k,\boldsymbol{x}_j} \cap V_{\ell,\boldsymbol{x}_j}$, we can have $\boldsymbol{y}_\ell^* = \boldsymbol{x}_k^* + \boldsymbol{x}_j$ for some $\boldsymbol{y}_\ell^* \in V_\ell$ and $\boldsymbol{x}_k^* \in V_k$. Further, we have $\boldsymbol{x}_\ell^* \neq \boldsymbol{y}_\ell^*$, from trivial intersection requirement, which is also shown in Lemma 15. Hence if both $\boldsymbol{x}_\ell, \boldsymbol{y}_\ell \in \{\boldsymbol{x}_\ell^*, \boldsymbol{y}_\ell *\}$, the repair is done by the three symbols $\boldsymbol{x}_i^*, \boldsymbol{x}_j, \boldsymbol{x}_k^*$. Otherwise, we'll have cases like $\boldsymbol{x}_\ell = \boldsymbol{x}_\ell^* = \boldsymbol{x}_i^* + \boldsymbol{x}_j$ but $\boldsymbol{x}_\ell + \boldsymbol{y}_\ell = \boldsymbol{y}_\ell^*$. In this case, we can get $\boldsymbol{y}_\ell = \boldsymbol{x}_\ell^* + \boldsymbol{y}_\ell^* = \boldsymbol{x}_i^* + \boldsymbol{x}_k^*$, i.e., the two symbols $\boldsymbol{x}_\ell, \boldsymbol{y}_\ell$ can be repaired by $\boldsymbol{x}_i^*, \boldsymbol{x}_j, \boldsymbol{x}_k^*$, still three helping symbols. □

5.6.5 Discussions

Clearly, if we pick any 4 nodes from the example given, we get a class of $(4, 2, 4; 2, 3)$ MSR codes. Also, no restriction is mentioned on the field of the data or encoded symbols. Actually symbols from any extension fields of GF(2) will share the same data reconstruction property, and simple data repair by XOR operations.

Another interesting fact is that the result of the construction process above is actually a non-overlap partition of the nonzero vectors of V into 5 nodes, each spanning a two-dimensional subspace of V. Hence the code shown in Table 5.5 is only one specific code obtained from our construction. In this sense, our construction is a general approach.

For other parameters with arbitrary α, k, r parameters, code construction can be very complicated as behaviors of subspaces may be very hard to track with high dimensions. This also rises an open problem in our construction, which is passive in the sense that no constraints is imposed on data repair during the construction. If we can design the combinations to form the vectors to be picked in latter nodes,

we may have control over intersection properties of the subspaces.

5.7 Conclusion

In this chapter, we propose three code constructions aiming at different features for data repair in the DSS, MDS codes with low computational complexity, LRC codes for local repair with optimal minimum distance, and MSR codes for minimum repair bandwidth. All the three classes of codes can be constructed based on a vector space analysis. The new MDS codes benefit from the utilization of linearized polynomials, invoking linear repair and quadratic reconstruction. Our LRC codes have a two-layer structure, which generalize existing constructions as special cases. The structure also facilitates efficient degraded reads in the DSS. Our construction for MSR codes are not complete as explicit codes are only obtained for small parameters. However, if we consider extra constraints when choosing basis vectors for latter nodes, this approach may also lead to codes with larger parameters.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, we investigate advanced error control schemes for random linear network coding (RLNC). We first propose a general interpolation algorithm to decode subspace codes used for error correction in both coherent and noncoherent RLNC, such as KK codes, MV codes and that from lifting of rank metric codes. Then based on an approach that transforms the decoding of RLNC into that of linear block codes, we further utilize linear programming (LP) to implement the corresponding decoder, and formulate efficient LP decoding algorithms for codes defined over nonbinary finite fields. Finally, we consider error control schemes for a particular type of network, the distributed storage system (DSS), and propose code constructions and decoding algorithms for different optimization goals. Theoretical feasibility and complexity analysis are provided for our encoding and decoding algorithms, as well as examples of our code construction and decoding procedure. We

also present hardware implementations to prove the efficiency of our interpolator.

