Coding for Information Storage

Thesis by

Zhiying Wang

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2013

(Defended March 14, 2013)

To my parents

and

my fiancé Li

Acknowledgements

I would like to acknowledge my advisor Professor Jehoshua (Shuki) Bruck for his constant guidance and support, without which this work would never exist. He is the most wonderful mentor one could hope for. He taught me not only how to do research, but also how to be a hard-working and kind person.

Itzhak Tamo has been a great friend and collaborator, even though I always have problem pronouncing his name. To work with and learn from him is inspiring and fun, and I would like to thank him for his amazing ideas and great help.

I also would like to thank Professor Anxiao (Andrew) Jiang, whom I believe is the most diligent and efficient researcher. Countless new ideas flow from him as the waves flow from the ocean. But more importantly, he cares about his students and friends as if they were family.

I would like to thank Professor Alexandros G. Dimakis, who introduced me to the distributed storage problem. He has so much positive energy in work and life that everybody near him cannot avoid being caught up into it.

I also would like to thank my friends and colleagues from the Paradise Lab: Professor Yuval Cassuto, Robert Mateescu, Dan Wilhelm, Hongchao Zhou, Eyal En Gad, and Eitan Yaakobi. Only because of them has life in an unfamiliar city been filled with joy and excitement.

Last but not least, I would like to thank my thesis committee: Professor Michelle Effros, Professor Babak Hassibi, Professor Tracey Ho, and Professor Erik Winfree. Their illuminating questions and comments provide me with different points of view and new ways of thinking. More importantly, their encouragement and help inspires me to further on my path of research.

Abstract

Storage systems are widely used and have played a crucial rule in both consumer and industrial products, for example, personal computers, data centers, and embedded systems. However, such system suffers from issues of cost, restricted-lifetime, and reliability with the emergence of new systems and devices, such as distributed storage and flash memory, respectively. Information theory, on the other hand, provides fundamental bounds and solutions to fully utilize resources such as data density, information I/O and network bandwidth. This thesis bridges these two topics, and proposes to solve challenges in data storage using a variety of coding techniques, so that storage becomes faster, more affordable, and more reliable.

We consider the system level and study the integration of RAID schemes and distributed storage. Erasure-correcting codes are the basis of the ubiquitous RAID schemes for storage systems, where disks correspond to symbols in the code and are located in a (distributed) network. Specifically, RAID schemes are based on MDS (maximum distance separable) array codes that enable optimal storage and efficient encoding and decoding algorithms. With r redundancy symbols an MDS code can sustain r erasures. For example, consider an MDS code that can correct two erasures. It is clear that when two symbols are erased, one needs to access and transmit all the remaining information to rebuild the erasures. However, an interesting and practical question is: What is the smallest fraction of information that one needs to access and transmit in order to correct a single erasure? In Part I we will show that the lower bound of $1/2$ is achievable and that the result can be generalized to codes with arbitrary number of parities and optimal rebuilding.

We consider the device level and study coding and modulation techniques for emerging non-volatile memories such as flash memory. In particular, rank modulation is a novel data representation scheme proposed by Jiang et al. for multi-level flash memory cells, in which a set of n cells stores information in the permutation induced by the different charge levels of the individual cells. It eliminates the need for discrete cell levels, as well as overshoot errors, when programming cells. In order to decrease the decoding complexity, we propose two variations of this scheme in Part II:

bounded rank modulation where only small sliding windows of cells are sorted to generated permutations, and *partial rank modulation* where only part of the n cells are used to represent data. We study limits on the capacity of bounded rank modulation and propose encoding and decoding algorithms. We show that overlaps between windows will increase capacity. We present Gray codes spanning all possible partial-rank states and using only "push-to-the-top" operations. These Gray codes turn out to solve an open combinatorial problem called *universal cycle*, which is a sequence of integers generating all possible partial permutations.

Contents

List of Figures

Chapter 1

Introduction

Information theory studies the compression, the storage, and the communication of information. Among these topics, the storage of data has attracted a large amount of interest among both industrial and academic researchers in the era of information explosion, especially with the rapid development of the Internet. Moreover, the implementation of distributed storage and the success of new storage devices, such as flash memory, also provide people with new media for more data storage. Therefore, two of the most exciting topics in information theory today are information representation in distributed storage and flash memory, which will also be the two subjects of this thesis.

Network solution providers such as Cisco predicted that the amount of Internet data traffic will increase by 32% every year from 2011 to 2016 [Cis12] (see Figure 1.1). And the overall data will reach 45000 petabytes per month next year (year 2014). This also indicates the huge demands for data storage in the near future. Therefore, it is an urgent challenge to invent reliable, inexpensive, high-performance and power-efficient information storage.

Flash memory has emerged as a strong competitor to the traditional HDD (hard disk drive) in the past decade. Compared to the hard disks, flash memory does not have any spin parts, leading to faster access speed, more flexible fragmentation, and better durability in terms of physical collision. More importantly, because of the scaling-down of the advanced CMOS technology indicated by Moore's law, flash memory has the potential to have higher density and lower price. Therefore, it is very suitable for consumer mobile devices such as cameras, cell phones, and tablets. The vast demand for these products in turn pushes forward the flash technologies. As a result, flash devices have evolved from discrete small storage units into the primary drives in computers and servers. Moreover, flash memory has been developed as an alternative to HDD in enterprise applications such as servers and data centers, once it meets the specifications of industrial standards. Figure 1.2 shows

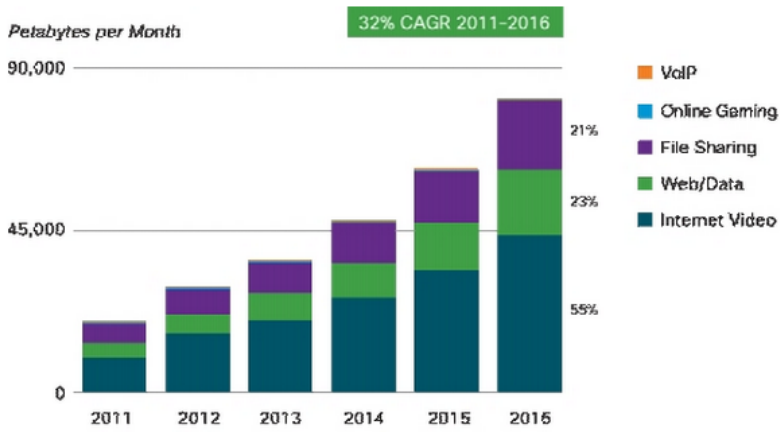

Figure 1.1: The amount of data traffic in the recent years. The compound average growth rate (CAGR) is 32% every year. Source: Cisco [Cis12].

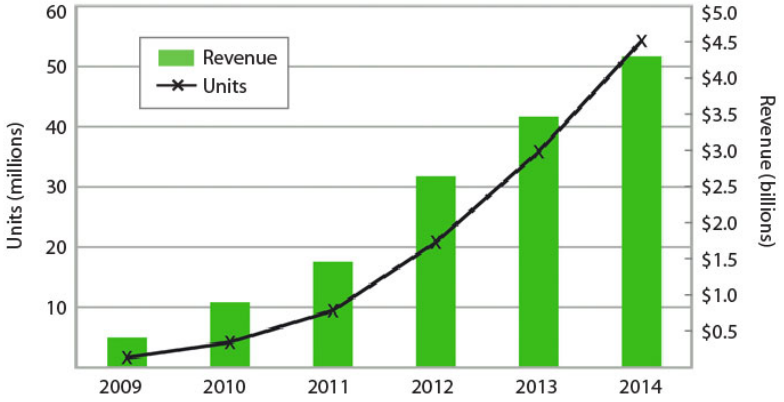

Figure 1.2: The trend of SSD in terms of unit and revenue. Source: SanDisk [Ore11].

the trend of SSD (solid state drive) in terms of unit and revenue estimated by SanDisk [Ore11]. Projections show that the number of SSD units will jump to 50 million by 2014.

Meanwhile, with the development of mobile devices assisted by flash memory, more and more data storage in the servers is required by applications such as video, file sharing, social network, and gaming. Therefore, data centers has expanded from the size of a room to that of a warehouse. Moreover, distributed or cloud storage has also changed people's life and the way of computing. Data is stored and backed up across the Internet instead of in a single place. Its advantage is obvious: more resistant to unpredictable failures, such as natural disasters and electrical black out. Also data can be retrieved from convenient locations to save access time and reduce network traffic. Figure 1.3 is from Seagate [Woj12] and shows that by the year 2014 there will be about 775 million subscribers of cloud storage, which means 2325 Petabytes of storage.

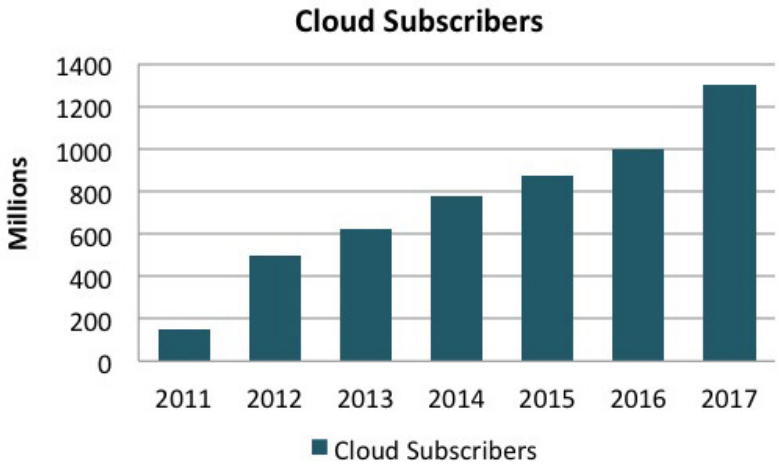

Figure 1.3: The development of cloud storage in terms of number of subscribers. Source: Seagate [Woj12].

In distributed storage, as the system scales failures are inevitable. Hence redundant data must be stored in order to combat errors. Erasure correction codes are the most commonly used technique for this purpose. When a failure happens and repair is executed, a large data traffic in the network as well as large data I/O in each storage unit is generated. In fact, [SAP+13] shows that in a system with 3000 servers (or nodes) of Facebook, it is quite typical to have 20 or more node failures per day that triggers repair jobs (Figure 1.4). Moreover, with the current configuration the repair traffic is estimated around 10 to 20 percent of the total average of 2 petabytes per day cluster network traffic. Therefore, it is crucial to study practical solutions that repair with low traffic cost. Meanwhile, these challenges in distributed storage can be similar to those in a single data center. In such cases, our work can apply to both scenarios.

In Part I we study the rebuilding of distributed storage first proposed by [DGW+10]. Figure 1.5 shows an example of an erasure correction code called EVENODD [BBBM95] with four nodes. One can easily verify the following properties of this code. (i) *Node size*: each node has size two. (ii) *Reconstruction*: if we lose any two nodes the information bits a, b, c, d can be still computed from the rest nodes or 4 bits. (iii) *Rebuilding*: if only one node fails, transmitting only 3 bits we are able to solve the lost bits. Sometimes we can access or read some bits and directly transmit them such as Figure 1.5 (a). In other cases we need to access and compute before we transmit, such as bit $a+b+c+d$ in (b), where the number of accesses is larger than the number of transmitted bits.

While erasure correction codes help with distributed storage, coding techniques also benefits information storage in flash memory. In spite of the attempting advantages of flash memory, it still

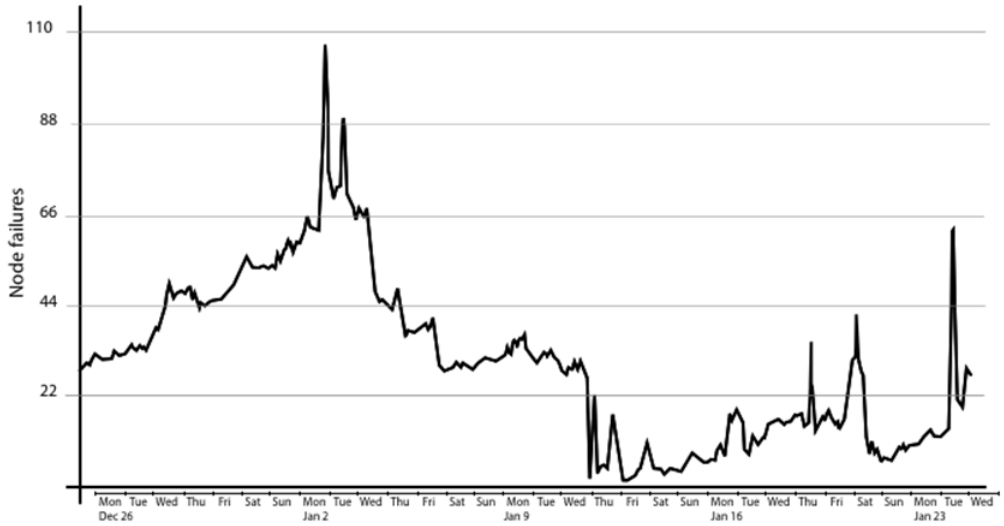

Figure 1.4: Number of failed nodes in a month in a distributed storage with 3000 nodes of Facebook [SAP+13].

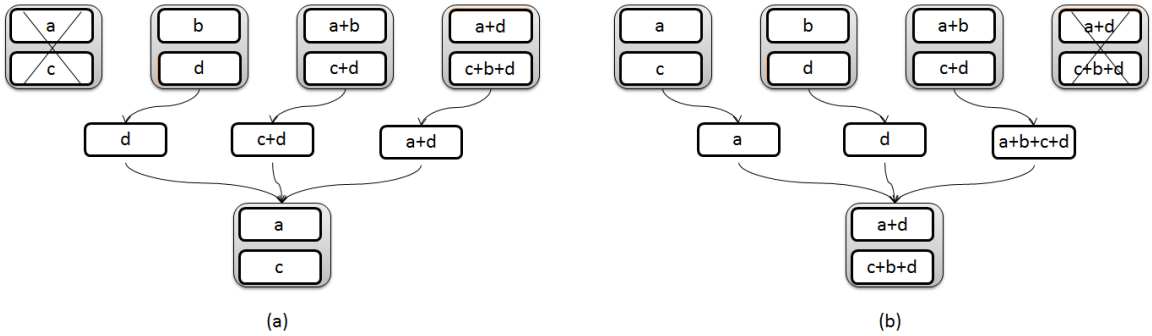

Figure 1.5: An erasure correction code (EVENODD) with two information and two redundant nodes. All symbols are binary and all additions are XOR. Any two node failures can be corrected. For a single failure, only 3 symbols are transmitted to rebuild (a) first node or (b) last node. In (b) the bit $a+b+c+d$ is obtained by XOR of the two bits $a+b$ and $c+d$.

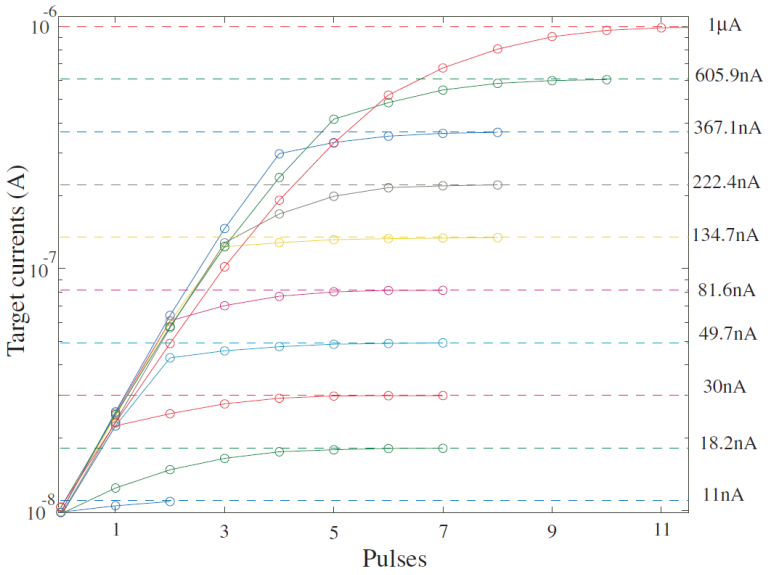

Figure 1.6: Iterative programming towards different targets (dashed lines). The circles in the solid lines shows the result after each iteration [BSH05].

suffers from limited life time (10^5 write/erase cycles), read/write disturbances, limited retention, and high cost of erasure. For example, due to the high cost of erasure, decrease a data value is very expensive. As a result, the writing of data is processed in a very conservative interactive method in current flash technology. Figure 1.6 is from [BSH05] and shows that the average number of pulses required to hit a target in this example is 7 to 8. Here the current target corresponds to the desired value to store. One can see that writing can be very slow and thus degrades the performance of the system. For the consumer applications, these problems of flash are not critical because the requirement for the performance are topically not very high. However, they have to be solved in order to satisfy the enterprise requirements such as frequent fast access, long lifespan, and high reliability.

In Part II, we mainly address the problem of coding for flash memory. Specifically, we focus on a novel data representation technique called *rank modulation* [JMSB09], where permutations of length n are used to represent data. Please refer to Figure 1.7 as an example. Here each cylinder is a flash cell and the filled part represents its level of electric charge or current. In (a), each cell represents one bit by two fixed target levels. As mentioned above, reaching a specific level will incur iterative programming and slow writing. On the other hand, (b) uses the ranks of two cells to represent one bit. Once the permutation or the relative values are obtained, cells can be programmed with more freedom. If the lower cell is programmed too high, instead of lowering a cell level which

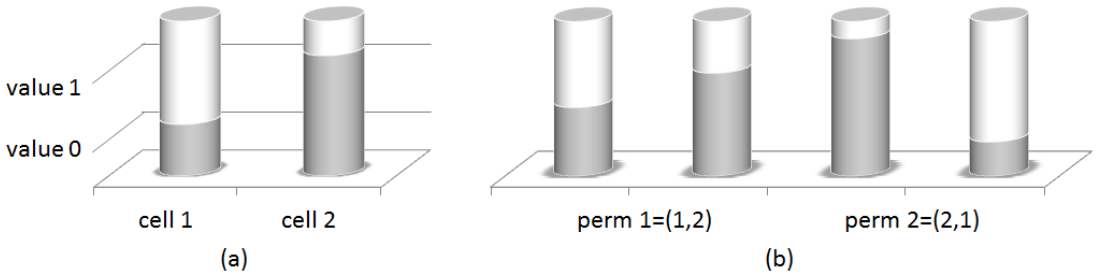

Figure 1.7: Information representation schemes in flash memory. (a) Absolute-value representation. Every cell must fall in one of the two target levels. Cell 1 and 2, respectively, represent value 0 and 1. (b) Rank modulation of length 2. Every two cells constitute a permutation. If the first cell is lower than the second, the permutation is (1,2) and represents value 0; otherwise the permutation is (2,1) and represents value 1.

is very expensive, the other cells will be merely increased.

This thesis strives to provide easier, faster, and more reliable ways to store data with the help of coding techniques. The rest of the thesis is divided into two parts: coding for distributed storage and coding for flash memory.

The rebuilding access or transmission problem of distributed storage will be more thoroughly explained in Chapter 2. Our contributions are inventing efficient rebuilding algorithm for existing codes in Chapter 3, constructing optimal rebuilding codes, zigzag codes, and their extensions in Chapter 4 and 5, rebuilding multiple failures in Chapter 6, and studying the relationship of the number of nodes and the node size in Chapter 7.

Chapter 8 gives a more detailed introduction to rank modulation in flash memory. We will study rank modulation with more constraints on the permutation size, e.g., bounded rank modulation in Chapter 9 and partial rank modulation in Chapter 10. Error correction in rank modulation will be discussed in Chapter 11.

Finally, Chapter 12 concludes the thesis with summary and future directions.

Part I

Coding for Distributed Storage

Chapter 2

Introduction to the Rebuilding Problem

Distributed storage systems involving storage nodes connected over networks have become one of the most popular architecture in current file systems. MDS (maximum distance separable) codes can be used for erasure protection in distributed storage systems where encoded information is stored in a distributed manner. A code with r parity (redundancy) nodes is MDS if and only if it can recover from any r erasures. For example, Reed-Solomon codes [RS60] are the most commonly seen MDS codes.

Moreover, a more general framework of file storage can be a collection of storage nodes located in a distributed or a centralized manner. RAID (redundant array of independent disks) is such a storage technique and it combines multiple disk drive components into a logical unit. MDS array codes are used extensively as the basis for RAID storage systems. An array code consists of a 2-D array where each column can be considered as a disk. We will use the term column, node, or disk interchangeably. We call the entries in the array symbols, elements, or blocks. Examples of MDS array codes are EVENODD [BBBM95, BBV96], B-code [XBBW99], X-code [XB99], RDP [CEG$^+$04], and STAR-code [HX08].

If no more than r storage nodes are lost, then all the information can still be recovered from the surviving nodes. In order to correct r erasures, it is obvious that one has to access (or read) and transmit the information in all the surviving nodes. However, in practice it is more likely to encounter a single erasure rather than r erasures. Suppose one node is erased, and instead of retrieving the entire file, if we are only interested in repairing the lost node, then what is smallest amount of transmission or access needed? The amount of transmission is called the *repair bandwidth* and the amount of access if called *rebuilding access*. Assume a file of size $\mathcal{M}$ is stored in n nodes, each storing a piece of size $\mathcal{M}/k$. Further assume we have access to d of the surviving nodes, $k \leq d \leq n-1$. This *repair problem* was first raised in [DGW$^+$10], and a cut-set lower bound for

repair bandwidth of one node erasure was derived:

$$\frac{\mathcal{M}d}{k(d-k+1)}. \tag{2.1}$$

Besides bandwidth and access, there are quite a few other key features of an array code in storage. We list some of them below, and in the following chapters we will study in more details on them for code constructions.

Systematic. A code is systematic if the information is stored exclusively in the first k nodes (systematic nodes), and the redundancies are stored exclusively in the last r nodes (parity nodes). The advantage of such codes is that information can be easily obtained from only the first k nodes.

Finite-Field Size It is the size of the finite field where all entries or symbols in the array belong to. One reason of the popularity of array code is its small field size and hence low computational complexity.

Update. It is the number of symbol rewrites when an information symbol is updated. It also equals the number of appearances of a symbol in the array code. If the code is MDS, any information symbol appears at least $r+1$ times, because otherwise we can erase those nodes containing this symbol and violate the MDS property. If each information symbol appears only $r+1$ times, we say the code is *optimal update*.

Bandwidth. It is the smallest amount of data transmission for rebuilding a node erasure. If a code matches the bound (2.1) we say it is *optimal bandwidth*. When the file size and the number of nodes is large, rebuilding is a common operation and takes up a large portion of network resources. Small bandwidth indicates low traffic consumption in the rebuilding process and is desirable for large systems.

Access. It is the smallest amount of data access for rebuilding a node erasure. Note that a transmitted symbol can be a function of many accessed symbols. As a result, the rebuilding access is always no less than the repair bandwidth. If the access of a code matches the bound (2.1) we say is it *optimal access*. Small access leads to small disk I/O and also fast information retrieval.

Array Size. It is the dimension of the code array. Let the *code length* be the number of information

nodes k. In order to achieve optimal rebuilding, sometimes each node needs to be subpacketized into many symbols. The length of each column equals to the number of subpackets, denoted by l. We would like to have l small for given k so that the file size can be flexible and the code can be easy to work with.

The problem of rebuilding or regenerating information in (distributed) storage systems has attracted considerable interest in recent years. A recent survey of the repair problem can be found in [DRWS11]. If we repair the lost information *functionally*, namely to obtain a function of the file and maintain the MDS property, [WDR07, Wu09] showed existence of codes using network coding techniques. If we repair the lost information *exactly*, [SR10b, CJM10] proved that the lower bound is asymptotically achievable when the column length l goes to infinity. However, the proofs in the above work are theoretical and is based on very large finite fields. Hence, it does not provide the basis for constructing practical codes with small finite fields. In [WD09] [CDH09], this lower bound is matched for exact repair by code constructions for $k = 2, 3$, or $n - 1$. In [RSKR09, SR10a, SRKR10] codes achieving the repair bandwidth lower bound were studied where the number of systematic nodes is less than the number of parity nodes (low code rate). And [CHL11, CJ11, CHLM11, PDC11b, PDC11a, TWB11, WTB11, TWB13] studied codes with more systematic nodes than parity nodes (high code rate) and finite l, and achieved the lower bound of the repair bandwidth.

In this thesis, we mainly focus on the regime of high-rate codes. We first try to rebuild erasures in existing erasure-correcting codes, and observe the advantages and disadvantages of such codes. Then we try to construct our own codes, which match the repair bandwidth/access lower bound and are therefore optimal.

In Chapter 3, we study the rebuilding of existing MDS array codes, such as EVENODD. Such codes are not only MDS but also binary codes, namely they only require XOR operations for encoding and decoding. We show that when there are two redundancy nodes, to rebuild one erased systematic node, only $3/4$ of the information needs to be transmitted. Interestingly, in many cases, the required disk I/O is also minimized.

Actually (2.1) shows that the lower bound of rebuilding access is a fraction of $1/2$. There is a gap between the lower bound and the existing array codes. In Chapter 4 we construct *zigzag codes* and close this gap. For the case of a single systematic erasure with a 2-erasure-correcting code, the rebuilding access is a fraction of $1/2$. In general, we construct a new family of r-erasure correcting

MDS array codes that has optimal rebuilding access fraction of $\frac{1}{r}$ in the case of a single systematic erasure. Our array codes have efficient encoding and decoding algorithms (for the cases $r = 2$ and $r = 3$ they use a finite field of size 3 and 4, respectively) and an optimal update property.

However, we have not yet solved the problem entirely. For a parity node erasure, all the information needs to be accessed. Namely, constructing array codes with optimal rebuilding for an arbitrary erasure was left as an open problem. In Chapter 5, we solve this open problem and present array codes that achieve the lower bound (2.1) for rebuilding any single systematic or parity node.

We discuss other decoding problems in Chapter 6. For zigzag codes with two parities, we study how to correct one erasure, two erasures, or one column error. Notice that a column is composed of l symbols, we also discuss rebuilding an erased column in the presence of symbol errors. Moreover, we show that zigzag codes have optimal rebuilding for multiple erasures. When e nodes are erased, we show that only e/r fraction of the file needs to be accessed, for all $1 \leq e \leq r$.

If we consider the array size, zigzag codes have length $k = \log_r l$ given column length l. In Chapter 7 we try to lengthen the array so that we can accommodate more storage nodes given the capacity of each one. To relax the constraints, we look for optimal repair bandwidth codes instead of optimal access. We give code constructions such that the code length is $k = (r+1)\log_r l$.

Chapter 3

Rebuild for Existing Array Codes

3.1 Introduction

Coding techniques for storage systems have been used widely to protect data against errors or erasure for CDs, DVDs, Blu-ray Discs, and SSDs. Assume the data in a storage system is divided into packets of equal sizes. An (n,k) block code takes k information packets and encodes them into a total of n packets of the same size. Among coding schemes, maximum distance separable (MDS) codes offer maximal reliability for a given redundancy: any k packets are sufficient to retrieve all the information. Reed-Solomon codes [RS60] are the most well-known MDS codes that are used widely in storage and communication applications. Another class of MDS codes are MDS array codes, for example, EVENODD [BBBM95] and its extension [BBV96], B-code [XBBW99], X-code [XB99], RDP [CEG$^+$04], and STAR code [HX08]. In an array code, each of the packets consists of a column of elements (one or more binary bits), and the parities are computed by XORing some information bits. These codes have the advantage of low computational complexity over RS codes because the encoding and decoding only involve XOR operations.

The rebuilding problem studies the repair bandwidth. That is, the necessary amount of information transmission in order to rebuild a node erasure. In particular, works have been done on lower bound of the repair bandwidth [DGW$^+$10], asymptotic achievability [SR10b] [CJM10] and constructions of optimal codes for specific parameters, e.g., [WD09] [CDH09] [SR10b]. In this chapter we take a different route: rather than trying to construct MDS codes that are easily repairable, we try to find ways to repair existing codes and specifically focus on the families of MDS array codes. A related and independent work can be found in [XXLC10], where single-disk recovery for RDP code was studied, and the recovery method and repair bandwidth is indeed similar to our result. Besides, [XXLC10] discussed balancing disk I/O reads in the recovery. Our work discusses the

recovery of single or double disk recovery for EVENODD, X-code, STAR, and RDP code.

In practice, there is a difference between erasures of the information (also called systematic) and the parity nodes. An erasure of the former will affect the information access time since part of the raw information is missing, however erasure of the latter does not have such an effect, since the entire information is still accessible. Moreover, in most storage systems the number of parity nodes is quite small compared to the number of systematic nodes. Therefore, our study focus on rebuilding for the systematic nodes. The rebuilding of a parity node will require transmitting all the information in the systematic nodes.

In the general framework of [DGW$^+$10], an acceptable rebuilding is one that retains the MDS property and not necessarily rebuilds the original erased node, whereas, we restrict our solutions to *exact* rebuilding. Exact rebuilding has the benefit that we can retrieve the original information immediately if the code is *systematic*, i.e., the information are stored in k nodes.

Moreover, we assume that all the surviving nodes are accessible. This is reasonable if the nodes are located in a single disk array, or if all the links in the storage network is available. By accessing more nodes, we can parallelize the rebuilding process, and read and transmit less information in total. However, for some cloud storage applications, the main repair performance bottleneck is the disk I/O overhead, which is proportional to the number of nodes involved in the repair process of a failed node. Therefore, codes with low number of accessed nodes, or locally repairable codes, are studied in the literature (e.g., [GHSY12] [OD11] [SRKV13] [TPD13]).

If the whole data object stored has size $\mathcal{M}$ bits, repairing a single erasure naively would require communicating (and reading) $\mathcal{M}$ bits from surviving storage nodes. Here we show that a single failed systematic node can be rebuilt after communicating only $\frac{3}{4}\mathcal{M} + O(\mathcal{M}^{1/2})$ bits. Note that the cut-set lower bound [DGW$^+$10] scales like $\frac{1}{2}\mathcal{M} + O(\mathcal{M}^{1/2})$, so it remains open if the repair communication for EVENODD codes can be further reduced. This question will be answered positively in the next few chapters. Interestingly our repair scheme also requires significantly less disk I/O reads compared to naively reading the whole data object.

In this chapter we will review EVENODD and other related array codes, propose rebuilding algorithms for one or two erasures, where the number of redundancies are two or more.

3.2 Definitions

An $R \times n$ array code contains R rows and n columns (or packets). Each element in the array can be a single bit or a block of bits. We are going to call an element a *block*. In an (n, k) array code, k information columns, or systematic columns, are encoded into n columns. The *total amount of information* is $\mathcal{M} = Rk$ blocks.

An EVENODD code [BBBM95] is a binary MDS array code that can correct up to 2 column erasures. For a prime number $p \geq 3$, the code contains $R = p - 1$ rows and $n = p + 2$ columns, where the first $k = p$ columns are information and the last two are parity. And the information is $\mathcal{M} = (p-1)p$ blocks.

We will write an EVENODD code as:

$$\begin{array}{cccccc} a_{1,1} & a_{1,2} & \dots & a_{1,p} & b_{1,0} & b_{1,1} \\ a_{2,1} & a_{2,2} & \dots & a_{2,p} & b_{2,0} & b_{2,1} \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ a_{p-1,1} & a_{p-1,2} & \dots & a_{p-1,p} & b_{p-1,0} & b_{p-1,1} \end{array}$$

And we define an imaginary row $a_{p,j} = 0$, for all $j = 1, 2, \dots, p$, where 0 is a block of zeros. The *slope 0* or *horizontal parity* is defined as

$$b_{i,0} = \sum_{j=1}^{p} a_{i,j} \tag{3.1}$$

for $i = 1, \dots, p-1$. The addition here is bit-by-bit XOR for two blocks. A parity block of *slope* v, $-p < v < p$ and $v \neq 0$ is defined as

$$b_{i,v} = \sum_{j=1}^{p} a_{j,<i+v(1-j)>} + S_v = \sum_{j=1}^{p} a_{<i+v(1-j)>,j} + S_v \tag{3.2}$$

where $S_v = a_{p,1} + a_{p-v,2} + \cdots + a_{<p+v>,p} = \sum_{j=1}^{p} a_{<v(1-j)>,j}$ and $< x >= (x-1) \bmod p + 1$. Sometimes we omit the "$<>$" notation. When $v = 1$, we call it the *slope 1*, or *diagonal parity*. In EVENODD, parity columns are of slopes 0 and 1.

A similar code is RDP [CEG+04], where $R = p - 1$, $n = p + 1$, and $k = p - 1$, for a prime number p. The diagonal parity sums up both the corresponding information blocks and one horizontal parity block. Another related code is X-code [XB99], where the parity blocks are of

slope -1 and 1, and are placed as two additional rows, instead of two parity columns.

The code in [BBV96] extended EVENODD to more than 2 columns of parity. This code has $n = p + r$, $k = p$, and $R = p - 1$. The information columns are the same as EVENODD, but r parity columns of slopes $0, 1, \ldots, r-1$ are used. It is shown in [BBV96] that such a code is MDS when $r \leq 3$ and conditions for a code to be MDS are derived for $r \leq 8$.

STAR code [HX08] is an MDS array code with $k = p, R = p - 1, n = p + 3$, and the parity columns are of slope 0, 1, and -1.

A *parity group* $B_{i,v}$ of slope v contains a parity block $b_{i,v}$ and the information blocks in the sum in equations (3.1) (3.2), $i = 1, 2, \ldots, p-1$. S_v is considered as a single information block. If $v = 0$, it is a *horizontal parity group*, and if $v = 1$, we call it a *diagonal parity group*.

By (3.1), each horizontal parity group $B_{i,0}$ contains $a_{i,<k+1-i>} \in B_{k,1}$, for all $k = 1, 2, \ldots, p-1$. So we say $B_{i,0}$ *crosses with* $B_{k,1}$, for all $k = 1, 2, \ldots, p-1$. Conversely, each diagonal parity group $B_{i,1}$ contains $a_{k,<i+1-k>} \in B_{k,0}$, for all $k = 1, 2, \ldots, p-1$. Therefore, $B_{i,1}$ crosses with $B_{k,0}$ for all $k = 1, 2, \ldots, p-1$. The shared block of two parity groups is called the *crossing*. Generally, two parity groups $B_{i,v}$ and $B_{k,u}$ cross, for $v \neq u$, $1 \leq i, k \leq p-1$. If they cross at $a_{p,<i+v>} = 0$, we call it a *zero crossing*. A zero crossing does not really exist since the p-th row is imaginary. A zero crossing occurs if and only if

$$u, v \neq 0 \text{ and } < i + v > = < k + u > \tag{3.3}$$

Moreover, each information block belongs to only one parity group of slope v.

In this chapter, we use MDS array codes as distributed storage codes. We will give repair methods and compute the corresponding bandwidth γ.

Example 3.1 *Consider the EVENODD code with $p = 3$. Set $a_{1,3} = a_{2,3} = 0$ for all codewords, then the code will contain only 2 columns of information. The resulting code is a $(4, 2)$ MDS code and this is called* shortened *EVENODD (see Figure* 3.1*). It can be verified that if any node is erased, then sending 1 block from each of the other nodes is sufficient to recover it. And this actually matches the bound (*2.1*). Figure* 3.1 *shows how to recover the first or the fourth column. Notice that a sum block is sent in some cases. For instance, to recover the first column, the sum $b_{1,1} + b_{2,1}$ is sent from the fourth column.*

In this chapter, shortening of a code is not considered and we will focus on the recovery of systematic nodes, given that 1 or 2 systematic nodes are erased. And we send no linear combinations

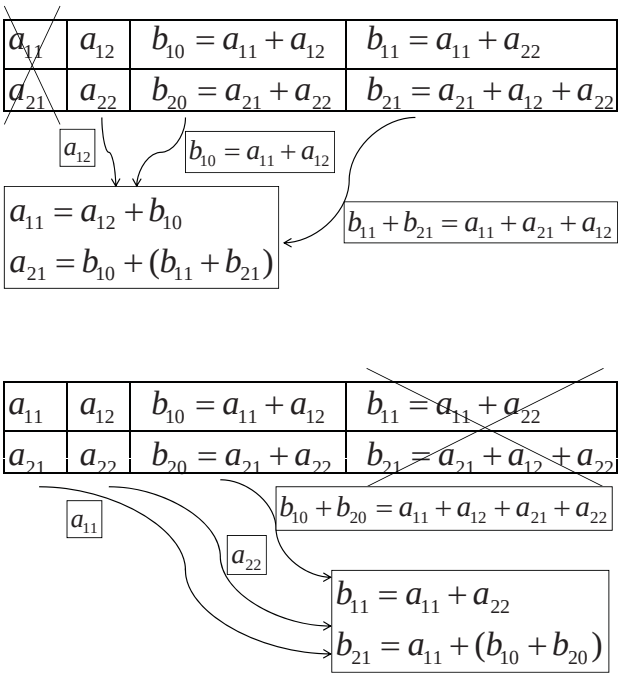

Figure 3.1: Repair of a $(4,2)$ EVENODD code if the first column (top graph) or the fourth column (bottom graph) is erased. In both cases, three blocks are transmitted.

of data except the sum $\sum_{i=1}^{p-1} b_{i,v}$ from the parity node of slope v, for all v defined in an array code. In addition, we assume that each node can transmit a different number of blocks.

3.3 Repair for Codes with Two Parity Nodes

First, let us consider the repair problem of losing one systematic node, $n-d=1$, and $n-k=2$. We will use EVENODD to explain the repair method, and the recovery will be very similar if RDP or X-code is considered.

By the symmetry of the code, we assume that the first column is missing. Each block in the first column must be recovered through either the horizontal or the diagonal parity group including this block. Suppose we use x horizontal parity groups and $p-1-x$ diagonal parity groups to recover the column, $0 \le x \le p-1$. These parity groups include all blocks of the first column exactly once.

Notice that $S_1 = \sum_{i=1}^{p-1} b_{i,0} + \sum_{i=1}^{p-1} b_{i,1}$, so we can send $\sum_{i=1}^{p-1} b_{i,0}$ from the $(p+1)$-th node, and $\sum_{i=1}^{p-1} b_{i,1}$ from the $(p+2)$-th node, and recover S_1 with 2 blocks of transmission. For the discussion below, assume S_1 is known.

For each horizontal parity group $B_{i,0}$, we send $b_{i,0}$ and $a_{i,j}$, $j = 2,3,\ldots,p$. So we need p blocks. For each diagonal parity group $B_{i,1}$, as S_1 is known, we send $b_{i,1}$ and $a_{j,<i+1-j>}$, $j =$

$1, 2, \ldots, i-1, i+1, \ldots, p-1$, which is $p-1$ blocks in total.

If two parity groups cross at one block, there is no need to send this block twice. As shown in Section 3.2, any horizontal and any diagonal parity group cross at a block, and each block can be the crossing of two groups at most once. There are $x(p-1-x)$ crossings. The total number of blocks sent is

$$\begin{aligned}\gamma &= \underbrace{xp}_{\text{horizontal}} + \underbrace{(p-1-x)(p-1)}_{\text{diagonal}} + \underbrace{2}_{S_1} - \underbrace{x(p-1-x)}_{\text{crossings}} \\ &= (p-1)p + 2 - (x+1)(p-1-x) \\ &\geq (p-1)p + 2 - (p^2-1)/4 = (3p^2-4p+9)/4\end{aligned} \tag{3.4}$$

The equality holds when $x = (p-1)/2$ or $x = (p-3)/2$, where x is an integer.

This result states that we only need to send about $3/4$ of the total amount of information. And the slopes of the n chosen parity groups do not matter as long as half are horizontal and half are diagonal. Moreover, similar repair bandwidth can be achieved using RDP or X-code. For RDP code, the repair bandwidth is

$$\frac{3(p-1)^2}{4}$$

which was also derived independently in [XXLC10]. For X-code, the repair bandwidth is at most

$$\frac{3p^2-2p+5}{4}$$

The derivation for RDP is the following. For RDP code, the first $p-1$ columns are information. The p-th column is the horizontal parity. The $(p+1)$-th column is the slope 1 diagonal parity (including the p-th column). The diagonal starting at $a_{p,1} = 0$ is not included in any diagonal parities. Suppose the first column is erased. Each horizontal or diagonal parity group will require $p-1$ blocks of transmission. Every horizontal parity group crosses with every diagonal parity group. Suppose $(p-1)/2$ horizontal parity groups and $(p-1)/2$ diagonal parity groups are transmitted. Then the total transmission is

$$\gamma = \underbrace{(p-1)(p-1)}_{p-1 \text{ parity groups}} - \underbrace{\frac{p-1}{2}\frac{p-1}{2}}_{\text{crossings}} = \frac{3(p-1)^2}{4}$$

This result is also derived independently in [XXLC10].

The derivation for X-code is as follows. For X-code, the $(p-1)$-th row is the parity of slope -1, excluding the p-th row. And the p-th row is the parity of slope 1, excluding the $(p-1)$-th row. Suppose the first column is erased. First notice that for each parity group, $p-2$ blocks need to be transmitted. To recover the parity block $a_{p-1,1}$, one has to transmit the slope -1 parity group starting at $a_{p-1,1}$. To recover the parity block $a_{p,1}$, the slope 1 parity group starting at $a_{p,1}$ must be transmitted. But it should be noted that by the construction of X-code, this slope 1 parity group essentially is the diagonal starting at $a_{p-1,1}$, except for the first element $a_{p,1}$. Zero crossings happen between two parity groups of slopes -1 and 1, starting at $a_{i,1}$ and $a_{j,1}$, if

$$< i+j >= p-2 \text{ or } < i+j >= p$$

Each slope 1 parity group has no more than 2 zero crossings with the slope -1 parity groups.

Suppose we choose arbitrarily $(p-1)/2$ slope 1 parity groups and $(p-3)/2$ slope -1 parity groups for the information blocks in the first column. Then not considering the parity group containing $a_{p,1}$, the number of slope 1 and slope -1 parity groups are both $(p-1)/2$. Excluding zero crossings, each slope 1 parity group crosses with at least

$$(p-1)/2-2=(p-5)/2$$

slope -1 parity groups. The total transmission is

$$\gamma \leq \underbrace{p(p-2)}_{p \text{ parity groups}} - \underbrace{\frac{p-1}{2}\frac{p-5}{2}}_{\text{crossings}} = \frac{3p^2-2p+5}{4}$$

Also, equation (3.4) is optimal in some conditions:

Theorem 3.2 *The transmission bandwidth in (*3.4*) is optimal to recover a systematic node for EVENODD if no linear combinations are sent except* $\sum_{i=1}^{p-1} b_{i,v}$*, for* $v=0,1$.

Proof: To recover a systematic node, say, the first node, parity blocks $b_{i,v}$, $i=1,2,\ldots,p-1$ must be sent, where v can be 0 or 1 for each i. This is because $a_{i,1}$ is only included in $b_{i,0}$ or $b_{i,1}$. Besides, given $b_{i,v}$, the whole parity group $B_{i,v}$ must be sent to recover the lost block. Therefore, our strategy of choosing x horizontal parity groups and $p-1-x$ diagonal parity groups has the most efficient transmission. Finally, since (3.4) is minimized over all possible x, it is optimal. □

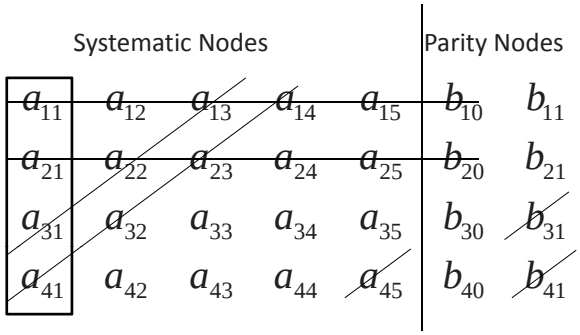

Figure 3.2: Repair of an EVENODD code with $p = 5$. The first column is erased, shown in the box. 14 blocks are transmitted, shown by the blocks on the horizontal or diagonal lines. Each line (with wrap around) is a parity group. 2 blocks in summation form, $\sum_{i=1}^{p-1} b_{i,0}, \sum_{i=1}^{p-1} b_{i,1}$ are also needed but are not shown in the graph.

The lower bound by (2.1) is

$$\frac{\mathcal{M}d}{(d-k+1)k} = \frac{\mathcal{M}(n-1)}{(n-k)k} = \frac{p(p-1)(p+1)}{2p} = \frac{p^2-1}{2}$$

where $d = n-1$, $n = p+2$, $k = p$, and $\mathcal{M} = p(p-1)$. It should be noted that (2.1) assumes that each node sends the same number of blocks, but our method does not.