Our work was published in the following conferences and journals [18, 64, 65, 75–83]. We briefly summarize our main contributions in this dissertation as follows.

In Chapter 2, we investigate interpolation in a free module of a **linearized polynomial ring**, in parallel to the work of Wang *et al.* [25], and propose a polynomial complexity interpolation algorithm in a well ordered free module of a linearized polynomial ring. This interpolation algorithm is applied to decode subspace codes used in random linear network coding, such as KK codes [9], MV codes [10], and that from lifting of Gabidulin codes [11, 12]. For Gabidulin codes, our interpolation algorithm always produces the correct decoding results compared to some existing approach. It also has a lower complexity for KK and MV codes, compared to Gaussian elimination, currently the only algorithm existing aside from ours.

In Chapter 3, we extend the work of [10] on list decoding of MV codes, in terms of greater decoding radius and analytical performance evaluation. We first remove the impractical assumption of no erasures in [10], and derive the condition of decodability in the presence of both errors and erasures. Analysis on the asymmetric importance of the two types of error patterns in the new decodability condition is also provided. Then we attempt to achieve a greater decoding radius for high rate codes by introducing multiplicity into the interpolation step. However, our results show that the decoding radius is slightly reduced, and we point out the reason lies in some nasty property of linearized polynomials. Finally, based on the results of [43], we obtain the decoder error probability (DEP) of a nearest neighbor decoder, after the decoding list is obtained.

In Chapter 4, based on the work of Yan *et al.* [18], where the decoding of linear

network coding was transformed into classic decoding problems of linear block codes named rank deficient decoding, we further propose their implementation by linear programming. Hence the problem turns into the formulation of decoding linear block codes into solving linear equations. We adapt Yang's formulation in [50] of reduced complexity for binary codes to accommodate both even and odd parity equations, and then propose a simplified formulation for codes over nonbinary fields. We prove that our LP algorithm has the desired ML certificate property, i.e., whenever our LP decoder outputs a codeword, it is the ML codeword. Simulation results who that LP decoding algorithm performs the rank deficient decoding algorithms efficiently.

In Chapter 5, we construct linear codes for distributed storage systems (DSS), viewed from a vector space's perspective. Encoded symbols in each node correspond to a subspace of the linear transformation space that defines the code, and then data reconstruction and repair from a number of nodes are formed into intersection properties of the corresponding subspaces. Using this approach, we propose constructions of three codes for DSSs, maximum distance separable (MDS) codes, minimum storage regenerating (MSR) codes and locally repairable codes (LRC), for low computational complexity, optimized repair bandwidth and locality, respectively. We prove the optimality of the codes obtained from our approach, and also present explicit constructions. We also show that our LRC code structure induces efficient degraded reads [1] in practice.

6.2 Future Work

For future work, the following points may be worthy to be examined:

- The interpolation algorithm we investigated in Chapter 2 works over a linearized polynomial ring. It has two major differences with the ordinary polynomial interpolation [25]. First, the polynomial ring is commutative, while the linearized one not. Second, the definitions of multiplication between models and rings are different for the two cases. However, the two algorithms do share some basic principles, such as a free module, a total ordering, linear functionals, etc, so as to find a minimum element satisfying certain constraints. Hence it would be reasonable to device a general interpolation algorithm that can accommodate these two algorithms as special cases.

- In Chapter Chapter 3, we found that applying multiplicities to the interpolation points directly does not produce an expanded decoding radius as for RS codes in [20]. As we analyzed, the reason is the fact that multiplicities of different interpolation points have to be the same for multivariate linearized polynomials. As a result, not enough extra linear constraints are invoked from the introduction our multiplications. Hence an interesting topic is to define a new multiplication for linearized polynomials, such that the number of new linear equations is at least proportional to the number of points to be interpolated.