Example 3.3 *Consider the EVENODD code with $p = 5$ in Figure 3.2. For $1 \le i \le 4$, the code has information blocks $a_{i,j}$, $1 \le j \le 5$, and parity blocks $b_{i,v}$, $v = 0, 1$. Suppose the first column is lost. Then by (3.4), we can choose parity groups $B_{1,0}, B_{2,0}, B_{3,1}, B_{4,1}$. The blocks sent are: $\sum_{i=1}^{p-1} b_{i,0}, \sum_{i=1}^{p-1} b_{i,1}, b_{1,0}, b_{2,0}, b_{3,1}, b_{4,1}$ from the parity nodes and $a_{1,2}, a_{1,3}, a_{1,4}, a_{1,5}, a_{2,2}, a_{2,3}, a_{2,4}, a_{2,5}, a_{4,5}, a_{3,2}$ from the systematic nodes. Altogether, we send 16 blocks, the number specified by (3.4). We can see that $a_{1,3}$ is the crossing of $B_{1,0}$ and $B_{3,1}$. Similarly, $a_{1,4}, a_{2,2}, a_{2,3}$ are crossings and are only sent once for two parity groups.*

3.4 r Parity Nodes and One Erased Node

Next we discuss the repair of array codes with r columns of parity, $r \ge 3$. And we consider the recovery in case of one missing systematic column. In this section, we are going to use the extended EVENODD code [BBV96], i.e., codes with parity columns of slope $0, 1, \ldots, r-1$. Similar results can be derived for STAR code. Suppose the first column is erased without loss of generality.

Let us first assume $r = 3$, so the parity columns have slopes $0, 1, 2$. The repair strategy is: sending parity groups $B_{3n+v,v}$ for $v = 0, 1, 2$ and $1 \le 3n+v \le p-1$. Let $A = \lfloor (p-1)/3 \rfloor$.

Notice that $0 \leq n \leq A$ and each slope has no more than $\lceil (p-1)/3 \rceil$ but no less than $\lfloor (p-1)/3 \rfloor = A$ parity groups.

Since there are three different slopes, there are crossings between slope 0 and 1, slope 1 and 2, and slope 2 and 0. For any two parity groups $B_{i,1}$ and $B_{k,2}$, $< k-i > \neq 1$, so (3.3) does not hold. Hence no zero crossing exists for the chosen parity groups. Hence, every crossing corresponds to one block of saving in transmission. However, the total number of crossings is not equal to the sum of crossings between every two parity groups with different slopes. Three parity groups with slopes 0, 1, and 2 may share a common block, which should be subtracted from the sum.

Notice that the parity group $B_{i,v}$ contains block $a_{i-vy,y+1}$. The modulo function "<>" is omitted in the subscripts. For three transmitted parity groups $B_{3n,0}, B_{3m+1,1}, B_{3l+2,2}$, if there is a common block in column $y+1$, then it is in row $3n \equiv 3m+1-y \equiv 3l+2-2y \pmod{p}$. To solve this, we get $y \equiv 3(m-n)+1 \equiv 3(l-m)+1 \pmod{p}$, or $m-n \equiv l-m \pmod{p}$. Notice $0 \leq n,m,l < p/3$, so $-p/3 < m-n, l-m < p/3$. Therefore, $m-n = l-m$ without modulo p. Thus $l-n$ must be an even number. For fixed n, either $n \leq m \leq l \leq A$, and there are no more than $(A-n)/2+1$ solutions for (m,l); or $0 \leq l < m < n$, and the number of (m,l) is no more than $n/2$. Hence, the number of (n,m,l) is no more than $\sum_{n=1}^{A}((A-n)/2+1+n/2) = A^2/2+A$.

The total number of blocks in the $p-1$ chosen parity groups is less than $p(p-1)$. There are no less than A parity groups of slope v, for all $0 \leq v \leq 2$, therefore for $0 \leq u < v \leq 2$, parity groups with slopes u and v have no less than A^2 crossings. Hence the total number of blocks sent in order to recover one column is:

$$\begin{aligned} \gamma &< \underbrace{p(p-1)}_{p-1 \text{ parity groups}} - \underbrace{\binom{3}{2}A^2}_{\text{crossings}} + \underbrace{\frac{A^2+2A}{2}}_{\text{common}} + \underbrace{3}_{\sum_{i=1}^{p-1} b_{i,v}} \\ &< \frac{13}{18}p^2 + \frac{17}{9}p - \frac{47}{18} \end{aligned} \tag{3.5}$$

where $(p-4)/3 < A \leq (p-1)/3$. The above estimation is an upper bound because there may be better ways to assign the slopes of each parity group. When $r = 3$, we need to send no more than about $13\mathcal{M}/18$ blocks.

By abuse of notation, we write $B_{m,v} = \{a_{<m+v(1-j)>,j} : j = 2,\ldots,p\}$ as the set of blocks (including the imaginary p-th row) in the parity group except S_v and $a_{m,1}$. Let $M_v \subseteq \{1,2,\ldots,q\}$, $0 \leq v \leq r-1$, be disjoint sets such that $\cup_{v=0}^{r-1} M_v = \{1,2,\ldots,q-1\}$. Let $B_{M_v,v} = \cup_{m \in M_v} B_{m,v}$. For given M_v, define a function f as $f(v_1,v_2,\ldots,v_k) = |\{m_1 \in M_{v_1},\ldots,m_k \in M_{v_k} : (m_2 -$

$m_1)/(v_2 - v_1) \equiv (m_3 - m_2)/(v_3 - v_2) \equiv \ldots (m_k - m_{k-1})/(v_k - v_{k-1}) \bmod p\}|$, for $k \geq 3$, and $0 \leq v_1 < v_2 < \cdots < v_k \leq r-1$. Then we have the following theorem:

Theorem 3.4 *For the extended EVENODD with $r \geq 3$, the repair bandwidth for one erased systematic node is*

$$\begin{aligned} \gamma \;<\; & p(p-1) + p + r - \sum_{0 \leq v_1 < v_2 \leq r-1} |M_{v_1}||M_{v_2}| \\ & + \sum_{0 \leq v_1 < v_2 < v_3 \leq r-1} f(v_1, v_2, v_3) - \cdots + (-1)^{r-1} f(0, 1, \ldots, r-1) \end{aligned} \tag{3.6}$$

Proof: Suppose the first column is missing and we transmit the parity groups $B_{m,v}$, $m \in M_v$ for $v = 0, 1, \ldots, r-1$. Since the union of M_v covers $\{1, 2, \ldots, q-1\}$, all the blocks in the first column can be recovered. The repair bandwidth is the cardinality of the union of $B_{M_v,v}$ plus the number of zero crossings and the summation blocks $\sum_{i=1}^{p-1} b_{i,v}$. The number of zero crossings is no more than the size of the imaginary row, p. The number of the summation blocks is r.

By inclusion–exclusion principle, the cardinality of the union of $B_{M_v,v}$ is

$$\begin{aligned} & \sum_{0 \leq v \leq r-1} |B_{M_v,v}| - \sum_{0 \leq v_1 < v_2 \leq r-1} |B_{M_{v_1},v_1} \cap B_{M_{v_2},v_2}| \\ & + \sum_{0 \leq v_1 < v_2 < v_3 \leq r-1} |B_{M_{v_1},v_1} \cap B_{M_{v_2},v_2} \cap B_{M_{v_3},v_3}| - \cdots + (-1)^{r-1} |B_{M_0,0} \cap B_{M_1,1} \ldots B_{M_{r-1},r-1}| \end{aligned}$$

Every $|B_{m,v}| \leq p$, so $\sum_{0 \leq v \leq r-1} |B_{M_v,v}| \leq p(p-1)$. Every two parity groups B_{m_1,v_1}, B_{m_2,v_2} cross at a block. Hence $|B_{M_{v_1},v_1} \cap B_{M_{v_2},v_2}| = |M_{v_1}||M_{v_2}|$. Since $B_{m,v}$ contains $a_{<m+v(1-j)>,j}$, $j = 2, \ldots, p$, the intersection of more than two parity groups $B_{m_1,v_1}, \ldots, B_{m_k,v_k}$ is equivalent to the solutions of

$$m_1 - v_1 y \equiv m_2 - v_2 y \equiv \cdots \equiv m_k - v_k y \bmod p$$

where $y + 1$ is the column index of the intersection. Or,

$$y \equiv \frac{m_2 - m_1}{v_2 - v_1} \equiv \cdots \equiv \frac{m_k - m_{k-1}}{v_k - v_{k-1}} \bmod p$$

Therefore,

$$|B_{M_{v_1},v_1} \cap B_{M_{v_2},v_2} \cap \ldots B_{M_{v_k},v_k}| = f(v_1, v_2, \ldots, v_k)$$

And (3.6) follows. □

We can see that (3.5) is a special case of (3.6), with $M_v = \{3n + v : 1 \leq 3n + v \leq p-1\}$,

for $v = 0, 1, 2$. For $r = 4, 5$, we can derive similar bounds by defining M_v.

Choose

$$M_v = \{rn + v : 1 \leq rn + v \leq p - 1\} \tag{3.7}$$

for $v = 0, 1, \ldots, r-1$. Let $A = \lfloor (p-1)/r \rfloor$. And for $0 \leq v_1 < v_2 < v_3 \leq r-1$, $f(v_1, v_2, v_3)$ becomes the number of (n_1, n_2, n_3), $1 \leq rn_i + v_i \leq p - 1$, such that

$$(n_2 - n_1)(v_3 - v_2) \equiv (n_3 - n_2)(v_2 - v_1) \bmod p$$

Since $-p/r < n_2 - n_1, n_3 - n_2 < p/r$, and $(v_3 - v_2) + (v_2 - v_1) < r$, the above equation becomes

$$(n_2 - n_1)(v_3 - v_2) = (n_3 - n_2)(v_2 - v_1)$$

without modulo p. Therefore,

$$\begin{aligned} & n_3 - n_1 = (n_3 - n_2) + (n_2 - n_1) \\ = \;& c \cdot \mathrm{lcm}(v_3 - v_2, v_2 - v_1) \left(\frac{1}{v_3 - v_2} + \frac{1}{v_2 - v_1} \right) \\ = \;& c \frac{v_3 - v_1}{\gcd(v_3 - v_2, v_2 - v_1)} \end{aligned}$$

where c is an integer constant, lcm is the least common multiplier and gcd is the greatest common divisor. And for fixed n_1, the number of solutions for (n_2, n_3) is no more than $1 + (A - n_1)\gcd(v_3 - v_2, v_2 - v_1)/(v_3 - v_1)$, when $n_1 \leq n_2 \leq n_3 \leq A$; and no more than $n_1 \gcd(v_3 - v_2, v_2 - v_1)/(v_3 - v_1)$, when $0 \leq n_3 < n_2 < n_1$. The number of (n_1, n_2, n_3) is

$$\begin{aligned} f(v_1, v_2, v_3) \;&<\; \sum_{n_1} 1 + (A - n_1 + n_1) \frac{\gcd(v_3 - v_2, v_2 - v_1)}{v_3 - v_1} \\ &= \; A \left(1 + A \frac{\gcd(v_3 - v_2, v_2 - v_1)}{v_3 - v_1} \right) \end{aligned}$$

Similarly, for four parity groups,

$$f(v_1, v_2, v_3, v_4) > A \left(1 + (A + 2) \frac{\gcd(v_4 - v_3, v_3 - v_2, v_2 - v_1)}{v_4 - v_1} \right)$$

For five parity groups,

$$f(v_1, v_2, v_3, v_4, v_5) < A + A^2 \frac{\gcd(v_5 - v_4, v_4 - v_3, v_3 - v_2, v_2 - v_1)}{v_5 - v_1}$$

When $r = 4$, equation (3.6) becomes

$$\gamma < p(p-1) + p + 4 - \sum_{0 \le v_1 < v_2 \le 3} |M_{v_1}||M_{v_2}| + \sum_{0 \le v_1 < v_2 < v_3 \le 3} f(v_1, v_2, v_3) - f(0,1,2,3)$$

By the previous equations,

$$f(0,1,2), f(1,2,3) < A(1 + A/2)$$

$$f(0,1,3), f(0,2,3) < A(1 + A/3)$$

$$f(0,1,2,3) > A(1 + (A+2)/3$$

And the repair bandwidth is

$$\gamma \approx p^2 - \binom{4}{2}(\frac{p}{4})^2 + (2 \times \frac{1}{2} + 2 \times \frac{1}{3})(\frac{p}{4})^2 - \frac{1}{3}(\frac{p}{4})^2 = \frac{7}{24}p^2$$

where the terms of lower orders are omitted.

When $r = 5$, we can use (3.6) again and get

$$\gamma \approx p^2 + (-\binom{5}{2} + \frac{4}{2} + \frac{4}{3} + \frac{2}{4} - \frac{2}{3} - \frac{3}{4} + \frac{1}{4})(\frac{p}{5})^2 = \frac{53}{75}p^2$$

where the terms of lower orders are omitted.

It should be noted that the number of common blocks affects the bandwidth a lot. If we consider only the first 4 terms in (3.6), any assignment of M_v with equal sizes will result in a lower bound of $\gamma > (r+1)p^2/(2r) \approx p^2/2$, when r is large. But due to the common blocks, the true γ values for $r = 4, 5$ using (3.7) has only slight improvement compared to the case of $r = 3$.

The lower bound (2.1) is $\frac{\mathcal{M}d}{k(d-k+1)} = \frac{p(p-1)(p+r-1)}{pr} \approx \frac{p(p+r-1)}{r}$. When $r = 3$, this bound is about $p^2/3$.

3.5 Three Parity Nodes and Two Erased Nodes

Up to now, we have considered the recovery problem given that one column is erased. Next, let us assume that two information columns are erased and we need to recover them successively. So we first recover one of the erased nodes, and then the other one. The first recovery is discussed in this section, and the second recovery was already discussed in the previous sections. Suppose we have 3 columns of parity with slopes -1, 0, and 1, which is in fact the STAR code in [HX08]. Again, the arguments can be applied to extended EVENODD in a similar way. Without loss of generality, assume the first and $(x+1)$-th columns are missing, $1 \le x \le p-1$.

Let $B_{i,0}$,$B_{i,1}$, and $B_{i,-1}$ be i-th parity group of slope 0, 1, and -1, respectively, $i = 1, 2, \ldots, p-1$. The following are $3(p-1)/2$ parity groups that repair the first column: $B_{0,-1}, B_{x,0}, B_{2x,1}, B_{2x,-1}$, $B_{3x,0}, B_{4x,1}, \ldots, B_{(p-3)x,-1}, B_{(p-2)x,0}, B_{(p-1)x,1}$. For each parity block above, the corresponding recovered blocks are: $a_{x,1+x}, a_{x,1}, a_{2x,1}, a_{3x,1+x}, a_{3x,1}, a_{4x,1}, \ldots, a_{(p-2)x,1+x}, a_{(p-2)x,1}, a_{(p-1)x,1}$. An example of $p = 5, x = 1$ is shown in Figure 3.3.

Rearrange the columns in the following order: columns $1, 1+x, 1+2x, \ldots, 1+(p-1)x$ (every index is computed modulo p). We can see that the chosen parity groups $B_{jx,0}$, $j = x, 3x, \ldots,$ $(p-2)x$ contain the blocks in rows $Z = \{x, 3x, \ldots, (p-2)x\}$. $B_{jx,1}$ contains blocks $a_{jx,1}$, $a_{(j-1)x,1+x}, \ldots, a_{(j-p+1)x,1+(p-1)x}$, for $j = 2, 4, \ldots, p-1$. And similarly $B_{jx,-1}$ contains blocks $a_{jx,1}, a_{(j+1)x,1+x}, \ldots, a_{(j+p-1)x,1+(p-1)x}$, for $j = 0, 2, \ldots, p-3$.

Now notice that the blocks included in the above parity groups have the $(1+x)$-th column as the vertical symmetry axis. That is, the row indices of the blocks needed in columns 1 and $1+2x$ are the same; those of columns $1+(p-1)x$ and $1+3x$ are the same; ...; those of columns $1+(p+3)x/2$ and $1+(p+1)x/2$ are the same. For example, the second column in Figure 3.3 is the symmetry axis. Thus, we only need to consider columns $1+2x, 1+3x, \ldots, 1+(p+1)x/2$.

For columns $1+ix$, where i is even and $2 \le i \le (p+1)/2$, parity groups $\{B_{2x,1}, B_{4x,1}, \ldots,$ $B_{(p-1)x,1}\}$ include the blocks in rows $X = \{2x, 4x, \ldots, (p-1-i)x\}$. And parity groups $\{B_{0,-1},$ $B_{2x,-1}, \ldots, B_{(p-3)x,-1}\}$ include the blocks in rows $Y = \{ix, (i+2)x, \ldots, (p-1)x\}$. Since $2 \le$ $i \le (p+1)/2$, we have $i \le (p-1-i)+2$, and $X \cup Y = \{2x, 4x, \ldots, (p-1)x\}$. Hence $X \cup Y \cup Z = \{1, 2, \ldots, p-1\}$. Thus every block in Column $1+ix$ needs to be sent, for even i.

Similarly, for columns $1+ix$, where i is odd and $3 \le i \le (p+1)/2$, parity groups $\{B_{2x,1},$ $B_{4x,1}, \ldots, B_{(p-1)x,1}\}$ include the blocks in rows $X = \{(p-i+2)x, (p-i+4)x, \ldots, (p-1)x\}$. Parity groups $\{B_{0,-1}, B_{2x,-1}, \ldots, B_{(p-3)x,-1}\}$ include the blocks in rows $Y = \{2x, 4x, \ldots, (i-$

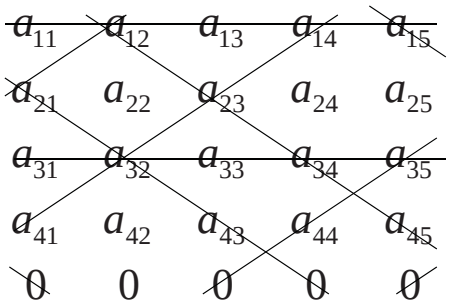

Figure 3.3: The recovery strategy for the first column in STAR code when the first and second columns are missing. $p = 5$, $x = 1$.

$3)x\}$. Since $2 \leq i \leq (p+1)/2$, we have $i-3 < p-i+2$, and $X \cup Y = \{2x, 4x, \ldots, (i-3)x, (p-i+2)x, (p-i+4)x, \ldots, (p-1)x\}$. Therefore, the rows not included in X or Y or Z are $W = \{(i-1)x, (i+1)x, \ldots, (p-i)x\}$ and $|W| = (p+3)/2 - i$. The total saving in block transmissions for all the columns is:

$$2 \sum_{i \text{ odd},\, 3 \leq i \leq (p+1)/2} \left(\frac{p+3}{2} - i\right) = \begin{cases} \frac{(p-1)^2}{8}, & \frac{p+1}{2} \text{ odd} \\ \frac{(p+1)(p-3)}{8}, & \frac{p+1}{2} \text{ even} \end{cases}$$

The above argument can be summarized in the following theorem.

Theorem 3.5 *When two systematic nodes are erased in a STAR code, there exist a strategy that transmit about* $7/8$ *of all the information blocks, and about* $1/2$ *of all the parity blocks so as to recover one node.*

The repair bandwidth γ in the above theorem is about $7p^2/8$. Comparing it to the lower bound (2.1), $\frac{\mathcal{M}d}{k(d-k+1)} = \frac{p(p-1)(p+1)}{2p} \approx \frac{p^2}{2}$, we see a gap of $\frac{3p^2}{8}$ in total transmission.

3.6 Conclusions

We presented an efficient way to repair one lost node in EVENODD codes and two lost nodes in STAR codes. Our achievable schemes outperform the naive method of rebuilding by reconstructing all the data. For EVENODD codes, a bandwidth of roughly $3\mathcal{M}/4$ is sufficient to repair an erased systematic node. Moreover, if no linear combinations of bits are transmitted, the proposed repair method has optimal repair-bandwidth with the sole exception of the sum of the parity nodes. Since array codes only operate on binary symbols, and our repair method involves no linear combination of content within a node except in the parity nodes, the proposed construction is computationally

simple and also requires smaller disk I/O to read data during repairs.

There are several open problems on using array codes for distributed storage. Although our scheme does not achieve the information-theoretic cut-set bound, it is not clear if that bound is achievable for fixed code structures or limited field sizes. If we allow linear combinations of bits within each node, the optimal repair remains unknown. Our simulations indicate that shortening of EVENODD (using less than p columns of information) further reduces the repair bandwidth but proper shortening rules and repair methods need to be developed. Repairing other families of array codes or Reed-Solomon codes would also be of substantial practical interest.

Chapter 4

Zigzag Code

4.1 Introduction

With r redundancy symbols, an MDS code is able to reconstruct the original information if no more than r symbols are erased. An array code is a two-dimensional array, where each column corresponds to a symbol in the code and is stored in a disk in the RAID scheme. We are going to refer to a disk/symbol as a node or a column interchangeably, and an entry in the array as an element.

In this chapter, we will study the rebuilding access instead of bandwidth. Suppose that some nodes are erased in a systematic MDS array code, we will rebuild them by accessing (reading) some information in the surviving nodes, all of which are assumed to be accessible. The fraction of the accessed information in the surviving nodes is called the *rebuilding ratio*, or simply *ratio*. If r nodes are erased, then the rebuilding ratio is 1 since we need to read all the remaining information. Is it possible to lower this ratio for less than r erasures? Apparently, it is possible. The ratio of rebuilding a single systematic node was shown to be $\frac{3}{4} + o(1)$ for EVENODD as shown in the previous chapter. Figure 4.1 shows an example of our new MDS code with 2 information nodes and 2 redundancy nodes, where every node has 2 elements, and operations are over the finite field of size 3. Consider the rebuilding of the first information node. It requires access to 3 elements out of 6 (a rebuilding ratio of $\frac{1}{2}$), because $a = (a+b) - b$ and $c = (c+b) - b$.

It should be noted that the rebuilding ratio counts the amount of information accessed from the system. Therefore, if we can minimize the rebuilding ratio, then we also achieve optimal disk I/O, which is an important measurement in storage.

In [DGW$^+$10], the nodes are assumed to be distributed and fully connected in a network, and the concept of *repair bandwidth* is defined as the minimum amount of data that needs to be transmitted

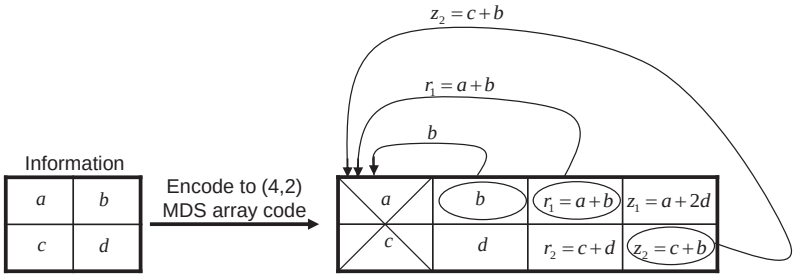

Figure 4.1: Rebuilding of a $(4,2)$ MDS array code over $\mathbb{F}_3$. Assume the first node (column) is erased.

over the network in order to rebuild the erased nodes. In contrast to our concept of *rebuilding ratio* a transmitted element of data can be a function of a number of elements that are accessed in the same node. In addition, we restrict ourselves to *exact* rebuilding. It is clear that our framework is a special case of the general framework, hence, the repair bandwidth is a lower bound on the rebuilding ratio. Let n be the total number of nodes and k be the number of systematic nodes. Suppose a file of size $\mathcal{M}$ is stored in an (n,k) MDS code, where each node stores an information of size $\mathcal{M}/k$. The number of redundancy/parity nodes is $r = n-k$, and in the rebuilding process all the surviving nodes are assumed to be accessible. A lower bound on the repair bandwidth for an (n,k) MDS code was derived in [DGW$^+$10]:

$$\frac{\mathcal{M}}{k} \cdot \frac{n-1}{n-k}.$$

It can be verified that Figure 4.1 matches this lower bound. Note that the above formula represents the amount of information, it should be normalized to reach the ratio. The normalized bandwidth compared to the size of the remaining array $\frac{\mathcal{M}(n-1)}{k}$ is

$$\text{ratio} = \frac{1}{n-k} = \frac{1}{r}. \tag{4.1}$$

A number of researchers addressed the construction of optimal repair-bandwidth codes [Wu09, WD09, RSKR09, SR10a, SRKR10, SR10b, CJM10, WDR07, RSK11], however they all have low code rate, i.e., $k/n < 1/2$. Moreover, related work on constructing codes with optimal rebuilding appeared independently in [CHL11, PDC11b]. Their constructions are similar to this work, but use larger finite-field size.

Our main goal in this chapter is to design (n,k) MDS array codes with optimal rebuilding ratio, for arbitrary number of parities. We first consider the case of 2 parities. We assume that the code is systematic. In addition, we consider codes over some finite field $\mathbb{F}_q$ with an *optimal update*

	0	1	2	R	Z
0	♣	♠	♡		♣
1	♡	♢	♣		♡
2	♠	♣	♢		♠
3	♢	♡	♠		♢

Figure 4.2: Permutations for zigzag sets in a $(5,3)$ code with 4 rows. Columns 0, 1, and 2 are systematic nodes and columns R, and Z are parity nodes. Each element in column R is a linear combination of the systematic elements in the same row. Each element in column Z is a linear combination of the systematic elements with the same symbol. The shaded elements are accessed to rebuild column 1.

property, namely, when an information symbol which is an element from the field is rewritten, only the element itself and one element from each parity node needs an update. In total $r+1$ elements are updated. For an MDS code, this achieves the minimum reading/writing during writing of information. Hence, in the case of a code with 2 parities only 3 elements are updated. Under such assumptions, we will prove that every parity element is a linear combination of exactly one information element from each systematic column. We call this set of information elements a *parity set*. Moreover, the parity sets of a parity node form a partition of the information array.

For example, in Figure 4.1 the first parity node corresponds to parity sets $\{a,b\},\{c,d\}$, which are elements in rows. We say this node is the *row parity* and each row of information forms a *row set*. The second parity node corresponds to parity sets $\{a,d\},\{c,b\}$, which are elements in zigzag lines. We say that it is the *zigzag parity* and the parity set is called a *zigzag set*. For another example, Figure 4.2 shows a code with 3 systematic nodes and 2 parity nodes. Row parity R is associated with row sets. Zigzag parity Z is associated with sets of information elements with the same symbol. For instance, the first element in column R is a linear combination of the elements in the first row and in columns $0,1$, and 2. And the ♣ in column Z is a linear combination of all the ♣ elements in columns $0,1$, and 2. We can see that each systematic column corresponds to a permutation of the four symbols. For instance, if read from top to bottom, column 0 corresponds to the permutation $[\clubsuit,\heartsuit,\spadesuit,\diamondsuit]$. In general, we will show that each parity relates to a set of a permutations of the systematic columns. Without loss of generality, we assume that the first parity node corresponds to identity permutations, namely, it is linear combination of rows.

It should be noted that in contrast to existing MDS array codes such as EVENODD and X-code, the parity sets in our codes are not limited to elements that correspond to straight lines in the array, but can also include elements that correspond to zigzag lines. We will demonstrate that this property is essential for achieving an optimal rebuilding ratio.

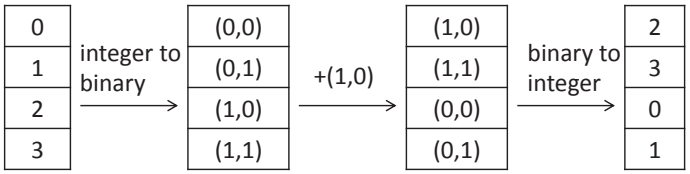

Figure 4.3: Generate the permutation by the binary vector $(1,0)$. Assume $m = 2$.

If a single systematic node is erased, we will rebuild each element in the erased node either by its corresponding row parity or zigzag parity, referred to as *rebuild by row (or by zigzag)*. In particular, we access the row (zigzag) parity element, and all the elements in this row (zigzag) set, except the erased element. For example, consider Figure 4.2, suppose that the column labeled 1 is erased, then one can access the 8 shaded elements and rebuild its first two elements by rows, and the rest by zigzags. Namely, only half of the remaining elements are accessed. It can be verified that for the code in Figure 4.2, all the three systematic columns can be rebuilt by accessing half of the remaining elements. Thus the rebuilding ratio is $1/2$, which is the lower bound expressed in (4.1).

The key idea in our construction is that for each erased node, the accessed row sets and the zigzag sets have a large intersection-resulting in a small number of accesses. Therefore it is crucial to find the permutations satisfying the above requirements. In this chapter, we will present an optimal solution to this question by constructing permutations that are derived from binary vectors. This construction provides an optimal rebuilding ratio of $1/2$ for any erasure of a systematic node. To generate the permutation over a set of integers from a binary vector, we simply add to each integer the vector and use the sum as the image of this integer. Here each integer is expressed as its binary expansion. For example, Figure 4.3 illustrates how to generate the permutation on integers $\{0,1,2,3\}$ from the binary vector $v = (1,0)$. We first express each integer in binary: $(0,0),(0,1),(1,0),(1,1)$. Then add (mod 2) the vector $v = (1,0)$ to each integer, and get $(1,0),(1,1),(0,0),(0,1)$. At last change each binary expansion back to integer and define it as the image of the permutation: $2,3,0,1$. Hence, $0,1,2,3$ are mapped to $2,3,0,1$ in this permutation, respectively. This simple technique for generating permutations is the key in our construction. We can generalize our construction for arbitrary r (number of parity nodes) by generating permutations using r-ary vectors. Our constructions are optimal in the sense that we can construct codes with r parities and rebuilding ratio of $1/r$.

So far we focused on the optimal rebuilding ratio, however, a code with two parity nodes should be able to correct two erasures. Namely, it needs to be an MDS code. We will prove that for a large

enough field size the code can be made MDS. In addition, a key result we prove is that for a code with two parity nodes, the field size is 3, and this field size is optimal. Moreover, the field size is 4 in the case of three parity nodes.

In addition, our codes have an optimal *array size* in the sense that for a given number of rows, we have the maximum number of columns among all systematic codes with optimal ratio and update. However, the length of the array is exponential in the width. We study different techniques for making the array wider, and the ratio will be asymptotically optimal when the number of rows increases. We mainly consider the case of two parities. One approach is to directly construct a larger number of permutations from binary vectors, and another is to use the same set of permutations multiple times.

In summary, the main contribution of this chapter is the first explicit construction of systematic (n,k) MDS array codes for any constant $r = n - k$, which achieves optimal rebuilding ratio of $\frac{1}{r}$. Our codes have a number of useful properties:

- They are systematic codes, hence it is easy to retrieve information.
- They have high code rate k/n, which is commonly required in storage systems.
- They have optimal update given a finite filed $\mathbb{F}_q$, namely, when an information element is updated, only $r+1$ elements in the array need update.
- The rebuilding of a failed node requires no computation in each of the surviving nodes, and thus achieves optimal disk I/O.
- The encoding and decoding of the codes can be easily implemented for $r = 2,3$, since the codes use small finite fields of size 3 and 4, respectively.
- They have optimal array size (maximum number of columns) among all systematic, optimal-update, and optimal-ratio codes. Moreover, we also have asymptotically optimal codes that have better array size.
- They achieve optimal rebuilding ratio of $\frac{1}{r}$ when a single systematic erasure occurs.

In this chapter, we will start with $(k+2,k)$ optimal rebuilding codes, and introduce its variations, and then generalize it to arbitrary number of parity columns.

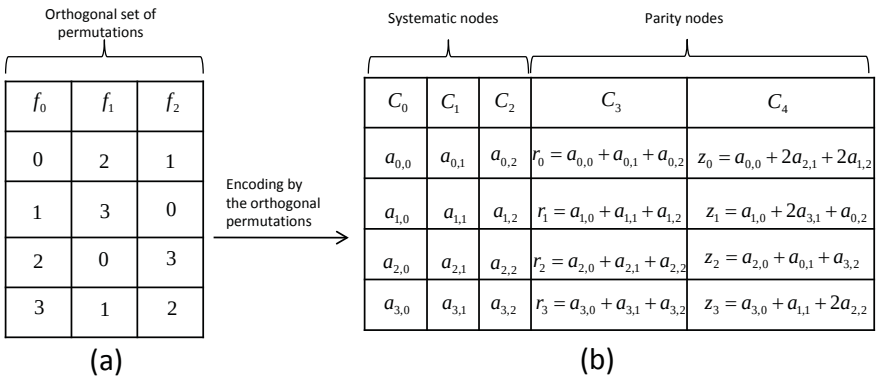

Figure 4.4: (a) The set of orthogonal permutations as in Theorem 4.3 with sets $X_0 = \{0,3\}, X_1 = \{0,1\}, X_2 = \{0,2\}$. (b) A $(5,3)$ MDS array code generated by the orthogonal permutations. The first parity column C_3 is the row sum and the second parity column C_4 is generated by the zigzags. For example, zigzag z_0 contains the elements $a_{i,j}$ that satisfy $f_j(i) = 0$.

4.2 $(k+2,k)$ MDS Array Code Constructions

Notations: In the rest of the chapter, we are going to use $[i,j]$ to denote $\{i,i+1,\ldots,j\}$ and $[i]$ to denote $\{1,2,\ldots,i\}$, for integers $i \leq j$. And denote the complement of a subset $X \subseteq M$ as $\overline{X} = M \backslash X$. For a matrix A, A^T denotes the transpose of A. For a binary vector $v = (v_1,...,v_n)$ we denote by $\overline{v} = (v_1 + 1 \mod 2, ..., v_n + 1 \mod 2)$ its complement vector. The standard vector basis of dimension m will be denoted as $\{e_i\}_{i=1}^m$ and the zero vector will be denoted as e_0. For two binary vectors $v = (v_1,\ldots,v_m), u = (u_1,\ldots,u_m)$, the inner product is $v \cdot u = \sum_{i=1}^m v_i u_i \mod 2$. For two permutations f, g, denote their composition by fg.

In this section we give the construction of MDS array code with two parities and optimal rebuilding ratio $1/2$ for one erasure, which uses a finite field of optimal size 3.

4.2.1 Constructions

Let us define an MDS array code with 2 parities. Let $A = (a_{i,j})$ be an information array of size $p \times k$ over a finite field $\mathbb{F}$, where $i \in [0, p-1], j \in [0, k-1]$. We add to the array two parity columns and obtain an $(n = k+2, k)$ MDS code of array size $p \times n$. Let the two parity columns be the *row parity* $C_k = (r_0, r_1, ..., r_{p-1})^T$, and the *zigzag parity* $C_{k+1} = (z_0, z_1 ..., z_{p-1})^T$. Let $\{f_0, f_1, \ldots, f_{k-1}\}$ be *zigzag permutations* on $[0, p-1]$ associated with the systematic columns $\{0,1,\ldots,k-1\}$. For any $l \in [0, p-1]$, define the *row set* as the subset of information elements in the same row: $R_l = \{a_{l,0}, a_{l,1}, \ldots, a_{l,k-1}\}$. The *zigzag set* is defined as elements in a zigzag

line: $Z_l = \{a_{i,j} | f_j(i) = l\}$. Then define the row parity element as $r_l = \sum_{a \in R_l} \alpha_a a$ and the zigzag parity element as $z_l = \sum_{a \in Z_l} \beta_a a$, for some sets of coefficients $\{\alpha_a\}, \{\beta_a\} \subseteq \mathbb{F}$. We can see that each parity element contains exactly one element from each systematic column, and we will show in Section 4.3 that this is equivalent to optimal update.

For example, in Figure 4.4 (a), we show three permutations on $[0,3]$. Therefore we have the 0th zigzag set $Z_0 = \{a_{0,0}, a_{2,1}, a_{1,2}\}$. The 0th row set is by default $R_0 = \{a_{0,0}, a_{0,1}, a_{0,2}\}$. And in (b) we show the corresponding code. Columns C_0, C_1, C_2 are systematic columns. The row parity C_3 sums up elements in a row, and each element in the zigzag parity C_4 is a linear combination of the elements in some zigzag set. For instance, r_0 (or z_0) is a linear combination of elements in R_0 (or Z_0, respectively). Actually, this example is the code in Figure 4.2 with more details.

The *rebuilding ratio* is the average fraction of accessed elements in the surviving systematic and parity nodes while rebuilding one systematic node. A more specific definition will be given in the next section. In order to rebuild a systematic node, each erased element can be computed either by using its row set or by zigzag set. During the rebuilding process, an element is said to be rebuilt by row (zigzag), if we use the linear equation of its row (zigzag) set in order to compute its value. Solving this equation is done simply by accessing and reading in the surviving columns the values of the rest of the intermediates.

From the example in Figure 4.2, we know that in order to get low rebuilding ratio, we need to find zigzag sets $\{Z_l\}$ (and hence permutations $\{f_i\}$) such that the row and zigzag sets used in the rebuilding intersect as much as possible. Moreover, it is clear that the choice of the coefficients is crucial if we want to ensure the MDS property. Noticing that all elements and all coefficients are from some finite field, we would like to choose the coefficients such that the finite-field size is as small as possible. So our construction of the code includes two steps:

1. Find zigzag permutations to minimize the ratio.

2. Assign the coefficients such that the code is MDS.

Next we generate zigzag permutations using binary vectors. We assume that the array has $p = 2^m$ rows.

In this section all the calculations for the indices are done over $\mathbb{F}_2$. By abuse of notation we use $x \in [0, 2^m - 1]$ both to represent the integer and its binary representation. It will be clear from the context which meaning is in use.

Let $v \in \mathbb{F}_2^m$ be a binary vector of length m. We define the permutation $f_v : [0, 2^m - 1] \to [0, 2^m - 1]$ by $f_v(x) = x + v$, where x is represented in its binary representation. For example, when $m = 2, v = (1, 0), x = 3$,

$$f_{(1,0)}(3) = 3 + (1, 0) = (1, 1) + (1, 0) = (0, 1) = 1.$$

In other words, in order to get a permutation from v, we first write all integers in $[0, 2^m - 1]$ in binary expansion, then add vector v, and at last convert binary vectors back to integers. This procedure is illustrated in Figure 4.3. Thus we can see that the permutation f_v in vector notation is $[2, 3, 0, 1]$. One can check that this is actually a permutation for any binary vector v. Next we present the code construction.

Construction 4.1 *Let A be the information array of size $2^m \times k$. Let $T = \{v_0, v_1, \ldots, v_{k-1}\} \subseteq \mathbb{F}_2^m$ be a set of vectors of size k. For $v \in T$, we define the permutation $f_v : [0, 2^m - 1] \to [0, 2^m - 1]$ by $f_v(x) = x + v$. Construct the two parities as row and zigzag parities.*

For example, in Figure 4.4 (a), the three permutations are generated by vectors $v_0 = (0, 0), v_1 = (1, 0), v_2 = (0, 1)$. In Figure 4.4 (b), the code is constructed with the row and the zigzag parities.

4.2.2 Rebuilding Ratio

Let us present the **rebuilding algorithm:** We define for a nonzero vector v, $X_v = \{x \in [0, 2^m - 1] : x \cdot v = 0\}$ as the set of integers whose binary representation is orthogonal to v. For example, $X_{(1,0)} = \{0, 1\}$. If v is the zero vector we define $X_v = \{x \in \mathbb{F}_2^m : x \cdot (1, 1, ..., 1) = 0\}$. For ease of notation, denote the permutation f_{v_j} as f_j and the set X_{v_j} as X_j. Assume column j is erased, and define $S_r = \{a_{i,j} : i \in X_j\}$ and $S_z = \{a_{i,j} : i \notin X_j\}$. Rebuild the elements in S_r by rows and the elements in S_z by zigzags.

Example 4.2 *Consider the code in Figure* 4.4*. Suppose node 1 (column C_1) is erased. Since $X_1 = X_{v_1} = X_{(1,0)} = \{0, 1\}$, we will rebuild $a_{0,1}, a_{1,1} \in S_r$ by row parity elements r_0, r_1, respectively. And rebuild $a_{2,1}, a_{3,1} \in S_z$ by zigzag parity elements z_0, z_1, respectively. In particular, we access*

the elements $a_{0,0}, a_{0,2}, a_{1,0}, a_{1,2}$*, and the following four parity elements*

$$r_0 = a_{0,0} + a_{0,1} + a_{0,2}$$
$$r_1 = a_{1,0} + a_{1,1} + a_{1,2}$$
$$z_{f_1(2)} = z_0 = a_{0,0} + 2a_{2,1} + 2a_{1,2}$$
$$z_{f_1(3)} = z_1 = a_{1,0} + 2a_{3,1} + a_{0,2}.$$

Here $f_1(2) = f_{v_1}(2) = f_{(1,0)}(2) = 0$ *and* $z_{f_1(2)} = z_0$*. Similarly,* $f_1(3) = 1$ *and* $z_{f_1(3)} = z_1$*. Note that each of the surviving node accesses exactly* $\frac{1}{2}$ *of its elements. Similarly, if node 0 is erased, we have* $X_0 = \{0, 3\}$ *so we rebuild* $a_{0,0}, a_{3,0}$ *by row and* $a_{1,0}, a_{2,0}$ *by zigzag. Since* $X_2 = \{0, 2\}$*, we rebuild* $a_{0,2}, a_{2,2}$ *by row and* $a_{1,2}, a_{3,2}$ *by zigzag in node 2. Rebuilding a parity node is easily done by accessing all the information elements.*

Theorem 4.3 *Construct permutations* $f_0, ..., f_m$ *and sets* $X_0, ..., X_m$ *by the standard basis and the zero vector* $\{e_i\}_{i=0}^m$ *as in Construction* 4.1*. Then the corresponding* $(m+3, m+1)$ *code has* optimal *ratio of* $\frac{1}{2}$.

Note that the code in Figure 4.4 is actually constructed as in Theorem 4.3. In order to prove Theorem 4.3, we first prove the following lemma. We represent each systematic node by the binary vector that generates its corresponding permutation. And define $|v \backslash u| = \sum_{i:v_i=1, u_i=0} 1$ as the number of coordinates at which v has a 1 but u has a 0.

Lemma 4.4 *(i) Let* $T \subseteq \mathbb{F}^m$ *be a set of vectors. For any* $v, u \in T$*, to rebuild node* v*, the number of accessed elements in node* u *is*

$$2^{m-1} + |f_v(X_v) \cap f_u(X_v)|.$$

(ii) If $v \neq 0$*, then*

$$|f_v(X_v) \cap f_u(X_v)| = \begin{cases} |X_v|, & |v \backslash u| \mod 2 = 0 \\ 0, & |v \backslash u| \mod 2 = 1. \end{cases} \tag{4.2}$$

Proof: (i) In the rebuilding of node v the elements in rows X_v are rebuilt by rows, thus the row parity column accesses the values of the sum of rows X_v. Therefore, the surviving node u also accesses its elements in rows X_v. Hence, by now $|X_v| = 2^{m-1}$ elements are accessed in node u.

The elements of node v in rows $\overline{X_v}$ are rebuilt by zigzags, thus the zigzag parity column accesses the values of the zigzag parity elements $\{z_{f_v(l)} : l \in \overline{X_v}\}$, and each surviving systematic node accesses its elements that are contained in the corresponding zigzag sets, unless these elements were already accessed during the rebuilding by rows. The elements of node u in rows $f_u^{-1}(f_v(\overline{X_v}))$ belong to zigzag sets $\{Z_{f_v(l)} : l \in \overline{X_v}\}$, where f_u^{-1} is the inverse permutation of f_u. Thus the extra elements node u needs to access are in rows $f_u^{-1}(f_v(\overline{X_v}))\backslash X_v$. But,

$$\begin{aligned}
& |f_u^{-1}(f_v(\overline{X_v}))\backslash X_v| \\
= \; & |\overline{f_u^{-1}(f_v(X_v))} \cap \overline{X_v}| \\
= \; & |\overline{f_u^{-1}(f_v(X_v)) \cup X_v}| \\
= \; & 2^m - |f_u^{-1}(f_v(X_v)) \cup X_v| \\
= \; & 2^m - (|f_u^{-1}(f_v(X_v))| + |X_v| - |f_u^{-1}(f_v(X_v)) \cap X_v|) \\
= \; & |f_u^{-1}(f_v(X_v)) \cap X_v| \\
= \; & |f_v(X_v) \cap f_u(X_v)|,
\end{aligned}$$

where we used the fact that f_v, f_u are bijections, and $|X_v| = 2^{m-1}$.

(ii) Consider the group $(\mathbb{F}_2^m, +)$, and recall that $f_v(X) = X + v = \{x + v : x \in X\}$. The sets $f_v(X_v) = X_v + v$ and $f_u(X_v) = X_v + u$ are cosets of the subgroup $X_v = \{w \in \mathbb{F}_2^m : w \cdot v = 0\}$, and they are either identical or disjoint. Moreover, they are identical iff $v - u \in X_v$, namely $(v-u)\cdot v = \sum_{i:v_i=1,u_i=0} 1 \equiv 0 \mod 2$. However, by definition $|v\backslash u| \equiv \sum_{i:v_i=1,u_i=0} 1 \mod 2$, and the result follows. □

Let $\{f_0, ..., f_{k-1}\}$ be a set of permutations over the set $[0, 2^m - 1]$ with associated subsets $X_0, ..., X_{k-1} \subseteq [0, 2^m - 1]$, where each $|X_i| = 2^{m-1}$. We say that this set is a set of *orthogonal permutations* if for any $i, j \in [0, k-1]$,

$$\frac{|f_i(X_i) \cap f_j(X_i)|}{2^{m-1}} = \delta_{i,j},$$

where $\delta_{i,j}$ is the Kronecker delta. Assume the code was constructed by a set of orthogonal permutations. By Lemma 4.4 only half of the information is accessed (2^{m-1} elements) in each of the surviving systematic columns during a rebuilding of a systematic column. Moreover, only 2^{m-1} elements are accessed from each parity node, too. Hence codes generated by orthogonal permutations have optimal rebuilding ratio $1/2$. Now we are ready to prove Theorem 4.3.