- The linear programming algorithm over nonbinary field we propose in Chapter 4 has a reduced complexity compared to that in [21]. However, given the high computational complexity of large finite field, there are still too many calculations involved, which hinders its application in practice. One possible way to reduce the complexity is to shrink the number of linear equations used

to represent the constraints over finite fields. For our LP formulation over extension fields of GF(2), the number of binary equations produced by one parity check equation over the extension field is exponential with respect to the dimension of the field. Naturally, one way to reduce the complexity is to reduce the number of binary equations used to represent the constraints over the extension field. As a tradeoff, the performance will be sacrificed. Fortunately, the rank deficient decoding in [18] introduces an embedded error control mechanism for the underlying RLNC. How to utilize this property to compensate the loss from the simplified LP deserves further examination.

- In Chapter 5, we present some code constructions for DSS, where the requirement of data repair and reconstruction on the nodes are transformed into properties of corresponding subspaces. However, our current results are not complete in two aspects. First, code constructions for MSR codes only work with small parameters, as intersection properties of subspaces soon become intractable following the increase of subspace dimensions. Part of the reason is that our current construction does not impose any constraints in choosing the basis vectors for subspaces spanned by latter nodes. Hence one possible way to extend this work is to strict our selection of later formed subspaces under some constraints that will satisfy data repair property. Second, the construction for LRC codes requires a large field size, from the sufficient but not necessary condition imposed on the first layer MDS codes. How to loose the constraint while still maintain the desired data reconstruction parameters is another way to derive practical codes for DSS.

Bibliography

[1] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for could file system: Minimizing I/O for recovery and degraded reads," in *10th USENIX Conference on FIle and Storage Technologies (FAST'12)*, San Jose, CA, Feb. 2012.

[2] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE Trans. Info. Theory*, vol. 46, pp. 1204–1216, July 2000.

[3] T. Ho, M. Médard, R. Kötter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Info. Theory*, vol. 52, no. 10, pp. 4413–4430, October 2006.

[4] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Info. Theory*, vol. 56, no. 9, pp. 4539–4551, September 2010.

[5] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Info. Theory*, vol. 49, no. 2, pp. 371–381, February 2003.

[6] N. Cai and R. W. Yeung, "Network coding and error correction," in *Proc. IEEE Information Theory Workshop*, Bangalore, India, October 2002, pp. 20–25.

[7] R. W. Yeung and N. Cai, "Network error correction, part I: Basic concepts and upper bounds," *Commun. Info. Syst.*, vol. 6, no. 1, pp. 19–36, 2006.

[8] N. Cai and R. W. Yeung, "Network error correction, part II: Lower bounds," *Commun. Info. Syst.*, vol. 6, no. 1, pp. 37–54, 2006.

[9] R. Kötter and F. R. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Trans. Info. Theory*, vol. 54, no. 8, pp. 3579–3591, August 2008.

[10] H. Mahdavifar and A. Vardy, "Algebraic list-decoding on the operator channel," in *Proc. IEEE Int. Symp. Info. Theory*, Austin, USA, June 2010, pp. 1193–1197.

[11] D. Silva, F. R. Kschischang, and R. Kötter, "A rank-metric approach to error control in random network coding," *IEEE Trans. Info. Theory*, vol. 54, no. 9, pp. 3951–3967, September 2008.

[12] E. M. Gabidulin, "Theory of codes with maximum rank distance," *Problems of Information Transmission*, vol. 21, no. 1, pp. 1–12, January 1985.

[13] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," *Proceedings of 2005 IEEE Infocom*, vol. 4, pp. 2235–2245, March 2005.

[14] S. Deb, M. Médard, and C. Choute, "Algebraic gossip: a network coding approach to optimal multiple rumor mongering," *IEEE Trans. Info. Theory*, vol. 52, no. 6, pp. 2486–2507, June 2006.

[15] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Médard, "The importantance of being opportunistic: practical network coding for wireless environments," September 2005.

[16] J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Médard, "Codecast: a network-coding-based ad hoc multicast protocol," vol. 13, no. 5, pp. 76–81, October 2006.

[17] Z. Liu, C. Wu, B. Li, and S. Zhao, "Uusee: Large-scale operational on-demand streaming with random network coding," pp. 1–9, March 2010.

[18] Z. Yan, H. Xie, and B. W. Suter, "Rank deficient decoding of linear network coding," in *The 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013*, Vancouver, Canada, May 2013.

[19] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *IEEE INFOCOM 2007*, Anchorage, AK, 2007.

[20] V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon Codes and Algebraic Geometry Codes," *IEEE Trans. Info. Theory*, vol. 45, no. 6, pp. 1757–1767, September 1999.

[21] M. Flanagan, V. Skachek, E. Byrne, and M. Greferath, "Linear-programming decoding of nonbinary linear codes," *IEEE Trans. Info. Theory*, vol. 55, no. 9, pp. 4134–4154, September 2009.

[22] R. M. Roth, *Introduction to Coding Theory.* Cambridge University Press, 2006.

[23] R. Kötter, "Fast Generalized Minimum-Distance Decoding of Algebraic Geometry and Reed-Solomon Codes," *IEEE Trans. Info. Theory*, vol. 42, no. 3, pp. 721–736, May 1996.

[24] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," U.S. Patent no. 4,633,470, December 30, 1986.

[25] B. Wang, R. J. McEliece, and K. Watanabe, "Kötter interpolation over free modules," in *Proc. 2005 Allerton Conf. Communications Control and Computing*, Monticello, IL, October 2005, pp. 2197–2206.

[26] D. Silva and F. R. Kschischang, "On metrics for error correction in network coding," *IEEE Trans. Info. Theory*, vol. 55, no. 12, pp. 5479–5490, December 2009.

[27] P. Lusina, E. M. Gabidulin, and M. Bossert, "Maximum rank distance codes as space-time codes," *IEEE Trans. Info. Theory*, vol. 49, no. 10, pp. 2757–2760, October 2003.

[28] E. M. Gabidulin, A. V. Paramonov, and O. V. Tretjakov, "Ideals over a non-commutative ring and their application in cryptology," in *Proc. Eurocrypt*, Brighton, UK, April 1991, pp. 482–489.

[29] E. M. Gabidulin, "Optimal codes correcting lattice-pattern errors," *Problems of Information Transmission*, vol. 21, no. 2, pp. 3–11, 1985.

[30] P. Loidreau, "A Welch-Berlekamp like algorithm for decoding Gabidulin codes," in *Proc. International Workshop on Coding and Cryptography*, Bergen, Norway, March 2005, pp. 36–45.

[31] R. Lidl and H. Niederreiter, *Finite Fields*, ser. Encyclopedia of Mathematics and its Applications, G. Rota, Ed., 1983, vol. 20.

[32] O. Ore, "On a special class of polynomials," *Transactions of the American Mathematical Society*, vol. 35, pp. 559–584, 1933.

[33] P. Delsarte, "Bilinear forms over a finite field, with applications to coding theory," *Journal of Combinatorial Theory A*, vol. 25, no. 3, pp. 226–241, November 1978.

[34] R. M. Roth, "Maximum-rank array codes and their application to crisscross error correction," *IEEE Trans. Info. Theory*, vol. 37, no. 2, pp. 328–336, March 1991.

[35] R. J. McEliece, "The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes," Interplanetary Network Progress Report 42-153, May 2003.

[36] O. Ore, "Contribution to the theory of finite fields," *Transactions of the American Mathematical Society*, vol. 36, pp. 243–274, 1934.

[37] D. Silva and F. R. Kschishcang, "Fast encoding and decoding of Gabidulin codes," in *Proc. IEEE Int. Symp. Info. Theory*, Seoul, Korea, June 2009, pp. 2858–2862.

[38] M. Gadouleau and Z. Yan, "Complexity of decoding Gabidulin codes," in *42nd Annual Conference on Information Sciences and Systems*, Princeton, USA, March 2008, pp. 1081–1085.

[39] N. Chen, Z. Yan, M. Gadouleau, Y. Wang and B. W. Suter, "Rank metric decoder architectures for random linear network coding with error control," *IEEE Trans. VLSI Systems*, vol. 20, no. 2, pp. 296–309, Feb 2012.