Proof: [Proof of Theorem 4.3] Let $i \neq 0, j$, then since $|e_i \backslash e_j| = 1$, we get by Lemma 4.4 part (ii)

$$|f_i(X_i) \cap f_j(X_i)| = 0.$$

Moreover, $f_i(X_0) = \{x + e_i : x \cdot (1,1,\ldots,1) = 0\} = \{y : y \cdot (1,1,\cdots,1) = 1\}$, and

$$\begin{aligned} & f_0(X_0) \cap f_i(X_0) \\ = \; & \{x : x \cdot (1,\cdots,1) = 0\} \cap \{y : y \cdot (1,\cdots,1) = 1\} \\ = \; & \emptyset. \end{aligned}$$

Hence the permutations $f_0, \ldots, f_m$ are orthogonal permutations, and the ratio is $1/2$ by Lemma 4.4 part (i). □

Note that the optimal code can be shortened by removing some systematic columns and still retain an optimal ratio, i.e., for any $k \leq m+1$ we have a code with optimal rebuilding.

4.2.3 Finite-Field Size

Having found the set of orthogonal permutations, we need to specify the coefficients in the parities such that the code is MDS.

Consider the $(m+3, m+1)$ code $\mathcal{C}$ constructed by Theorem 4.3 and the vectors $\{e_i\}_{i=0}^{m}$. Let $\mathbb{F}$ be the finite field in use, where the information in row i, column j is $a_{i,j} \in \mathbb{F}$. Let its row and zigzag coefficients be $\alpha_{i,j}, \beta_{i,j} \in \mathbb{F}$, respectively. For a row set $R_u = \{a_{u,0}, a_{u,1}, \ldots, a_{u,m}\}$, the row parity is $r_u = \sum_{j=0}^{m} \alpha_{u,j} a_{u,j}$. For a zigzag set $Z_u = \{a_{u,0}, a_{u+e_1,1}, \ldots, a_{u+e_m,m}\}$, the zigzag parity is $z_u = \sum_{j=0}^{m} \beta_{u+e_j,j} a_{u+e_j,j}$.

Recall that the $(m+3, m+1)$ code is indeed MDS iff we can recover the information from up to 2 columns erasures. It is clear that none of the coefficients $\alpha_{i,j}, \beta_{i,j}$ can be zero. Moreover, if we assign all the coefficients as $\alpha_{i,j} = \beta_{i,j} = 1$ we get that in an erasure of two systematic columns the set of equations derived from the parity columns are linearly dependent and thus not solvable (the sum of the equations from the row parity and the sum of those from the zigzag parity will both be the sum of the entire information array). Therefore the coefficients need to be from a field with more than one nonzero element, thus a field of size at least 3 is necessary.

Recall that we defined the permutations by binary vectors. This way of construction leads

to special structure of the code. We are going to take advantage of it and assign the coefficients accordingly. Surprisingly $\mathbb{F}_3$ is sufficient to correct two erasures.

Construction 4.5 *For the code $\mathcal{C}$ in Theorem* 4.3 *over $\mathbb{F}_3$, define $u_j = \sum_{l=0}^{j} e_l$ for $0 \leq j \leq m$. Assign row coefficients as $\alpha_{i,j} = 1$ for all i, j, and zigzag coefficients as*

$$\beta_{i,j} = 2^{i \cdot u_j} \tag{4.3}$$

where $i = (i_1, \ldots, i_m)$ is represented in binary and the calculation of the inner product in the exponent is done over $\mathbb{F}_2$.

The coefficients in Figure 4.4 are assigned by Construction 4.5. For example,

$$\beta_{3,1} = 2^{3 \cdot u_1} = 2^{(1,1)\cdot(1,0)} = 2^1 = 2.$$

$$\beta_{3,2} = 2^{3 \cdot u_2} = 2^{(1,1)\cdot(1,1)} = 2^0 = 1.$$

One can check that the code can tolerate any two erasures and hence is MDS.

The following theorem shows that the construction is MDS.

Theorem 4.6 *Construction* 4.5 *is an $(m+3, m+1)$ MDS code with optimal finite-field size of* 3.

Proof: It is easy to see that if at least one of the two erased columns is a parity column then we can recover the information. Hence we only need to show that we can recover from an erasure of any two systematic columns $i, j \in [0, m], i < j$. In this scenario, we access the entire remaining information in the array. For $r \in [0, 2^m - 1]$ set $r' = r + e_i + e_j$, and recall that $a_{r,i} \in Z_l$ iff $l = r + e_i$, thus $a_{r,i}, a_{r',j} \in Z_{r+e_i}$ and $a_{r,j}, a_{r',i} \in Z_{r+e_j}$. From the two parity columns we need to solve the following equations

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ \beta_{r,i} & 0 & 0 & \beta_{r',j} \\ 0 & \beta_{r,j} & \beta_{r',i} & 0 \end{bmatrix} \begin{bmatrix} a_{r,i} \\ a_{r,j} \\ a_{r',i} \\ a_{r',j} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}. \tag{4.4}$$

Here $y_1, \ldots, y_4 \in \mathbb{F}_3$ are the differences of the corresponding parity elements (the r-th, r'-th row parity, the $r + e_i$-th, $r + e_j$-th zigzag parity) after subtracting the weighted remaining elements in

the row/zigzag sets. This set of equations is solvable iff

$$\beta_{r,i}\beta_{r',i} \neq \beta_{r,j}\beta_{r',j}. \tag{4.5}$$

Note that the multiplicative group of $\mathbb{F}_3\backslash 0$ is isomorphic to the additive group of $\mathbb{F}_2$, hence multiplying two elements in $\mathbb{F}_3\backslash 0$ is equivalent to summing up their exponent in $\mathbb{F}_2$ when they are represented as a power of the primitive element of the field $\mathbb{F}_3$. For columns $0 \leq i < j \leq m$ and rows r, r' defined above, we have

$$\beta_{r,i}\beta_{r',i} = 2^{r\cdot u_i + r'\cdot u_i} = 2^{(r+r')\cdot u_i} = 2^{(e_i+e_j)\cdot\sum_{l=0}^{i} e_l} = 2^{e_i\cdot e_i} = 2.$$

However in the same manner we derive that

$$\beta_{r,j}\beta_{r',j} = 2^{(r+r')\cdot u_j} = 2^{(e_i+e_j)\cdot\sum_{l=0}^{j} e_l} = 2^{e_i\cdot e_i + e_j\cdot e_j} = 2^0 = 1.$$

Hence (4.5) is satisfied and the code is MDS. □

Remark: The above proof shows that $\beta_{r,i} \neq \beta_{r',i}$, and $\beta_{r,j} = \beta_{r',j}$ for $i < j$. And (4.5) is a necessary and sufficient condition for correcting erasure of columns i and j such that $v_i \neq v_j$.

It should be noted that in practice it is more convenient to use finite field $GF(2^s)$ for some integer s. In fact we can use a field of size 4 by simply modifying (4.3) to

$$\beta_{i,j} = c^{i\cdot u_j},$$

where c is a primitive element in $GF(4)$ and computations are done over $\mathbb{F}_2$ in the exponent. It is obvious that this will not affect the proof of Theorem 4.6.

In addition to optimal ratio and optimal field size, we will show in the next section that the code in Theorem 4.3 is also of optimal array size, namely, it has the maximum number of columns, given the number of rows.

4.3 Problem Settings and Properties

In this section, we prove some useful properties related to MDS array codes with optimal update.

Let $A = (a_{i,j})$ be an array of size $p \times k$ over a finite field $\mathbb{F}$, where $i \in [0, p-1], j \in [0, k-1]$, and each of its entries is an information element. Let $R = \{R_0, R_1, ..., R_{p-1}\}$ and $Z =$

$\{Z_0, Z_1, ..., Z_{p-1}\}$ be two sets such that R_l, Z_l are subsets of elements in A for all $l \in [0, p-1]$. Then for all $l \in [0, p-1]$, define the row/zigzag parity element as $r_l = \sum_{a \in R_l} \alpha_a a$ and $z_l = \sum_{a \in Z_l} \beta_a a$, for some sets of coefficients $\{\alpha_a\}, \{\beta_a\} \subseteq \mathbb{F}$. We call R and Z as the sets that generate the parity columns.

An MDS array code over $\mathbb{F}_q$ with r parities is said to be *optimal-update* if in the change of any information element only $r+1$ elements are changed in the array. It is easy to see that $r+1$ changes is the minimum possible number because if an information element appears only r times in the array, then deleting at most r columns will result in an unrecoverable r-erasure pattern and will contradict the MDS property. A small finite-field size is desirable because we can update a small amount of information at a time if needed, and also get low computational complexity. Therefore we assume that the code is optimal-update, while we try to use the smallest possible finite filed. When $r = 2$, only 3 elements in the code are updated when an information element is updated. Under this assumption, the following theorem characterizes the sets R and Z.

Theorem 4.7 *For a $(k+2, k)$ MDS code with optimal update, the sets R and Z are partitions of A into p equally sized sets of size k, where each set in R or Z contains exactly one element from each column.*

Proof: Since the code is a $(k+2, k)$ MDS code, each information element should appear at least once in each parity column C_k, C_{k+1}. However, since the code has optimal update, each element appears exactly once in each parity column.

Let $X \in R$, note that if X contains two entries of A from the systematic column C_i, $i \in [0, k-1]$, then rebuilding is impossible if columns C_i and C_{k+1} are erased. Thus X contains at most one entry from each column, therefore $|X| \leq k$. However each element of A appears exactly once in each parity column, thus if $|X| < k$, $X \in R$, there is $Y \in R$, with $|Y| > k$, which leads to a contradiction. Therefore, $|X| = k$ for all $X \in R$. As each information element appears exactly once in the first parity column, $R = \{R_0, \ldots, R_{p-1}\}$ is a partition of A into p equally sized sets of size k. Similar proof holds for the sets $Z = \{Z_0, \ldots, Z_{p-1}\}$. □

By the above theorem, for the j-th systematic column $(a_{0,j}, \ldots, a_{p-1,j})^T$, its p elements are contained in p distinct sets R_l, $l \in [0, p-1]$. In other words, the membership of the j-th column's elements in the sets $\{R_l\}$ defines a permutation $g_j : [0, p-1] \to [0, p-1]$, such that $g_j(i) = l$ iff $a_{i,j} \in R_l$. Similarly, we can define a permutation f_j corresponding to the second parity column, where $f_j(i) = l$ iff $a_{i,j} \in Z_l$. For example, in Figure 4.2 each systematic column corresponds to a

permutation of the four symbols.

Observing that there is no significance in the elements' ordering in each column, w.l.o.g. we can assume that the first parity column contains the sum of each row of A and g_j's correspond to identity permutations, i.e., $r_i = \sum_{j=0}^{k-1} \alpha_{i,j} a_{i,j}$ for some coefficients $\{\alpha_{i,j}\}$.

First we show that any set of zigzag sets $Z = \{Z_0, ..., Z_{p-1}\}$ defines a $(k+2, k)$ MDS array code over a field $\mathbb{F}$ large enough.

Theorem 4.8 *Let $A = (a_{i,j})$ be an array of size $p \times k$ and the zigzag sets be $Z = \{Z_0, ..., Z_{p-1}\}$, then there exists a $(k+2, k)$ MDS array code for A with Z as its zigzag sets over the field $\mathbb{F}$ of size greater than $p(k-1)+1$.*

In order to prove Theorem 4.8, we use the well-known Combinatorial Nullstellensatz by Alon [Alo99]:

Theorem 4.9 *(Combinatorial Nullstellensatz) [Alo99, Th 1.2] Let $\mathbb{F}$ be an arbitrary field, and let $f = f(x_1, ..., x_q)$ be a polynomial in $\mathbb{F}[x_1, ..., x_q]$. Suppose the degree of f is $\deg(f) = \sum_{i=1}^{q} t_i$, where each t_i is a nonnegative integer, and suppose the coefficient of $\prod_{i=1}^{q} x_i^{t_i}$ in f is nonzero. Then, if $S_1, ..., S_n$ are subsets of $\mathbb{F}$ with $|S_i| > t_i$, there are $s_1 \in S_1, s_2 \in S_2, ..., s_q \in S_q$ so that*

$$f(s_1, ..., s_q) \neq 0.$$

Proof: [Proof of Theorem 4.8] Assume the information of A is given in a column vector W of length pk, where column $i \in [0, k-1]$ of A is in the row set $[(ip, (i+1)p - 1]$ of W. Each systematic node i, $i \in [0, k-1]$, can be represented as $Q_i W$ where $Q_i = [0_{p \times pi}, I_{p \times p}, 0_{p \times p(k-i-1)}]$. Moreover define $Q_k = [I_{p \times p}, I_{p \times p}, ..., I_{p \times p}]$, $Q_{k+1} = [x_0 P_0, x_1 P_1, ..., x_{k-1} P_{k-1}]$ where the P_i's are permutation matrices (not necessarily distinct) of size $p \times p$, and the x_i's are variables, such that $C_k = Q_k W, C_{k+1} = Q_{k+1} W$. The permutation matrix $P_i = (p_{l,m}^{(i)})$ is defined as $p_{l,m}^{(i)} = 1$ if and only if $a_{m,i} \in Z_l$. In order to show that there exists such MDS code, it is sufficient to show that there is an assignment for the intermediates $\{x_i\}$ in the field $\mathbb{F}$, such that for any set of integers $\{s_1, s_2, ..., s_k\} \subseteq [0, k+1]$ the matrix $Q = [Q_{s_1}^T, Q_{s_1}^T, ..., Q_{s_k}^T]$ is of full rank. It is easy to see that if the parity column C_{k+1} is erased, i.e., $k+1 \notin \{s_1, s_2, ..., s_k\}$ then Q is of full rank. If $k \notin \{s_1, s_2, ..., s_k\}$ and $k+1 \in \{s_1, s_2, ..., s_q\}$ then Q is of full rank if none of the x_i's equals to zero. The last case is when both $k, k+1 \in \{s_1, s_2, ..., s_k\}$, i.e., there are $0 \leq i < j \leq k-1$ such that

$i, j \notin \{s_1, s_2, ..., s_k\}$. It is easy to see that in that case Q is of full rank if and only if the submatrix

$$B_{i,j} = \begin{pmatrix} x_i P_i & x_j P_j \\ I_{p\times p} & I_{p\times p} \end{pmatrix}$$

is of full rank. This is equivalent to $\det(B_{i,j}) \neq 0$. Note that $\deg(\det(B_{i,j})) = p$ and the coefficient of x_i^p is $\det(P_i) \in \{1, -1\}$. Define the polynomial

$$T = T(x_0, x_1, ..., x_{k-1}) = \prod_{0\leq i<j\leq k-1} \det(B_{i,j}),$$

and the result follows if there are elements $a_0, a_1, .., a_{k-1} \in \mathbb{F}$ such that $T(a_0, a_1, ..., a_{k-1}) \neq 0$. T is of degree $p\binom{k}{2}$ and the coefficient of $\prod_{i=0}^{k-1} x_i^{p(k-1-i)}$ is $\prod_{i=0}^{k-1} \det(P_i)^{k-1-i} \neq 0$. Set for any $i, S_i = \mathbb{F}\backslash 0$ in Theorem 4.9, and the result follows. □

The Theorem 4.8 states that there exist coefficients such that the code is MDS, and thus we will focus first on finding proper zigzag permutations $\{f_j\}$. The idea behind choosing the zigzag sets is as follows: assume a systematic column $(a_{0,j}, a_{1,j}, ..., a_{p-1,j})^T$ is erased. Each element $a_{i,j}$ is rebuilt either by row or by zigzag. The set $\mathbf{S} = \{S_0, S_1, ..., S_{p-1}\}$ is called a rebuilding set for column $(a_0, a_1, ..., a_{p-1})^T$ if for each i, $S_i \in R \cup Z$ and $a_i \in S_i$. In order to minimize the number of accesses to rebuild the erased column, we need to minimize the size of

$$|\cup_{i=0}^{p-1} S_i|, \tag{4.6}$$

which is equivalent to maximizing the number of intersections between the sets $\{S_i\}_{i=0}^{p-1}$. More specifically, the intersections between the row sets in $\mathbf{S}$ and the zigzag sets in $\mathbf{S}$.

For a $(k+2, k)$ MDS code $\mathcal{C}$ with p rows define the *rebuilding ratio* $R(\mathcal{C})$ as the average fraction of accesses in the surviving systematic and parity nodes while rebuilding one systematic node, i.e.,

$$R(\mathcal{C}) = \frac{\sum_j \min_{S_0,...,S_{p-1} \text{ rebuilds } j} |\cup_{i=0}^{p-1} S_i|}{p(k+1)k}.$$

Notice that in the two parity nodes, we access p elements because each erased element must be rebuilt either by row or by zigzag, however $\cup_{i=0}^{p-1} S_i$ contains p elements from the erased column. Thus the above expression is exactly the rebuilding ratio. Define the *ratio function* for all $(k+2, k)$

MDS codes with p rows as

$$R(k) = \min_{\mathcal{C}} R(\mathcal{C}),$$

which is the minimal average portion of the array needed to be accessed in order to rebuild one erased column. By (4.1), we know that $R(k) \geq 1/2$. For example, the code in Figure 4.4 achieves the lower bound of ratio $1/2$, and therefore $R(3) = 1/2$. Moreover, we will see in Corollary 4.17 that $R(k)$ is almost $1/2$ for all k and $p = 2^m$, where m is large enough.

So far we have discussed the characteristics of an arbitrary MDS array code with optimal update. Next, let us look at our code in Construction 4.1.

Recall that by Theorem 4.8 this code can be an MDS code over a field large enough. The ratio of the constructed code will be proportional to the size of the union of the elements in the rebuilding set in (4.6). The following theorem gives the ratio for Construction 4.1 and can be easily derived from Lemma 4.4 part (i). Recall that given vectors $v_0, \ldots, v_{k-1}$, we write $f_i = f_{v_i}$ and $X_i = X_{v_i}$.

Theorem 4.10 *The code described in Construction* 4.1 *and generated by the vectors* $v_0, v_1, ..., v_{k-1}$ *is a* $(k+2, k)$ *MDS array code with ratio*

$$R = \frac{1}{2} + \frac{\sum_{i=0}^{k-1} \sum_{j \neq i} |f_i(X_i) \cap f_j(X_i)|}{2^m k(k+1)}. \tag{4.7}$$

Note that different orthogonal sets of permutations can generate equivalent codes, hence we define equivalence of two sets of orthogonal permutations as follows. Let $F = \{f_1, f_2, \ldots, f_{k-1}, f_0\}$ be an orthogonal set of permutations over integers $[0, p-1]$, associated with subsets $X_1, X_2, \ldots, X_{k-1}, X_0$. And let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, \sigma_0\}$ be another orthogonal set over $[0, p-1]$ associated with subsets $Y_1, Y_2, \ldots, Y_{k-1}, Y_0$. Then F and Σ are said to be *equivalent* if there exist permutations g, h such that $\forall i \in [0, k-1]$,

$$h f_i g = \sigma_i,$$

$$g^{-1}(X_i) = Y_i.$$

Note that multiplying g on the right is the same as permuting the rows of the systematic nodes, and multiplying h on the left permutes the rows of the second parity node. Therefore, codes constructed using F or Σ are essentially the same.

In particular, let us assume that the permutations are over integers $[0, 2^m - 1]$, and the set of permutations Σ and the subsets Y_i's are the same as in Theorem 4.3: $\sigma_i = f_{e_i}$, $Y_i = \{x \in [0, 2^m -$

$1] : x \cdot e_i = 0\}$, and $Y_0 = \{x \in [0, 2^m - 1] : x \cdot (1, 1, \ldots, 1) = 0\}$. Next we show the optimal code in Theorem 4.3 is optimal in size, namely, it has the maximum number of columns given the number of rows. In addition any optimal-update, optimal-access code with maximum size is equivalent to the construction using standard-basis vectors.

Theorem 4.11 *Let F be an orthogonal set of permutations over the integers $[0, 2^m - 1]$,*
(i) the size of F is at most $m + 1$;
(ii) if $|F| = m + 1$ then it is equivalent to Σ defined by the standard basis and zero vector.

Proof: We will prove it by induction on m. For $m = 0$ there is nothing to prove. (i) We first show that $|F| = k \leq m + 1$. It is trivial to see that for any permutations g, h on $[0, 2^m - 1]$, the set $hFg = \{hf_0g, hf_1g, ..., hf_{k-1}g\}$ is also a set of orthogonal permutations with sets $g^{-1}(X_0), g^{-1}(X_1), ..., g^{-1}(X_{k-1})$. Thus w.l.o.g. we can assume that f_0 is the identity permutation and $X_0 = [0, 2^{m-1} - 1]$. From the orthogonality we get that

$$\cup_{i=1}^{k-1} f_i(X_0) = \overline{X_0} = [2^{m-1}, 2^m - 1].$$

We claim that for any $i \neq 0, |X_i \cap X_0| = \frac{|X_0|}{2} = 2^{m-2}$. Assume the contrary, thus if $|X_i \cap X_0| > 2^{m-2}$, then for any distinct $i, j \in [1, k-1]$ we get that

$$f_j(X_i \cap X_0), f_i(X_i \cap X_0) \subseteq \overline{X_0}, \tag{4.8}$$

$$|f_j(X_i \cap X_0)| = |f_i(X_i \cap X_0)| > 2^{m-2} = \frac{|\overline{X_0}|}{2}. \tag{4.9}$$

From equations (4.8) and (4.9) we conclude that $f_j(X_i \cap X_0) \cap f_i(X_i \cap X_0) \neq \emptyset$, which contradicts the orthogonality property. If $|X_i \cap \overline{X_0}| > 2^{m-2}$ the contradiction follows by a similar reasoning. Define the set of permutations $F^* = \{f_i^*\}_{i=1}^{k-1}$ over the set of integers $[0, 2^{m-1} - 1]$ by $f_i^*(x) = f_i(x) - 2^{m-1}$, which is a set of orthogonal permutations with sets $X_i^* = \{X_i \cap X_0\}, i = 1, ..., k-1$. By induction $k - 1 \leq m$ and the result follows.