[40] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *Proc. IEEE Int. Conf. Microelectron. Syst. Education (MSE07)*, San Diego, USA, Jun. 2007, pp. 173–174.

[41] H. Mahdavifar and A. Vardy, "Algebraic List-Decoding on the Operator Channel," *submitted to IEEE Trans. Info. Theory*, April 2010.

[42] M. Sudan, "Decoding of Reed-Solomon codes beyond the error-correction bound," *J. Complexity*, vol. 13, pp. 180–193, 1997.

[43] M. Gadouleau and Z. Yan, "On the decoder error probability of rank metric codes and constant-dimension codes," available at http://arxiv.org/abs/0812.2379.

[44] H. Chen, "Distributed file sharing: Network coding meets compressed sensing," *Proceedings of First International Conference on Communications and Networking in China (ChinaCom'06)*, pp. 1–5, October 2006.

[45] S. Katti, S. Shintre, S. Jaggi, D. Katabi, and M. Médard, "Real network coding," *Proceedings of Forty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, pp. 389–395, September 2007.

[46] N. Nguyen, D. Jones, and S. Krishnamurthy, "Netcompress: Coupling network coding and compressed sensing for efficient data communication in wireless

sensor networks," *Proceedings of 2010 IEEE Workshop on Signal Processing Systems (SiPS 2010)*, pp. 356–361, October 2010.

[47] S. Shintre, S. Katti, S. Jaggi, B. K. Dey, D. Katabi, and M. Médard, "'Real' and 'complex' network codes: Promises and challenges," *Fourth Workshop on Network Coding, Theory and Applications (NetCod 2008)*, pp. 1–6, January 2008.

[48] Z. Zhang, "Linear network error correction codes in packet networks," *IEEE Trans. Info. Theory*, vol. 54, no. 1, pp. 209–218, January 2008.

[49] J. Feldman, M. Wainwright, and D. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Info. Theory*, vol. 51, no. 3, pp. 954–972, March 2005.

[50] K. Yang, X. Wang, and J. Feldman, "A new linear programming approach to decoding linear block codes," *IEEE Trans. Info. Theory*, vol. 54, no. 3, pp. 1061–1072, March 2008.

[51] J. Honda and H. Yamamoto, "Fast linear-programming decoding of LDPC codes over GF(2^m)," in *ISITA 2012*, Honolulu, Hawaii, USA, October 2012, pp. 754–758.

[52] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantiative comparison," in *Proc. Int. Workshop Peer-to-Peer Syst.*, Cambridge, MA, USA, March 2002.

[53] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Info. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[54] Y. Wu and A. G. Kimakis, "Reducing repair traiffic for erasure coding-based storage via interference alighment," in *Proc. IEEE Int. Symp. on Information Theory*, Seoul, Korea, Jun. 2009, pp. 2276–2280.

[55] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," 2009, available online at http://arxiv.org/abs/0906.4913v2.

[56] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *available online at http://arxiv.org/abs/1005.1634*, September 2010.

[57] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, March 2011.

[58] C. Suh and K. Ramchandran, "Exact-repair MDS codes for distributed storage using interference alignment," in *IEEE Int. Symp. Info. Theory*, Austin, Texas, USA, June 2010, pp. 161–165.

[59] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix

construction," *IEEE Trans. Info. Theory*, vol. 57, no. 8, pp. 5227–5239, August 2011.

[60] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," in *Proc. IEEE Int. Symp. on Information Theory*, Cambridge, MA, July 2012.

[61] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis, "Optimal locally repairable codes and connection to matroid theory," February 2013, available online at http://arxiv.org/abs/1301.7693v2.

[62] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with local regeneration," in *2013 Information Theory and Applications Workshop (ITA 2013)*, San Diego, USA, Feb. 2013.

[63] A. S. Rawat, N. Silberstein, O. O. Koyluoglu, and S. Vishwanath, "Optimal locally repairable codes with local minimum storage regeneration via rank-metric codes," in *Information Theory and Applications Workshop (ITA)*, San Diego, CA, Feb. 2013.