(ii) Next we show that if $|F| = m + 1$ then it is equivalent to Σ associated with $\{Y_i\}$. Let $F = \{f_1, f_2, \ldots, f_m, f_0\}$. Take two permutations g', h' such that

$$g'^{-1}(X_1) = Y_1$$

and $h' f_1 g'(Y_1) = \overline{Y_1}$. Define $f'_i = h' f_i g'$ for all $i \in [0, m]$. Then

$$f'_1(Y_1) = \overline{Y_1}, f'_1(\overline{Y_1}) = Y_1.$$

The new set of permutations $\{f'_i\}_{i=0}^m$ is also orthogonal with subsets $\{g'^{-1}(X_i)\}_{i=0}^m$, so $f'_i(Y_1) \cap f'_1(Y_1) = \emptyset$. Hence for all $i \neq 1$

$$f'_i(Y_1) = Y_1 = [0, 2^{m-1} - 1],$$

$$f'_i(\overline{Y_1}) = \overline{Y_1} = [2^{m-1}, 2^m - 1].$$

By similar argument of part (i), we know $\{f'_2, \ldots, f'_m, f'_0\}$ restricted to Y_1 (or to $\overline{Y_1}$) is an orthogonal set of permutations, associated with subsets $Y_1 \cap g'^{-1}(X_i)$ (or with $\overline{Y_1} \cap g'^{-1}(X_i)$, respectively), $i \neq 1$. By the induction hypothesis, there exist permutations p, q over Y_1 such that for $i \neq 1$

$$\begin{aligned} \sigma_i &= p f'_i q, \\ q^{-1}(Y_1 \cap g'^{-1}(X_i)) &= Y_1 \cap Y_i, \end{aligned} \tag{4.10}$$

where σ_i, f'_i are restricted to Y_1. Similarly, there exist permutations r, s over $\overline{Y_1}$ such that for $i \neq 1$

$$\begin{aligned} \sigma_i &= r f'_i s, \\ s^{-1}(\overline{Y_1} \cap g'^{-1}(X_i)) &= \overline{Y_1} \cap Y_i, \end{aligned} \tag{4.11}$$

where σ_i, f'_i are restricted to $\overline{Y_1}$. Define permutation g'' over $[0, 2^m - 1]$ as the union of q and s: $g''(x) = q(x)$ if $x \in Y_1$, and $g''(x) = s(x)$ if $x \in \overline{Y_1}$. Also define h'' over $[0, 2^m - 1]$ as the union of p and r. So g'', h'' map Y_1 (or $\overline{Y_1}$) to itself. We will show that $\{f_i\}_{i=0}^m$ is equivalent to Σ using $g = g' g''$ and $h = h'' h'$. For $i \neq 1$, this is obvious from (4.10)(4.11). For $i = 1$, we have

$$g^{-1}(X_1) = g''^{-1} g'^{-1}(X_1) = g''^{-1}(Y_1) = Y_1.$$

We know $\sigma_i = h f_i g$, for $i \neq 1$. Let $f = h f_1 g$ and we will show $f = \sigma_1$. By orthogonality $f(Y_i) \cap \sigma_i(Y_i) = \emptyset$ for $i \neq 1$. It is easy to see that for $i \in [2, m]$, $\sigma_i(Y_i) = \overline{Y_i}$. Hence for $i \in [2, m]$

$$f(Y_i) = Y_i, f(\overline{Y_i}) = \overline{Y_i}. \tag{4.12}$$

	standard basis	duplication of standard basis	constant weight vectors
# sys. nodes	$m+1$	$s(m+1)$	$O(m^c)$
ratio	$\frac{1}{2}$	$\frac{1}{2}+\frac{s-1}{2s(m+1)+2} \approx \frac{1}{2}+\frac{1}{2(m+1)}$	$\frac{1}{2}+\frac{c^2}{2m}$
field size	3	$s+2$	2^c+1

Figure 4.5: Comparison among codes constructed by the standard basis and zero vector, by s-duplication of standard basis and zero vector, and by constant weight vectors. The number of systematic nodes, the rebuilding ratio, and the finite-field size are listed. We assume that all the codes have 2 parities and 2^m rows. For the duplication code, the rebuilding ratio is obtained when the number of copies s is large. For the constant weight code, the weight of each vector is equal to c, which is an odd number and relatively small compared to m.

Moreover, by construction $f(Y_1) = h''f_1'g''(Y_1) = h''f_1'(Y_1) = h''(\overline{Y_1}) = \overline{Y_1}$, so

$$f(Y_1) = \overline{Y_1}, f(\overline{Y_1}) = Y_1. \tag{4.13}$$

Any integer $x \in [0, 2^m - 1]$ can be written as the intersection of Y_i or $\overline{Y_i}$, for all $i \in [m]$, depending on its binary representation. For example, $x = 1$ means $\{x\} = \cap_{i=1}^{m-1} Y_i \cap \overline{Y_m}$. For another example if $x = 0$ then $\{x\} = \cap_{i=1}^{m} Y_i$, and $f(\{0\}) = f(\cap_{i=1}^{m} Y_i) = \cap_{i=2}^{m} Y_i \cap \overline{Y_1} = \{2^{m-1}\}$ by (4.12)(4.13) and since f is a bijection. Thus $f(0) = 2^{m-1}$. By a similar argument, $f(x) = 2^{m-1} + x$ for all x and

$$f = \sigma_1.$$

Thus the proof is completed. □

Note that by similar reasoning we can show that if $|F| = m$, it is equivalent to $\{\sigma_1, \ldots, \sigma_m\}$ defined by the standard basis. Part (ii) in the above theorem says that if we consider codes with optimal update, optimal access, and optimal size, then they are equivalent to the standard-basis construction. In this sense, Theorem 4.3 gives the *unique* code. Moreover, if we find the smallest finite field for one code (as in Construction 4.5), there does not exist a code using a smaller field.

Part (i) of the above theorem implies that the number of rows has to be exponential in the number of columns in any systematic code with optimal ratio and optimal update. Notice that the code in Theorem 4.3 achieves the *maximum* possible number of columns, $m+1$. An exponential number of rows can be practical in some storage systems, since they are composed of dozens of nodes (disks) each of which has size in an order of gigabytes. However, a code may corresponds to only a small portion of each disk and we will need the flexibility of the array size. The following example shows a code of flatter array size with a cost of a small increase in the ratio.

Example 4.12 *Let* $T = \{v \in \mathbb{F}_2^m : \|v\|_1 = 3\}$ *be the set of vectors with weight 3 and length* m*. Notice that* $|T| = \binom{m}{3}$*. Construct the code* $\mathcal{C}$ *by* T *according to Construction* 4.1*. Given* $v \in T$*,* $|\{u \in T : |v \backslash u| = 3\}| = \binom{m-3}{3}$*, which is the number of vectors with 1's in different positions than* v*. Similarly,* $|\{u \in T : |v \backslash u| = 2\}| = 3\binom{m-3}{2}$ *and* $|\{u \in T : |v \backslash u| = 1\}| = 3(m-3)$*. By Theorem* 4.10 *and Lemma* 4.4*, for large* m *the ratio is*

$$\frac{1}{2} + \frac{2^{m-1}\binom{m}{3}3\binom{m-3}{2}}{2^m\binom{m}{3}(\binom{m}{3}+1)} \approx \frac{1}{2} + \frac{9}{2m}.$$

Note that this code reaches the lower bound of the ratio as m tends to infinity, and has $O(m^3)$ columns. More discussions on increasing the number of columns is presented in the next section.

4.4 Lengthening the Code

As we mentioned, it is sometimes useful to construct codes with longer k given the number of rows in the array. In this section we will provide two ways to reach this goal: we will first modify Example 4.12 and obtain an MDS code with a small finite field. Increasing the number of columns can also be done using code duplication (Theorem 4.16). In both methods, we sacrifice the optimal-ratio property for longer k, and the ratio is asymptotically optimal in both cases. Figure 4.5 summarizes the tradeoffs of different constructions. We will study the table in more details in the end of this section.

4.4.1 Constant Weight Vector

We will first give a construction based on Example 4.12 where all the binary vectors used have a constant weight. And we also specify the finite-field size of the code.

Construction 4.13 *Let* $3|m$*, and consider the following set of vectors* $S \subseteq \mathbb{F}_2^m$*: for each vector* $v = (v_1, \ldots, v_m) \in S$*,* $\|v\|_1 = 3$ *and* $v_{i_1}, v_{i_2}, v_{i_3} = 1$ *for some* $i_1 \in [1, m/3], i_2 \in [m/3+1, 2m/3], i_3 \in [2m/3+1, m]$*. For simplicity, we write* $v = \{i_1, i_2, i_3\}$*. Construct the* $(k+2, k)$ *code as in Construction* 4.1 *using the set of vectors* S*, hence the number of systematic columns is* $k = |S| = (\frac{m}{3})^3 = \frac{m^3}{27}$*. For any* $i \in [jm/3+1, (j+1)m/3]$ *and some* $j = 0, 1, 2$ *, define a row vector* $M_i = \sum_{l=jm/3+1}^{i} e_l$*. Then define a* $m \times 3$ *matrix*

$$M_v = \begin{bmatrix} M_{i_1}^T & M_{i_2}^T & M_{i_3}^T \end{bmatrix}$$

for $v = \{i_1, i_2, i_3\}$. Let a be a primitive element of $\mathbb{F}_9$. Assign the row coefficients as 1 and the zigzag coefficient for row r, column v as a^t, where $t = rM_v \in \mathbb{F}_2^3$ (in its binary expansion).

For example, let $m = 6$, and $v = \{1, 4, 6\} = (1, 0, 0, 1, 0, 1) \in S$. The corresponding matrix is

$$M_v = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T.$$

For row $r = 26 = (0, 1, 1, 0, 1, 0)$, we have

$$t = rM_v = (0, 1, 1) = 3,$$

and the zigzag coefficient is a^3.

Theorem 4.14 *Construction* 4.13 *is a $(k+2, k)$ MDS code with array size $2^m \times (k+2)$ and $k = m^3/27$. Moreover, the rebuilding ratio is $\frac{1}{2} + \frac{9}{2m}$ for large m.*

Proof: For each vector $v \in S$, there are $3(m/3-1)^2$ vectors $u \in S$ such that they have one 1 in the same location as v, i.e., $|v \backslash u| = 2$. Hence by Theorem 4.10 and Lemma 4.4, for large m the ratio is

$$\frac{1}{2} + \frac{3((\frac{m}{3})-1)^2}{2(\frac{m^3}{27}+1)} \approx \frac{1}{2} + \frac{9}{2m}.$$

Next we show that the MDS property of the code holds. Consider columns u, v for some $u = \{i_1, i_2, i_3\} \neq v = \{j_1, j_2, j_3\}$ and $i_1, j_1 \in [1, m/3], i_2, j_2 \in [m/3+1, 2m/3], i_3, j_3 \in [2m/3+1, m]$. Consider rows r and $r' = r + u + v$. The condition for the MDS property from (4.5) becomes

$$a^{rM_u^T + r'M_u^T \mod 8} \neq a^{rM_v^T + r'M_v^T \mod 8} \tag{4.14}$$

where each vector of length 3 is viewed as an integer in $[0, 7]$ and the addition is the usual addition mod 8. Since $v \neq u$, let $l \in [1, 3]$ be the largest index such that $i_l \neq j_l$. W.l.o.g. assume that $i_l < j_l$, hence by the remark after Theorem 4.6

$$rM_{i_l}^T \neq r'M_{i_l}^T \tag{4.15}$$

and

$$rM_{j_l}^T = r'M_{j_l}^T. \tag{4.16}$$

Note that for all t, $l < t \leq 3$, $i_t = j_t$, then since $r'M_{i_t}^T = (r + e_{i_t} + e_{j_t})M_{i_t}^T = rM_{i_t}^T$, we have

$$rM_{i_t}^T = r'M_{i_t}^T = rM_{j_t}^T = r'M_{j_t}^T. \tag{4.17}$$

It is easy to infer from (4.15),(4.16),(4.17) that the l-th bit in the binary expansions of $rM_u^T + r'M_u^T \mod 8$ and $rM_v^T + r'M_v^T \mod 8$ are not equal. Hence (4.14) is satisfied, and the result follows. □

Notice that if we do mod 15 in (4.14) instead of mod 8, the proof still follows because 15 is greater than the largest possible sum in the equation. Therefore, a field of size 16 is also sufficient to construct an MDS code, and it is easier to implement in a storage system.

Construction 4.13 can be easily generalized to any constant c such that it contains $O(m^c)$ columns and it uses any field of size at least $2^c + 1$. For simplicity assume that $c|m$, and simply construct the code using the set of vectors $\{v\} \subset \mathbb{F}_2^m$ such that $\|v\|_1 = c$, and for any $j \in [0, c-1]$, there is a unique $i_j \in [jm/c + 1, (j+1)m/c]$ and $v_{i_j} = 1$. Moreover, the finite field of size 2^{c+1} is also sufficient to make it an MDS code. When c is odd the code has ratio of $\frac{1}{2} + \frac{c^2}{2m}$ for large m.

4.4.2 Code Duplication

Next we are going to duplicate the code to increase the number of columns in the constructed $(k+2, k)$ MDS codes, such that k does not depend on the number of rows, and the rebuilding ratio is approximately $\frac{1}{2}$. Then we will show the optimality of the duplication code based on the standard basis. After that, finite-field size will be analyzed.

The constructions so far assume that each zigzag permutation appears only once in the systematic columns. The key idea of code duplication is to use a multiset of permutations to define the zigzag parity. Let $\mathcal{C}$ be a $(k+2, k)$ array code with p rows, where the zigzag sets $\{Z_l\}_{l=0}^{p-1}$ are defined by the set of permutations $\{f_i\}_{i=0}^{k-1}$ acting on the integers $[0, p-1]$. For an integer s, an *s-duplication code* of $\mathcal{C}$, denoted by $\mathcal{C}'$, is an $(sk+2, sk)$ MDS code with zigzag permutations defined by duplicating the k permutations s times each. The formal definition and rebuilding algorithm are as follows. We are going to use superscripts to represent different copies of the ordinal code.

Construction 4.15 *Define the multiset of permutations $F = \{f_0, \ldots, f_{k-1}, f_0, \ldots, f_{k-1}, \ldots, f_0, \ldots, f_{k-1}\}$, where each permutation f_j has multiplicity s, for all $j \in [0, k-1]$. In order to distinguish*

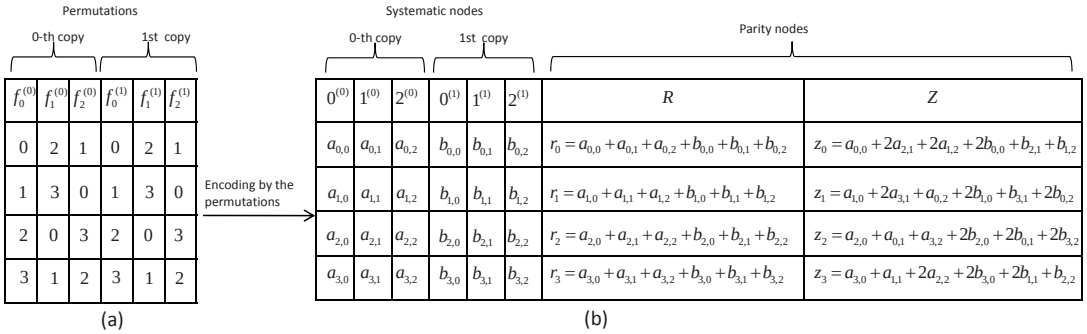

Figure 4.6: A 2-duplication of the code in Figure 4.4. The code has 6 information nodes and 2 parity nodes. The rebuilding ratio is $4/7$.

different copies of the same permutation, denote the t-th f_j as $f_j^{(t)}$. Let the $p \times sk$ information array be $[A^{(0)}, A^{(1)}, \ldots, A^{(s-1)}] = [(a_{i,j}^{(0)}), (a_{i,j}^{(1)}), \ldots, (a_{i,j}^{(s-1)})]$, with $i \in [0, p-1], j \in [0, k-1]$. Define the zigzag sets $Z_0, ..., Z_{p-1}$ as $a_{i,j}^{(t)} \in Z_l$ if $f_j(i) = l$. Notice that this definition is independent of t. For the s-duplication code $\mathcal{C}'$, let the first parity still be the row parity, and the second parity be the zigzag parity according to the above zigzag sets. Denote the column corresponding to $f_j^{(t)}$ as column $j^{(t)}$, $0 \leq t \leq s-1$. Call the columns $\{j^{(t)} : j \in [0, k-1]\}$ the t-th copy of the original code.

Suppose in the optimal rebuilding algorithm of $\mathcal{C}$ for column i, elements of rows $J = \{j_1, j_2, \ldots, j_u\}$ are rebuilt by zigzags, and the rest by rows. In $\mathcal{C}'$, all the s columns corresponding to f_i are rebuilt in the same way: the elements in rows J are rebuilt by zigzags, and the rest by rows.

In order to make the code MDS, the coefficients in the parities may be different from the original code $\mathcal{C}$. An example of a 2-duplication of the code in Figure 4.4 is illustrated in Figure 4.6. Columns $0^{(0)}, 1^{(0)}, 2^{(0)}$ is the 0th copy of the original code, and columns $0^{(1)}, 1^{(1)}, 2^{(1)}$ is the 1st copy.

Theorem 4.16 *If a $(k+2, k)$ code $\mathcal{C}$ has rebuilding ratio $R(\mathcal{C})$, then its s-duplication code $\mathcal{C}'$ has rebuilding ratio $R(\mathcal{C})(1 + \frac{s-1}{sk+1})$.*

Proof: We will show that the rebuilding method in Construction 4.15 has rebuilding ratio of $R(\mathcal{C})(1 + \frac{s-1}{sk+1})$, and is actually optimal.

W.l.o.g. assume column $i^{(0)}$ is erased. Since column $i^{(t)}$, $t \in [1, s-1]$ corresponds to the same zigzag permutation as the erased column, for the erased element in the l-th row, no matter if it is rebuilt by row or by zigzag, we have to access the element in the l-th row and column $i^{(t)}$ (e.g., permutations $f_0^{(0)}, f_0^{(1)}$ and the corresponding columns $0^{(0)}, 0^{(1)}$ in Figure 4.6). Hence all the

elements in column $i^{(t)}$ must be accessed. Moreover, the optimal way to access the other surviving columns cannot be better than the optimal way to rebuild in the code $\mathcal{C}$. Thus the proposed algorithm has optimal rebuilding ratio.

When column $i^{(0)}$ is erased, the average (over all $i \in [0, k-1]$) of the number of elements needed to be accessed in columns $l^{(t)}$, for all $l \in [0, k-1], l \neq i$ and $t \in [0, s-1]$ is

$$R(\mathcal{C})p(k+1) - p.$$

Here the term $-p$ corresponds to the access of the parity nodes in $\mathcal{C}$. Moreover, we need to access all the elements in columns $i^{(t)}, 0 < t \leq s-1$, and access p elements in the two parity columns. Therefore, the rebuilding ratio is

$$\begin{aligned} R(\mathcal{C}') &= \frac{s(R(\mathcal{C})p(k+1) - p) + (s-1)p + p}{p(sk+1)} \\ &= R(\mathcal{C})\frac{s(k+1)}{sk+1} \\ &= R(\mathcal{C})(1 + \frac{s-1}{sk+1}) \end{aligned}$$

and the proof is completed. □

Theorem 4.16 gives us the rebuilding ratio of the s-duplication of a code $\mathcal{C}$ as a function of its rebuilding ratio $R(\mathcal{C})$. As a result, for the optimal-rebuilding ratio code in Theorem 4.3, the rebuilding ratio of its duplication code is slightly more than $1/2$, as the following corollary suggests.

Corollary 4.17 *The s-duplication of the code in Theorem* 4.3 *has ratio* $\frac{1}{2}(1 + \frac{s-1}{s(m+1)+1})$*, which is* $\frac{1}{2} + \frac{1}{2(m+1)}$ *for large s.*

For example, we can rebuild the column $1^{(0)}$ in Figure 4.6 by accessing the elements in rows $\{0, 1\}$ and in columns $0^{(0)}, 2^{(0)}, 0^{(1)}, 2^{(1)}, R, Z$, and all the elements in column $1^{(1)}$. The rebuilding ratio for this code is $4/7$.

Using duplication we can have *arbitrarily large number of columns*, independent of the number of rows. Moreover the above corollary shows that it also has an almost optimal ratio. The next obvious question to be asked is: The duplication of which set of permutations will give the best asymptotic rebuilding ratio, when the number of duplications s tends to infinity? The following theorem states that if we restrict ourselves to codes constructed using Construction 4.1 then the duplication of the permutations generated by the standard basis, gives the best asymptotic ratio.

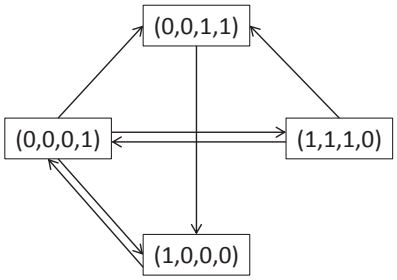

Figure 4.7: The induced subgraph of D_4 for the set of vertices $H = \{(0,0,0,1),(0,0,1,1),(1,1,1,0),(1,0,0,0)\}$.

Theorem 4.18 *The optimal asymptotic ratio among all codes constructed using duplication and Construction* 4.1 *is* $\frac{1}{2}(1+\frac{1}{m})$ *and is achieved using the standard basis.*

In order to prove the theorem, we need to define a related graph and to prove an extra theorem and lemma. Define the directed graph $D_m = D_m(V,E)$ as $V = \{v \in \mathbb{F}_2^m : v \neq 0\}$, and $E = \{(v_1,v_2) : |v_2 \backslash v_1| = 1 \mod 2\}$. Hence the vertices are the nonzero binary vectors of length m, and there is a directed edge from v_1 to v_2 if $|v_2 \backslash v_1|$ is odd. Let H be an induced subgraph of D_m on a subset of V. Let S and T be two disjoint subsets of vertices of H. We define the density between S and T to be $d_{S,T} = \frac{E_{S,T}}{2|S||T|}$, and the density of the set S to be $d_S = \frac{E_S}{|S|^2}$, where E_S is the number of edges with both of its endpoints in S, and $E_{S,T}$ is the number of edges incident with a vertex in S and a vertex in T.

For example, suppose the vertices of H are the vectors $(0,0,0,1),(0,0,1,1),(1,1,1,0),(1,0,0,0)$. The graph H is shown in Figure 4.7. The density of the graph is $d_H = 7/16$, and for $S = (1,0,0,0), T = \{(0,0,1,1),(1,1,1,0)\}$ the density is $d_{S,T} = 1/2$. Denote by $\mathcal{C}(H)$ the code constructed using the vertices of H, and Construction 4.1. In this example the code $\mathcal{C}(H)$ has four systematic disks and is encoded using the permutations generated by the four vectors of H. Duplication of the code $\mathcal{C}(H)$, s times, namely duplicating s times the permutations generated by the vectors of H will yield to a code with $4s$ systematic disks and two parities.