[64] H. Xie, Z. Yan, and B. W. Suter, "General linearized polynomial interpolation and its applications," in *2011 International Symposium on Network Coding (NetCod11)*, Beijing, China, July 2011.

[65] j. . I. y. . . v. . . n. . . p. . . n. . . m. . J. Hongmei Xie and Jun Lin and Zhiyuan Yan and Bruce W. Suter, title = Linearized Polynomial Interpolation and Its Applications.

[66] M. Blaum and R. M. Roth, "On lowest density MDS codes," *IEEE Trans. Info. Theory*, vol. 45, no. 1, pp. 46–59, January 1999.

[67] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," August 2013, available online at http://arxiv.org/abs/1210.6954v2.

[68] T. P. Berger and A. V. Ourivski, "Construction of new MDS codes from Gabidulin codes," available online at http://www.unilim.fr/pages_perso/thierry.berger/publis/actes/actes-04-acct9-2.pdf.

[69] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *IEEE INFOCOM 2011*, Shanghai, China, April 2011, pp. 1215–1223.

[70] R. M. Roth and A. Lempel, "On MDS codes via Cauchy matrices," *IEEE Trans. Info. Theory*, vol. 35, no. 6, pp. 1314–1319, November 1989.

[71] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," *Technical Report TR-95-048, International Computer Science Institute*, August 1995.

[72] S. E. Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *48th Annual Allerton Conference on Communication, Control, and Computing*, Urbana Champaign, IL, Sep. 2010, pp. 1510–1517.

[73] O. Olmez and A. Ramamoorthy, "Replication based storage systems with local repair," May 2013, available online at http://arxiv.org/abs/1305.5764.

[74] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," in *Proc. 6th IEEE Int. Symp. Netw. Comput. Appl. (NCA)*, 2007, pp. 79–86.

[75] H. Xie, Z. Yan, and B. W. Suter, "On list decoding of mahdavifar – vardy codes," in *2011 International Symposium on Network Coding (NetCod11)*, Beijing, China, July 2011.

[76] J. Lin, H. Xie, and Z. Yan, "Efficient kötter-kschischang decoder architectures for noncoherent error control in random linear network coding," in *2012 IEEE Workshop on Signal Processing Systems (SiPS 2012)*, Qubec City, Canada, Oct. 2012.

[77] Z. Yan and H. Xie, "Enhanced algebraic error control for random linear network coding," in *Military Communications Conference 2012 (Milcom 2012)*, Orlando, USA, Oct. 2012.

[78] H. Xie and Z. Yan, "MDS codes with low repair complexity for distributed storage networks," in *2013 22nd Wireless and Optical Communication Conference (WOCC 2013)*, Chongqing, China, May 16-18 2013.

[79] C. Chen, H. Xie, and B. Bai, "Layered subspace codes for random network coding," *Transactions on Emerging Telecommunications Technologies*, Jun. 2013.

[80] J. Lin, H. Xie, and Z. Yan, "Efficient error control decoder architectures for noncoherent random linear network coding," *The Journal of Signal Processing Systems*, Aug. 2013.

[81] Z. Yan, H. Xie, and B. W. Suter, "Rank deficient decoding of linear network coding," journal draft in preparation.

[82] H. Xie and Z. Yan, "Two-layer locally repairable codes for distributed storage systems," in *The 2014 IEEE International Conference on Communications (ICC'14)*, Sydney, Australia, Jun. 2014.

[83] ——, "Distributed storage code constructions from a vector space approach," journal draft in preparation.