Let v_1, v_2 be two vertices of H, such that there is a directed edge from v_1 to v_2. By Lemma 4.4 we know that this edge means $f_{v_2}(X_{v_2}) \cap f_{v_1}(X_{v_2}) = \emptyset$, therefore only half of the information from the column corresponding to v_1 is accessed and read while rebuilding the column corresponding to v_2. Note that this observation is also correct for an s-duplication code of $\mathcal{C}(H)$. Namely, if we rebuild any column corresponding to a copy of v_2, only half of the information is accessed in any of the columns corresponding to a copy of v_1. Intuitively, when the number of copies s is large, the

density of the graph captures how often such savings will occur. The following theorem shows that the asymptotic ratio of any code constructed using Construction 4.1 and duplication is a function of the density of the corresponding graph H.

Theorem 4.19 *Let H be an induced subgraph of D_m. Let $\mathcal{C}_s(H)$ be the s-duplication of the code constructed using the vertices of H and Construction* 4.1. *Then the asymptotic ratio of $\mathcal{C}_s(H)$ is*

$$\lim_{s\to\infty} R(\mathcal{C}_s(H)) = 1 - \frac{d_H}{2}$$

Proof: Let the set of vertices and edges of H be $V(H) = \{v_i\}$ and $E(H)$, respectively. Denote by $v_i^{(l)}$, $v_i \in V(H), l \in [0, s-1]$, the l-th copy of the column corresponding to the vector v_i. In the rebuilding of column $v_i^{(l)}, l \in [0, s-1]$ each remaining systematic column $v_j^{(l')}, l' \in [0, s-1]$, needs to access all of its 2^m elements unless $|v_i \backslash v_j|$ is odd, and in that case it only has to access 2^{m-1} elements. Hence the total amount of accessed information for rebuilding this column is

$$(s|V(H)| - 1)2^m - \deg^+(v_i)s2^{m-1},$$

where $\deg^+$ is the indegree of v_i in the induced subgraph H. Averaging over all the columns in $\mathcal{C}_s(H)$ we get the ratio:

$$\begin{aligned}
&R(\mathcal{C}_s(H)) \\
=&\frac{\sum_{v_i^l \in \mathcal{C}_s(H)} (s|V(H)| - 1)2^m - \deg^+(v_i)s2^{m-1}}{s|V(H)|(s|V(H)| + 1)2^m} \\
=&\frac{s|V(H)|(s|V(H)| - 1)2^m - s^2 \sum_{v_i \in V(H)} \deg^+(v_i)2^{m-1}}{s|V(H)|(s|V(H)| + 1)2^m} \\
=&\frac{s|V(H)|(s|V(H)| - 1)2^m - s^2|E(H)|2^{m-1}}{s|V(H)|(s|V(H)| + 1)2^m}.
\end{aligned}$$

Hence

$$\lim_{s\to\infty} R(\mathcal{C}_s(H)) = 1 - \frac{|E(H)|}{2|V(H)|^2} = 1 - \frac{d_H}{2}.$$

□

We conclude from Theorem 4.19 that the asymptotic ratio of any code using duplication and a set of binary vectors $\{v_i\}$ is a function of the density of the induced subgraph on this set of vertices. Hence the induced subgraph of D_m with maximal density corresponds to the code with optimal

asymptotic ratio. It is easy to check that the induced subgraph with its vertices as the standard basis $\{e_i\}_{i=1}^m$ has density $\frac{m-1}{m}$. In fact this is the maximal possible density among all the induced subgraphs and therefore it gives a code with the best asymptotic ratio, but in order to show it we need the following technical lemma.

Lemma 4.20 *Let $D = D(V, E)$ be a directed graph and S, T be a partition of V, i.e., $S \cap T = \emptyset, S \cup T = V$, then*

$$d_V \leq \max\{d_S, d_T, d_{S,T}\}$$

Proof: Note that $d_V = \frac{|S|^2 d_S + |T|^2 d_T + 2|S||T| d_{S,T}}{|V|^2}$. W.l.o.g assume that $d_S \geq d_T$ therefore if $d_S \geq D_{S,T}$,

$$\begin{aligned} d_V &= \frac{|S|^2 d_S + |T|^2 d_T + 2|S||T| d_{S,T}}{|V|^2} \\ &\leq \frac{|S|^2 d_S + |T|^2 d_S - |T|^2 d_S + |T|^2 d_T + 2|S||T| d_S}{|V|^2} \\ &= \frac{d_S(|S| + |T|)^2 - |T|^2(d_S - d_T)}{|V|^2} \\ &\leq d_S. \end{aligned}$$

If $d_{S,T} \geq \max\{d_S, d_T\}$ then,

$$\begin{aligned} d_V &= \frac{|S|^2 d_S + |T|^2 d_T + 2|S||T| d_{S,T}}{|V|^2} \\ &\leq \frac{|S|^2 d_{S,T} + |T|^2 d_{S,T} + 2|S||T| d_{S,T}}{|V|^2} \\ &= d_{S,T} \end{aligned}$$

and the result follows. □

Now we are ready to prove the optimality of the duplication of the code using the standard basis, if we assume that the number of copies s tends to infinity. We will show that for any induced subgraph H of D_m, $d_H \leq \frac{m-1}{m}$. Hence the optimal asymptotic ratio among all codes constructed using duplication and Construction 4.1 is $1 - \frac{1}{2}\frac{m-1}{m} = \frac{1}{2}(1 + \frac{1}{m})$, and is achieved using the standard basis.

Proof: [Proof of Theorem 4.18] We say that a binary vector is an even (odd) vector if it has an

even (odd) weight. For two binary vectors v_1, v_2, $|v_2 \backslash v_1|$ being odd is equivalent to

$$1 = v_2 \cdot \overline{v_1} = v_2 \cdot ((1, ..., 1) + v_1) = \|v_2\|_1 + v_2 \cdot v_1.$$

Hence, one can check that when v_1, v_2 have the same parity, there are either no edges or 2 edges between them. Moreover, when their parities are different, there is exactly one edge between the two vertices.

When $m = 1$, the graph D_1 has only one vertex and the only nonempty induced subgraph is itself. $d_H = d_{D_1} = 0 = \frac{m-1}{m}$. When $m = 2$, the graph D_2 has three vertices and one can check that the induced subgraph with maximum density contains $v_1 = (1, 0), v_2 = (0, 1)$, and the density is $1/2 = (m-1)/m$.

For $m > 2$, assume to the contrary that there exists a subgraph of D_m with density greater than $\frac{m-1}{m}$. Let H a subgraph of D_m with minimum number of vertices among all subgraphs with maximal density. Hence for any subset of vertices $S \subsetneq V(H)$, we have $d_S < d_H$. Therefore from Lemma 4.20 we conclude that for any nontrivial partition S, T of $V(H)$, $d_H \leq d_{S,T}$. If H contains both even and odd vectors, denote by S and T the set of even and odd vectors of H, respectively. Since between any even and any odd vertex there is exactly one directed edge we get that $d_H \leq d_{S,T} = \frac{1}{2}$. However

$$\frac{1}{2} < \frac{m-1}{m} < d_H,$$

and we get a contradiction. Thus H contains only odd vectors or even vectors.

Let $V(H) = \{v_1, ..., v_k\}$. If this set of vectors is independent then $k \leq m$ and the outgoing degree for each vertex v_i is at most $k-1$ hence $d_H = \frac{E(H)}{|V(H)|^2} \leq \frac{k(k-1)}{k^2} \leq \frac{m-1}{m}$ and we get a contradiction. Hence assume that the dimension of the subspace spanned by these vectors in $\mathbb{F}_2^m$ is $l < k$ where $v_1, v_2, ...v_l$ are basis for it. Define $S = \{v_1, ...v_l\}, T = \{v_{l+1}, ..., v_k\}$. The following two cases show that the density cannot be higher than $\frac{m-1}{m}$.

H **contains only odd vectors:** Let $u \in T$. Since $u \in \text{span}\{S\}$ there is at least one $v \in S$ such that $u \cdot v \neq 0$ and thus $(u, v), (v, u) \notin E(H)$, therefore the number of directed edges between u and S is at most $2(l-1)$ for all $u \in T$, which means

$$d_H \leq d_{S,T} \leq \frac{2(l-1)|T|}{2|S||T|} = \frac{l-1}{l} \leq \frac{m-1}{m}$$

and we get a contradiction.

H **contains only even vectors:** Since the v_i's are even the dimension of $\text{span}\{S\}$ is at most $m-1$ (since, for example, $(1,0,...,0) \notin \text{span}\{S\}$) thus $l \leq m-1$. Let H^* be the induced subgraph of D_{m+1} with vertices $V(H^*) = \{(1,v_i)|v_i \in V(H))\}$. It is easy to see that all the vectors of H^* are odd, $((1,v_i),(1,v_j)) \in E(H^*)$ if and only if $(v_i,v_j) \in E(H)$, and the dimension of $\text{span}\{V(H^*)\}$ is at most $l+1 \leq m$. Having already proven the case for odd vectors, we conclude that

$$\begin{aligned} d_H = d_{H^*} &\leq \frac{\dim(\text{span}\{V(H^*)\})-1}{\dim(\text{span}\{V(H^*)\})} \\ &\leq \frac{l+1-1}{l+1} \\ &\leq \frac{m-1}{m}, \end{aligned}$$

and we get a contradiction. □

Next we address the problem of finding proper coefficients' assignments in the parities in order to make the code MDS. Let $\mathcal{C}'$ be the s-duplication of the optimal code of Theorem 4.3 and Corollary 4.17. Denote the coefficients for the element in row i and column $j^{(t)}$ by $\alpha_{i,j}^{(t)}$ and $\beta_{i,j}^{(t)}$, $0 \leq t \leq s-1$. Let $\mathbb{F}_q$ be a field of size q with primitive element a.

Construction 4.21 *Let $\mathbb{F}_q$ be a field of size at least $q \geq s+1+1_q$, where*

$$1_q = \begin{cases} 1, & q \text{ is even} \\ 0, & \text{else.o.w.} \end{cases}$$

Assign in $\mathcal{C}'$, for any i,j and $t \in [0,s-1]$, $\alpha_{i,j}^{(t)} = 1$,

$$\beta_{i,j}^{(t)} = \begin{cases} a^{(t+1)(1-2\cdot 1_q)}, & \text{if } u_j \cdot i = 1 \\ a^{t+1_q}, & \text{o.w.} \end{cases}$$

where $u_j = \sum_{l=0}^{j} e_l$.

Notice that the coefficients in each duplication have the same pattern as Construction 4.5 except that values 1 and 2 are replaced by a^t and a^{t+1} if q is odd (or a^{t+1} and a^{-t-1} if q is even).

Theorem 4.22 *Construction* 4.21 *is an* $(s(m+1)+2, s(m+1))$ *MDS code.*

Proof: For the two elements in columns $i^{(t_1)}, i^{(t_2)}$ and row r, $t_1 \neq t_2$, we can see that they are

in the same row set and the same zigzag set. The corresponding two equations from the two parities are

$$\begin{bmatrix} 1 & 1 \\ \beta_{r,i}^{t_1} & \beta_{r,i}^{t_2} \end{bmatrix} \begin{bmatrix} a_{r,i}^{t_1} \\ a_{r,i}^{t_2} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{4.18}$$

Where y_1, y_2 are easily calculated from the surviving information in the system. Therefore the value of $a_{r,i}^{t_1}, a_{r,i}^{t_2}$ can be computed iff

$$\beta_{r,i}^{(t_1)} \neq \beta_{r,i}^{(t_2)}, \tag{4.19}$$

which is satisfied by the construction. Similarly to (4.5), the four elements in columns $i^{(t_1)}, j^{(t_2)}$ and rows $r, r' = r + e_i + e_j$, $0 \leq t_1, t_2 \leq s-1$, $0 \leq i < j \leq m$ can be rebuilt and therefore the code is MDS if

$$\beta_{r,i}^{(t_1)} \beta_{r',i}^{(t_1)} \neq \beta_{r,j}^{(t_2)} \beta_{r',j}^{(t_2)}. \tag{4.20}$$

By the remark after Theorem 4.6, we know that $\beta_{r,i}^{(t_1)} \neq \beta_{r',i}^{(t_1)}$, and $\beta_{r,j}^{(t_2)} = \beta_{r',j}^{(t_2)}$. Hence the left hand side of (4.20)

$$\beta_{r,i}^{(t_1)} \beta_{r',i}^{(t_1)} = a^{t_1+1_q} a^{(t_1+1)(1-2\cdot 1_q)} = a^{(2t_1+1)(1-1_q)}.$$

For even q the right-hand side of (4.20) equals to a^{2x}, for some $x \neq 0$ and (4.20) is satisfied. Similarly for odd q (4.20) is satisfied. Hence, the construction is MDS. □

Remark: For two identical permutations $f_i^{(t_1)} = f_i^{(t_2)}$, (4.19) is a necessary and sufficient condition for a code to be able to correct the two-column erasure: columns $i^{(t_1)}$ and $i^{(t_2)}$.

Theorem 4.23 *For an MDS s-duplication code, we need a finite field $\mathbb{F}_q$ of size $q \geq s+1$. Therefore, Theorem* 4.22 *is optimal for odd q.*

Proof: Consider the two information elements in row i and columns $j^{(t_1)}, j^{(t_2)}$, which are in the same row and zigzag sets, for $t_1 \neq t_2 \in [0, s-1]$. The code is MDS only if

$$\begin{bmatrix} \alpha_{i,j}^{(t_1)} & \alpha_{i,j}^{(t_2)} \\ \beta_{i,j}^{(t_1)} & \beta_{i,j}^{(t_2)} \end{bmatrix}$$

has full rank. All the coefficients are nonzero. Thus, $(\alpha_{i,j}^{(t_1)})^{-1}\beta_{i,j}^{(t_1)} \neq (\alpha_{i,j}^{(t_2)})^{-1}\beta_{i,j}^{(t_2)}$, and $(\alpha_{i,j}^{(t)})^{-1}\beta_{i,j}^{(t)}$ are distinct nonzero elements in $\mathbb{F}_q$, for $t \in [0, s-1]$. So $q \geq s+1$. □

For instance, the coefficients in Figure 4.6 are assigned as Construction 4.21 and $\mathbb{F}_3$ is used. One can check that any two-column erasure can be rebuilt in this code.

Consider, for example, an s-duplication of the code in Theorem 4.3 with $m = 10$, the array is of size $1024 \times (11s + 2)$. For $s = 2$ and $s = 6$, the ratio is 0.522 and 0.537 by Corollary 4.17. The code has 24 and 68 columns (disks), and the field size needed can be 4 and 8 by Theorem 4.22, respectively. Both of these two sets of parameters are suitable for practical applications.

As mentioned in Theorem 4.18 the optimal construction yields a ratio of $1/2 + 1/2m$ by using duplication of the code in Theorem 4.3. However the field size is a linear function of the number of duplications of the code.

A comparison among the code constructed by the standard basis and zero vector (Theorem 4.3), by duplication of the standard basis and zero vector (Corollary 4.17), and by constant weight vectors (Construction 4.13) is shown in Figure 4.5. We can see that these three constructions provide a tradeoff among the rebuilding ratio, the number of columns, and the field size. If we want to access exactly $1/2$ of the information during the rebuilding process, the standard-basis construction is the only choice. If we are willing to sacrifice the rebuilding ratio for the sake of increasing the number of columns (disks) in the system, the other two codes are good options. Constant weight vectors technique has the advantage of a smaller field size over duplication, e.g., for $O(m^3)$ columns, the field of size 9 and m^2 is needed, respectively. However, duplication provides us a simple technique to have an arbitrary number of columns.

4.5 Generalization of the Code Construction

In this section we generalize Construction 4.1 to an arbitrary number of parity nodes $r = n - k$. We will construct an (n, k) MDS array code, i.e., it can recover from up to r node erasures for arbitrary integers n, k. We will show the code has optimal rebuilding ratio of $1/r$ when a systematic node is erased. When $r = 3$, we will prove that finite-field size of 4 is sufficient for the code to be MDS.

As in the case for 2 parities, a file of size $\mathcal{M}$ is stored in the system, where each node (systematic or parity) stores a file of size $\frac{\mathcal{M}}{k}$. The k systematic nodes are stored in columns $[0, k-1]$. The i-th, $0 \leq i \leq r-1$ parity node is stored in column $k + i$, and is associated with zigzag sets $\{Z_j^i : j \in [0, p-1]\}$, where p is the number of rows in the array.

Construction 4.24 *Let $A = (a_{i,j})$ be the information array of size $r^m \times k$, for some integers k, m. Let $T = \{v_0, ..., v_{k-1}\} \subseteq \mathbb{Z}_r^m$ be a subset of vectors of size k, where for each $v = (v_1, ..., v_m) \in T$,*

$$\gcd(v_1, ..., v_m, r) = 1, \tag{4.21}$$

and gcd *is the greatest common divisor. For any* l, $0 \leq l \leq r-1$, *and* $v \in T$ *we define the permutation* $f_v^l : [0, r^m - 1] \to [0, r^m - 1]$ *by* $f_v^l(x) = x + lv$, *where by abuse of notation we use* $x \in [0, r^m - 1]$ *both to represent the integer and its* r*-ary representation, and all the calculations are done over* $\mathbb{Z}_r$.*For example, for* $m = 2, r = 3, x = 4, l = 2, v = (0,1)$,

$$f_{(0,1)}^2(4) = 4 + 2(0,1) = (1,1) + (0,2) = (1,0) = 3.$$

One can check that the permutation $f_{(0,1)}^2$ *in a vector notation is* $[2,0,1,5,3,4,8,6,7]$. *For simplicity denote the permutation* $f_{v_j}^l$ *as* f_j^l *for* $v_j \in T$. *For* $t \in [0, r^m - 1]$, *we define the zigzag set* Z_t^l *in parity node* l, *as the elements* $a_{i,j}$ *such that their coordinates satisfy* $f_j^l(i) = t$. *In a rebuilding of systematic node* i *the elements in rows* $X_i^l = \{x \in [0, r^m - 1] : x \cdot v_i = r - l\}$ *are rebuilt by parity node* l, $l \in [0, r-1]$, *where the inner product in the definition is done over* $\mathbb{Z}_r$. *From* (4.21) *we get that for any* i *and* l, $|X_i^l| = r^{m-1}$.

Note that similarly to Theorem 4.8, using a large enough field, the parity nodes described above form an (n,k) MDS array code under appropriate selection of coefficients in the linear combinations of the zigzags. We will prove this result formally in Chapter 5 Theorem 5.2.

Assume that the systematic column $i \in [0, k-1]$ was erased, what are the elements to be accessed in the systematic column $j \neq i$ during the rebuilding process? By the construction, the elements of column i and rows X_i^l are rebuilt by the zigzags of parity l. The indices of these zigzags are $f_i^l(X_i^l)$. Therefore we need to access in the surviving systematic columns, all the elements that are contained in these zigzags. Specifically, the elements of systematic column j and rows $f_j^{-l} f_i^l(X_i^l)$ are contained in these zigzags, and therefore need to be accessed. In total, the elements to be accessed in systematic column j are

$$\cup_{l=0}^{r-1} f_j^{-l} f_i^l(X_i^l). \tag{4.22}$$

The following lemma will help us to calculate the size of (4.22), and in particular to calculate the ratio of codes constructed by Construction 4.24.

Lemma 4.25 *For any* $v = (v_1, ...v_m), u \in \mathbb{Z}_r^m$ *and* $l, s \in [0, r-1]$ *such that* $\gcd(v_1, ..., v_m, r) =$

1, *define* $c_{v,u} = v \cdot (v-u) - 1$. *Then*

$$|f_u^{-l} f_v^l(X_v^l) \cap f_u^{-s} f_v^s(X_v^s)| = \begin{cases} |X_v^0|, & (l-s)c_{v,u} = 0 \\ 0, & o.w. \end{cases}$$

In particular for $s = 0$ *we get*

$$|f_u^{-l} f_v^l(X_v^l) \cap X_v^0| = \begin{cases} |X_v^0|, & if\, l c_{v,u} = 0 \\ 0, & o.w. \end{cases}$$

Proof: Consider the group $(\mathbb{Z}_r^m, +)$. Note that $X_v^0 = \{x : x \cdot v = 0\}$ is a subgroup of $\mathbb{Z}_r^m$ and $X_v^l = \{x : x \cdot v = r - l\}$ is a coset. Therefore, $X_v^l = X_v^0 + a_v^l, X_v^s = X_v^0 + a_v^s$, for some $a_v^l \in X_v^l, a_v^s \in X_v^s$. Hence $f_u^{-l} f_v^l(X_v^l) = X_v^0 + a_v^l + l(v-u)$ and $f_u^{-s} f_v^s(X_v^s) = X_v^0 + a_v^s + s(v-u)$ are cosets of X_v^0. So they are either identical or disjoint. Moreover they are identical if and only if

$$a_v^l - a_v^s + (l-s)(v-u) \in X_v^0,$$

i.e., $(a_v^l - a_v^s + (l-s)(v-u)) \cdot v = 0$. But by definition of X_v^l and X_v^s, $a_v^l \cdot v = -l$, $a_v^s \cdot v = -s$, so $(l-s) \cdot c_{v,u} = 0$ and the result follows. □

The following theorem gives the ratio for any code of Construction 4.24.

Theorem 4.26 *The ratio for the code constructed by Construction* 4.24 *and set of vectors* T *is*

$$\frac{\sum_{v \in T} \sum_{u \neq v \in T} \frac{1}{\gcd(r, c_{v,u})} + |T|}{|T|(|T| - 1 + r)},$$

which also equals to

$$\frac{1}{r} + \frac{\sum_{v \in T} \sum_{u \in T, u \neq v} |F_{u,v}(\overline{X_v^0}) \cap \overline{X_v^0}|}{|T|(|T| - 1 + r) r^m}.$$

Where $F_{u,v}(t) = f_u^{-i} f_v^i(t)$ *for* $t \in X_v^i$.