Appendix A

Proof of Lemmas in Chapter 2

Proof of Lemma 1. Suppose both Q^* and Q' have the minimum order in $\overline{K}_C$, with leading terms $\alpha\phi_J$ and $\beta\phi_J$ respectively, where $\alpha, \beta \in \mathrm{GF}(q^m)$. Then there exists a nontrivial linear combination $\beta Q^* - \alpha Q'$, such that $\beta Q^* - \alpha Q' <_o Q^*$. Since $\beta Q^* - \alpha Q' \in \overline{K}_C$, this contradicts the minimality of Q^* and Q'. □

Proof of Lemma 2. We deal with the three cases separately:

1. When $\Delta_{i+1,j} = 0$, $g_{i+1,j} = g_{i,j}$ if $g_{i,j} \in T_{i+1,j}$. Since $g_{i,j}$ is a minimum in $T_{i,j}$ and $T_{i,j} \supseteq T_{i+1,j}$, $g_{i,j}$ is also a minimum in $T_{i+1,j}$.

2. For any $g_{i,j}$ with $j \neq j^*$, $g_{i+1,j} = D_{i+1}(g_{i,j^*})g_{i,j} - D_{i+1}(g_{i,j})g_{i,j^*}$. One can verify that $D_{i+1}(g_{i+1,j}) = 0$, and thus $g_{i+1,j} \in K_{i+1}$. Furthermore, $D_k(g_{i+1,j}) = D_{i+1}(g_{i,j^*})D_k(g_{i,j}) - D_{i+1}(g_{i,j})D_k(g_{i,j^*}) = 0$ for any $k \leq i$, since $g_{i,j}, g_{i,j^*} \in \overline{K}_i$. Hence $g_{i+1,j} \in \overline{K}_{i+1}$. Since $\mathrm{Ind}_y(g_{i+1,j}) = \mathrm{Ind}_y(g_{i,j})$, $g_{i+1,j}$ is also in S_j, thus $g_{i+1,j} \in T_{i+1,j}$. Since $g_{i+1,j} =_o g_{i,j}$ and $g_{i,j}$ is a minimum in $T_{i,j}$, $g_{i+1,j}$ is also a minimum in $T_{i,j}$, hence a minimum in $T_{i+1,j} \subseteq T_{i,j}$.

3. In this case, $g_{i+1,j^*} = D_{i+1}(g_{i,j^*})(x^{[1]} \circ g_{i,j^*}) - D_{i+1}(x^{[1]} \circ g_{i,j^*})g_{i,j^*}$. First note that $D_{i+1}(g_{i+1,j^*}) = 0$, and hence $g_{i+1,j^*} \in K_{i+1}$. For any $k \leq i$, when we apply D_k to g_{i+1,j^*}, we also get zero because both $g_{i,j}$ and $x^{[1]} \circ g_{i,j^*}$ lie in $\overline{K}_i$, as $\overline{K}_i$ is a submodule of $\mathbb{L}[x]$. Thus $g_{i+1,j^*} \in \overline{K}_{i+1}$. Also, $\mathrm{Ind}_y(g_{i+1,j^*}) = \mathrm{Ind}_y(x^{[1]} \circ g_{i,j^*}) = j^*$ by our definition $\mathrm{Ind}_y(l(x) \circ b_j) = j$. Thus we have $g_{i+1,j^*} \in T_{i+1,j^*}$. Next we show that g_{i+1,j^*} is a minimum in T_{i+1,j^*} by contradiction. Suppose there exists $f_{i+1,j^*} \in T_{i+1,j^*}$ such that $f_{i+1,j^*} <_o g_{i+1,j^*}$. Note that $\mathrm{order}(g_{i+1,j^*}) = \mathrm{order}(x^{[1]} \circ g_{i,j^*})$. Since $T_{i,j^*} \supseteq T_{i+1,j^*}$, f_{i+1,j^*} also lies in T_{i,j^*}. Hence $\mathrm{order}(f_{i+1,j^*}) \geq \mathrm{order}(g_{i,j^*})$, as g_{i,j^*} is a minimum in $T_{i,j}$, which results in $\mathrm{order}(g_{i,j^*}) \leq \mathrm{order}(f_{i+1,j^*}) < \mathrm{order}(x^{[1]} \circ g_{i,j^*})$. Since both g_{i,j^*} and $x^{[1]} \circ g_{i,j^*}$ lie in the set S_{j^*} by definition, we can write $\mathrm{LM}(g_{i,j^*}) = x^{[k]} \circ b_{j^*}$ and $\mathrm{LM}(g_{i,j^*}) = x^{[k+1]} \circ b_{j^*}$ for some nonnegative integer k. Similarly, we can write $f_{i+1,j^*} = x^{[k']} \circ b_{j^*}$ for some nonnegative integer k'. Then given the two conditions of the total ordering on M defined previously, there does not exist $f_{i+1,j^*} \in S_{j^*}$ such that $\mathrm{order}(g_{i,j^*}) < \mathrm{order}(f_{i+1,j^*}) < \mathrm{order}(x^{[1]} \circ g_{i,j^*})$, as there does not exist a nonnegative integer k' such that $k < k' < k+1$. Hence the only possibility is that $f_{i+1,j^*} =_o g_{i,j^*}$. But in this case, we could construct $h = \alpha f_{i+1,j^*} + \beta g_{i,j^*}$ with $\alpha, \beta \in \mathrm{GF}(q^m)$ such that $h <_o g_{i,j^*}$. Note that $h \in \overline{K}_i$ but $h \notin \overline{K}_{i+1}$ as $f_{i+1,j^*} \in T_{i+1}$ but $g_{i,j^*} \notin T_{i+1}$. The fact that $h \in \overline{K}_i \backslash \overline{K}_{i+1}$ but $h <_o g_{i,j^*}$ contradicts the minimality of g_{i,j^*} in $\overline{K}_i \backslash \overline{K}_{i+1}$, as g_{i,j^*} has the lowest order among all $g_{i,j}$'s where $g_{i,j} \in \overline{K}_i$ but $g_{i,j} \notin \overline{K}_{i+1}$.

□

Proof of Lemma 4. In the initialization step of Algorithm 1, $g_{0,0} = x$ is of lower order than $g_{0,1} = y$, and $D_1(g_{0,0}) = x_0 \neq 0$ as x_i's are linearly independent, so

$g_{0,0} = x$ updates by the order-increase rule, while $g_{0,1} = y$ updates according to its discrepancy value. Then $g_{1,0}$ is actually a linearized polynomial in x of q-degree 1, and $g_{1,1}$ is a bivariate polynomial with a leading monomial cy, where $c \in \mathrm{GF}(q^m)$ is a constant.

In the second iteration, again $g_{1,0} <_o g_{1,1}$ based on our total ordering on M, and $D_2(g_{1,0}) \neq 0$ as x_i's are linearly independent, i.e., there does not exist a linearized polynomial of q-degree 1 that has two linearly independent roots. Hence $g_{1,0}$ takes the order-increase rule and $g_{1,1}$ adopts others accordingly. Similar situation occurs in all the first k iterations, given the total ordering we defined on M and the fact that $D_{i+1}(g_{i,0}) \neq 0$ for any $i \leq k$.

Finally, a polynomial $g_{k,0}$ in x of q-degree k is derived, which actually only interpolates over the first k x_i's. Note that $N_0(x)$ is obtained in a similar way in Loidreau's algorithm. Given that $V_0(y) = 0$, we have $Q_0 = N_0(x)$. Hence $g_{k,0} =_o Q_0$. On the other hand, $g_{k,1}$ is a bivariate polynomial with a leading monomial $c'y$, where $c' \in \mathrm{GF}(q^m)$ is also a constant. Since $N_1(x)$ is a linear combination of linearized polynomials of q-degree $k-1$, it is a linearized polynomial in x of q-degree at most $k-1$, then the leading monomial of Q_1 is y. As a result, $g_{k,1} =_o Q_1$. □

Vita

Hongmei Xie received the B.E. degree from Qingdao University, Qingdao, China, in 2005, and the M.E. degree from Xidian University, Xi'an, China, in 2008, both in electrical engineering. She is currently pursuing the Ph.D. degree in electrical engineering at Lehigh University, Bethlehem, Pennsylvania.

Her research interests are in coding theory and its applications in communication systems.