Proof: From any of the r parities, we access r^{m-1} elements during the rebuilding process of node v. Therefore by (4.22), the fraction of the remaining elements to be accessed during the rebuilding is

$$\frac{\sum_{u \neq v \in T} |\cup_{i=0}^{r-1} f_u^{-i} f_v^i(X_v^i)| + r \cdot r^{m-1}}{(|T| - 1 + r) r^m}.$$

Averaging over all the systematic nodes, the ratio is

$$\frac{\sum_{v\in T}\sum_{u\neq v\in T}|\cup_{i=0}^{r-1} f_u^{-i} f_v^i(X_v^i)| + |T|r^m}{|T|(|T|-1+r)r^m}. \tag{4.23}$$

From Lemma 4.25, and noticing that

$$|\{i : ic_{v,u} = 0 \mod r\}| = \gcd(r, c_{v,u}),$$

we get

$$|\cup_{i=0}^{r-1} f_u^{-i} f_v^i(X_v^i)| = r^{m-1} \times r/\gcd(r, c_{v,u}),$$

and the first part follows. For the second part,

$$\begin{aligned}
& \frac{\sum_{v\in T}\sum_{u\neq v\in T}|\cup_{i=0}^{r-1} f_u^{-i} f_v^i(X_v^i)| + |T|r^m}{|T|(|T|-1+r)r^m} \\
= & \frac{\sum_{v\in T}\sum_{u\neq v\in T}|X_v^0| + |\cup_{i=1}^{r-1} f_u^{-i} f_v^i(X_v^i)\backslash X_v^0| + |T|r^m}{|T|(|T|-1+r)r^m} \\
= & \frac{1}{r} + \frac{\sum_{v\in T}\sum_{u\neq v\in T}|\cup_{i=1}^{r-1} f_u^{-i} f_v^i(X_v^i) \cap \overline{X_v^0}|}{|T|(|T|-1+r)r^m} \\
= & \frac{1}{r} + \frac{\sum_{v\in T}\sum_{u\in T, u\neq v}|F_{u,v}(\overline{X_v^0}) \cap \overline{X_v^0}|}{|T|(|T|-1+r)r^m}.
\end{aligned} \tag{4.24}$$

□

Note that the elements in rows X_v^0 of any of the surviving systematic columns are accessed, in order to rebuild the elements of column v which are rebuilt by parity 0. $F_{u,v}(\overline{X_v^0})$ are elements to be accessed in column u in order to rebuild the elements of column v which are rebuilt by parities $1, ..., r-1$. Therefore $F_{u,v}(\overline{X_v^0}) \cap \overline{X_v^0}$ are the extra elements to be accessed in column u for rebuilding column v excluding X_v^0. In order to get a low rebuilding ratio, we need to minimize the amount of these extra elements, i.e., the second term in (4.24). We say that a family of permutation sets $\{\{f_0^l\}_{l=0}^{r-1}, ..., \{f_{k-1}^l\}_{l=0}^{r-1}\}$ together with sets $\{\{X_0^l\}_{l=0}^{r-1}, ..., \{X_{k-1}^l\}_{l=0}^{r-1}\}$ is a family of *orthogonal permutations* if for any $i, j \in [0, k-1]$ the set $\{X_i^l\}_{i=0}^{r-1}$ is an equally sized partition of $[0, r^m - 1]$ and

$$\frac{|F_{j,i}(\overline{X_i^0}) \cap \overline{X_i^0}|}{r^{m-1}(r-1)} = \delta_{i,j}.$$

One can check that for $r = 2$ the definition coincides with the previous definition of orthogonal permutations for two parities. It can be shown that the above definition is equivalent to that for any

$0 \leq i \neq j \leq k-1, 0 \leq l \leq r-1$,

$$f_j^l(X_i^0) = f_i^l(X_i^l). \tag{4.25}$$

For a set of orthogonal permutations, the rebuilding ratio is $1/r$ by (4.24), which is optimal according to (4.1).

Now we are ready to construct a code with optimal rebuilding ratio and r parities.

Theorem 4.27 *The set* $\{\{f_0^l\}_{l=0}^{r-1}, ..., \{f_m^l\}_{l=0}^{r-1}\}$ *together with set* $\{\{X_0^l\}_{l=0}^{r-1}, ..., \{X_m^l\}_{l=0}^{r-1}\}$ *constructed by the vectors* $\{e_i\}_{i=0}^m$ *and Construction* 4.24*, where* X_0^l *is modified to be* $X_0^l = \{x \in \mathbb{Z}_r^m : x \cdot (1,1,...,1) = l\}$ *for any* $l \in [0, r-1]$*, is a family of orthogonal permutations. Moreover the corresponding* $(m+1+r, m+1)$ *code has* optimal *ratio of* $\frac{1}{r}$.

Proof: For $1 \leq i \neq j \leq m$, $c_{i,j} = e_i \cdot (e_i - e_j) - 1 = 0$, hence by Lemma 4.25 for any $l \in [0, r-1]$

$$f_j^{-l} f_i^l(X_i^l) \cap X_i^0 = X_i^0,$$

and (4.25) is satisfied. For $1 \leq i \leq m$, and all $0 \leq l \leq r-1$,

$$\begin{aligned} f_0^{-l} f_i^l(X_i^l) &= f_i^l(\{v : v_i = -l\}) = \{v + le_i : v_i = -l\} \\ &= \{v : v_i = 0\} = X_i^0 \end{aligned}$$

Therefore, $f_0^{-l} f_i^l(X_i^l) \cap X_i^0 = X_i^0$, and (4.25) is satisfied. Similarly,

$$\begin{aligned} f_i^{-l} f_0^l(X_0^l) &= f_i^{-l}(\{v : v \cdot (1,...,1) = l\}) \\ &= \{v - le_i : v \cdot (1,...,1) = l\} \\ &= \{v : v \cdot (1,...,1) = 0\} = X_0^0. \end{aligned}$$

Hence again (4.25) is satisfied and this is a family of orthogonal permutations, and the result follows.

□

Surprisingly, one can infer from the above theorem that changing the number of parities from 2 to 3 adds only one node to the system, but reduces the rebuilding ratio from $1/2$ to $1/3$ in the rebuilding of any systematic column.

The example in Figure 4.8 shows a code with 3 systematic nodes and 3 parity nodes constructed by Theorem 4.27 with $m = 2$. The code has an optimal ratio of $1/3$. For instance, if column C_1 is erased, accessing rows $\{0,1,2\}$ in the remaining nodes will be sufficient for rebuilding.

Permutations of parity node C_4

f_0^1	f_1^1	f_2^1
0	3	1
1	4	2
2	5	0
3	6	4
4	7	5
5	8	3
6	0	7
7	1	8
8	2	6

Permutations of parity node C_5

f_0^2	f_1^2	f_2^2
0	6	2
1	7	0
2	8	1
3	0	5
4	1	3
5	2	4
6	3	8
7	4	6
8	5	7

Encoding by the orthogonal permutations →

Systematic Nodes			Parity Nodes		
C_0	C_1	C_2	C_3	C_4	C_5
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,0}+a_{0,1}+a_{0,2}$	$ca_{0,0}+a_{6,1}+a_{2,2}$	$c^2a_{0,0}+a_{3,1}+a_{1,2}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,0}+a_{1,1}+a_{1,2}$	$ca_{1,0}+a_{7,1}+ca_{0,2}$	$c^2a_{1,0}+a_{4,1}+ca_{2,2}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,0}+a_{2,1}+a_{2,2}$	$ca_{2,0}+a_{8,1}+a_{1,2}$	$c^2a_{2,0}+a_{5,1}+ca_{0,2}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,0}+a_{3,1}+a_{3,2}$	$ca_{3,0}+ca_{0,1}+ca_{5,2}$	$c^2a_{3,0}+ca_{6,1}+ca_{4,2}$
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$	$a_{4,0}+a_{4,1}+a_{4,2}$	$ca_{4,0}+ca_{1,1}+a_{3,2}$	$c^2a_{4,0}+ca_{7,1}+ca_{5,2}$
$a_{5,0}$	$a_{5,1}$	$a_{5,2}$	$a_{5,0}+a_{5,1}+a_{5,2}$	$ca_{5,0}+ca_{2,1}+a_{4,2}$	$c^2a_{5,0}+ca_{8,1}+a_{3,2}$
$a_{6,0}$	$a_{6,1}$	$a_{6,2}$	$a_{6,0}+a_{6,1}+a_{6,2}$	$ca_{6,0}+a_{3,1}+a_{8,2}$	$c^2a_{6,0}+ca_{0,1}+ca_{7,2}$
$a_{7,0}$	$a_{7,1}$	$a_{7,2}$	$a_{7,0}+a_{7,1}+a_{7,2}$	$ca_{7,0}+a_{4,1}+a_{6,2}$	$c^2a_{7,0}+ca_{1,1}+a_{8,2}$
$a_{8,0}$	$a_{8,1}$	$a_{8,2}$	$a_{8,0}+a_{8,1}+a_{8,2}$	$ca_{8,0}+a_{5,1}+ca_{7,2}$	$c^2a_{8,0}+ca_{2,1}+ca_{6,2}$

Figure 4.8: A $(6,3)$ MDS array code with optimal ratio $1/3$. The first parity C_3 corresponds to the row sums, and the corresponding identity permutations are omitted. The second and third parity C_4, C_5 are generated by the permutations f_i^1, f_i^2, respectively, $i = 0,1,2$. The elements are from $\mathbb{F}_4$, where c is a primitive element of $\mathbb{F}_4$.

Similar to the 2 parity case, the following theorem shows that Theorem 4.27 achieves the optimal number of columns. In other words, the number of rows has to be exponential in the number of columns in any systematic MDS code with optimal ratio, optimal update, and r parities. This follows since any such optimal code is constructed from a family of orthogonal permutations.

Theorem 4.28 *Let* $\{\{f_0^l\}_{l=0}^{r-1}, ..., \{f_{k-1}^l\}_{l=0}^{r-1}\}$ *be a family of orthogonal permutations over the integers* $[0, r^m - 1]$ *together with the sets* $\{\{X_0^l\}_{l=0}^{r-1}, ..., \{X_{k-1}^l\}_{l=0}^{r-1}\}$, *then* $k \leq m+1$.

Proof: We prove it by induction on m. When $m = 0$, it is trivial that $k \leq 1$. Now suppose we have a family of orthogonal permutations $\{\{f_0^l\}_{l=0}^{r-1}, ..., \{f_{k-1}^l\}_{l=0}^{r-1}\}$ over $[0, r^m - 1]$, and we will show $k \leq m+1$. Recall that orthogonality is equivalent to (4.25). Notice that for any permutations $g, h_0, ..., h_{r-1}$, the sets of permutations $\{\{h_l f_0^l g\}_{l=0}^{r-1}, ..., \{h_l f_{k-1}^l g\}_{l=0}^{r-1}\}$ are still a family of orthogonal permutations with sets $\{\{g^{-1}(X_0^l)\}, ..., \{g^{-1}(X_{k-1}^l)\}\}$. This is because

$$\begin{aligned} h_l f_j^l g(g^{-1}(X_i^0)) &= h_l f_j^l(X_i^0) \\ &= h_l f_i^l(X_i^l) \\ &= h_l f_i^l g(g^{-1}(X_i^l)). \end{aligned}$$

Therefore, w.l.o.g. we can assume $X_0^l = [lr^{m-1}, (l+1)r^{m-1} - 1]$, and f_0^l is the identity permuta-

tion, for $0 \leq l \leq r-1$.

Let $1 \leq i \neq j \leq k-1, l \in [0, r-1]$ and define

$$\begin{aligned} A &= f_j^l(X_i^0) = f_i^l(X_i^l), \\ B &= f_j^l(X_i^0 \cap X_0^0), \\ C &= f_i^l(X_i^l \cap X_0^0). \end{aligned}$$

Therefore B, C are subsets of A, and their complements in A are

$$\begin{aligned} A \backslash B &= f_j^l(X_i^0 \cap \overline{X_0^0}), \\ A \backslash C &= f_i^l(X_i^l \cap \overline{X_0^0}). \end{aligned}$$

From (4.25) for any $j \neq 0$,

$$f_j^l(X_0^0) = f_0^l(X_0^l) = X_0^l \tag{4.26}$$

hence,

$$B, C \subseteq X_0^l \tag{4.27}$$

Similarly, for any $j \neq 0$, $f_j^l(\overline{X_0^0}) = \overline{f_j^l(X_0^0)} = \overline{X_0^l}$, hence

$$A \backslash B, A \backslash C \subseteq \overline{X_0^l}. \tag{4.28}$$

From (4.27),(4.28) we conclude that $B = C = A \cap X_0^l$, i.e.,

$$f_j^l(X_i^0 \cap X_0^0) = f_i^l(X_i^l \cap X_0^0). \tag{4.29}$$

For each $l \in [0, r-1], j \in [1, k-1]$ define $\hat{f}_j^l(x) = f_j^l(x) - lr^{m-1}$ and $\hat{X}_j^l = X_j^l \cap X_0^0$ then,

$$\begin{aligned} \hat{f}_j^l([0, r^{m-1}-1]) &= f_j^l(X_0^0) - lr^{m-1} \\ &= X_0^l - lr^{m-1} \\ &= [0, r^{m-1}-1], \end{aligned} \tag{4.30}$$

where (4.30) follows from (4.26). Moreover, since f_i^l is bijective we conclude that $\hat{f}_i^l$ is a permuta-

tion on $[0, r^{m-1} - 1]$.

$$\begin{aligned} \hat{f}_i^l(\hat{X}_i^l) &= f_i^l(X_i^l \cap X_0^0) - lr^{m-1} \\ &= f_j^l(X_i^0 \cap X_0^0) - lr^{m-1} \\ &= \hat{f}_j^l(\hat{X}_i^0), \end{aligned} \tag{4.31}$$

where (4.31) follows from (4.29). Since $\{X_i^l\}_{l=0}^{r-1}$ is a partition of $[0, r^m - 1]$, then $\{\hat{X}_i^l\}_{l=0}^{r-1}$ is also a partition of $X_0^0 = [0, r^{m-1} - 1]$. Moreover, since $\hat{f}_i^l(\hat{X}_i^l) = \hat{f}_j^l(\hat{X}_i^0)$ for any $l \in [0, r-1]$, and $\hat{f}_i^l, \hat{f}_j^l$ are bijections, we conclude

$$|\hat{X}_i^l| = |\hat{X}_i^0|$$

for all $l \in [0, r-1]$, i.e., $\{\hat{X}_i^l\}$, $l \in [0, r-1]$, is an equally sized partition of $[0, r^{m-1} - 1]$. Therefore $\{\{\hat{f}_1^l\}_{l=0}^{r-1}, ..., \{\hat{f}_{k-1}^l\}_{l=0}^{r-1}\}$ together with $\{\{\hat{X}_1^l\}_{l=0}^{r-1}, ..., \{\hat{X}_{k-1}^l\}_{l=0}^{r-1}\}$ is a family of orthogonal permutations over integers $[0, r^{m-1} - 1]$, hence by induction $k - 1 \leq m$, and the result follows. □

After presenting the construction of a code with optimal ratio of $1/r$, we move on to discuss the problem of assigning the proper coefficients in order to satisfy the MDS property. This task turns out to be not easy when the number of parities $r > 2$. The next theorem gives a proper assignment for the code with $r = 3$ parities, constructed by the optimal construction in Theorem 4.27.

Construction 4.29 *Let c be a primitive element of $\mathbb{F}_4$ and define $P_j = (p_{i,l})$ to be the permutation matrix corresponding to the permutation $f_j = f_j^1$, with a slight modification. This matrix has two nonzero values and is defined as*

$$p_{i,l} = \begin{cases} 1, & l + e_j = i \text{ and } l \cdot \sum_{t=1}^{j} e_t \neq 0 \\ c, & l + e_j = i \text{ and } l \cdot \sum_{t=1}^{j} e_t = 0 \\ 0, & \text{o.w.} \end{cases} \tag{4.32}$$

For example, if $m = 2$ the permutation matrix is of size $3^2 = 9$, and we get the matrix $P_2 =$

$(p_{i,l})_{0\le i,l\le 8}$,

$$P_2 = \begin{bmatrix} & & 1 & & & & & & \\ c & & & & & & & & \\ & 1 & & & & & & & \\ & & & & c & & & & \\ & & & 1 & & & & & \\ & & & & 1 & & & & \\ & & & & & & & 1 & \\ & & & & & & 1 & & \\ & & & & & & & c & \end{bmatrix}.$$

For example, $p_{3,5} = c$*, since*

1. $5 + e_2 = (1,2) + (0,1) = (1,0) = 3$.
2. $5 \cdot \sum_{t=1}^{2} e_t = (1,2) \cdot (1,1) = 0$.

Let the generator matrix of the code be

$$G' = \begin{bmatrix} I & & \\ & \ddots & \\ & & I \\ I & \cdots & I \\ P_0 & \cdots & P_m \\ P_0^2 & \cdots & P_m^2 \end{bmatrix}_{(m+4)\times m+1}. \tag{4.33}$$

Here each block matrix is of size $3^m \times 3^m$*, and* P_j^2 *represents the square of the matrix* P_j*.*

Theorem 4.30 *When* $r = 3$*, a field of size* 4 *is sufficient to make the code MDS using Construction* 4.29.

For example, the coefficients of the parities in Figure 4.8 are assigned as Construction 4.29. One can check that the system is protected against any three erasures.

The key idea of the proof is that if three erasures happen, we do not try to solve for all of the unknown elements at the same time, but utilize the special structure of the permutations and solve a few linear equations at a time. No matter which columns are erased, we can always rearrange the

ordering of the unknown elements and the equations, such that the coefficient matrix of the linear equations has a common format. Therefore, as long as this format is an invertible matrix, we know the code is MDS.

Proof: [Proof of Theorem 4.8] We need to show we can rebuild three erasures, with x erasures of systematic nodes, and $3-x$ of the parities, for $x = 1,2,3$. It is easy to see that when c is a nonzero coefficient, we can rebuild from one systematic and two parity erasures.

In case of two systematic erasures, suppose information columns i, j and parity column 2 are erased, $0 \leq i < j \leq k-1$. We will show that instead of solving equations involving all the unknown elements, we only need to solve 6 linear equations at a time. In order to recover the elements in row v, consider the set of rows in the erased columns:

$$W(v) = v + \text{span}\{e_i - e_j\}.$$

We call v a *starting point*. $W(v)$ contains 3 elements and altogether there are 6 unknown elements in the two columns i, j. Notice that elements in rows $W(v)$ and column i, j are mapped to elements in rows $W(v)$ and parity 0. Also for parity 1 they are mapped to rows $W(v) + e_i = W(v) + e_j$, which are equal because they are both cosets of $\text{span}\{e_i - e_j\}$ and $v + e_i$ is a member in both cosets. Therefore, by accessing rows $W(v)$ in the surviving information nodes and parity 0, and rows $W(v) + e_j$ in parity 1, we get 6 equations on these 6 unknowns.

For example, in Figure 4.8 columns C_1, C_2, C_5 are erased, then $i = 1, j = 2$. And consider the starting point $v = (1,0)$, which is 3 as an integer. Then $W(v) = v + \text{span}\{e_2 - e_1\} = \{(1,0),(2,2),(0,1)\}$, or $\{3,8,1\}$ written as integers. Similarly, $W(v) + e_j = W(v) + e_i = \{4,6,2\}$ as integers. The 6 elements in rows $W(v)$ in columns C_1, C_2 are $\{a_{1,1}, a_{1,2}, a_{3,1}, a_{3,2}, a_{8,1}, a_{8,2}\}$. They are mapped to rows $W(v)$ in parity 0 (column C_4) and to rows $W(v) + e_j$ in parity 1 (column C_5). Therefore, we can solve for the 6 unknowns at a time.

Writing in matrix form, we need to solve the linear equations $Gx = y$, where x is the 6×1 unknown vector, y is a vector of size 6×1, and G is a 6×6 matrix. G can be written as

$$\begin{array}{l} \\ \text{parity } 0, W(v) \\ \text{parity } 1, W(v) + e_j \end{array} \begin{array}{c} \begin{array}{cc} \text{info } i, W(v) & \text{info } j, W(v) \end{array} \\ \begin{pmatrix} I & I \\ A & B \end{pmatrix} \end{array},$$

and each submatrix here is of size 3×3. The first 3 columns in G correspond to column i, the last

3 columns correspond to column j. The first 3 rows in G correspond to parity 0, the last 3 rows correspond to parity 1. We wrote the corresponding column and row indices at the top and on the left of the matrix. Since parity 0 is the row sum, the first 3 rows of G are two 3×3 identity matrices.

Now we reorder the row and columns of G and show that $\det(G) = \det(B - A) \neq 0$. For $t = 0, 1$, order the elements in cosets $W(v) + te_j$ as $(v + te_j, (v + te_j) + (e_i - e_j), (v + te_j) + 2(e_i - e_j)$. What are A and B? For row $u \in W(v)$ in column i, it is mapped to row $u + e_i = (u + e_j) + (e_i - e_j)$ in parity 1. So A corresponds to a cyclic shift permutation. Suppose $u = v + g(e_i - e_j)$, $g = 0, 1, 2$, then the coefficient is determined by

$$u \sum_{t=1}^{i} e_t = v \sum_{t=1}^{i} e_t + g.$$

According to (4.32), the coefficient is c if $u \sum_{t=1}^{i} e_t = 0$, and is 1 otherwise. For only one value of $g \in [0, 2]$, the above expression is 0. Therefore we have

$$A = \begin{bmatrix} & & a_1 \\ a_2 & & \\ & a_3 & \end{bmatrix}$$

with $a_1 a_2 a_3 = c$. Similarly, row u in column j is mapped to $u + e_j$ in parity 1. So B corresponds to diagonal matrix. And the coefficient is determined by

$$u \sum_{t=1}^{j} e_t = v \sum_{t=1}^{j} e_t + g - g = v \sum_{t=1}^{i} e_t,$$

which is a constant for $W(v)$. Hence

$$B = \begin{bmatrix} b & & \\ & b & \\ & & b \end{bmatrix}$$

with $b = 1$ or c. Now

$$\det(G) = \det(B - A) = \det \begin{bmatrix} b & & -a_1 \\ -a_2 & b & \\ & -a_3 & b \end{bmatrix}$$

$$= b^3 - a_1a_2a_3 = b^3 - c. \tag{4.34}$$

The above value is $1 - c$ or $c^3 - c$. If $c \neq 0$, and $c^2 \neq 1$, then $\det(G) \neq 0$. When c is a primitive element in $GF(4)$, the above conditions are satisfied.

For example, if in Figure 4.8 we erase columns C_1, C_2, C_6 and take the starting point $v = (1, 0)$, then $W(v)$ is ordered as $(3, 8, 1)$ and $W(v) + e_j$ is ordered as $(4, 6, 2)$. It is easy to check that

$$A = \begin{bmatrix} & & c \\ 1 & & \\ & 1 & \end{bmatrix}$$

and

$$B = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}.$$

Similarly, if column i, j and parity 0 are erased, we can show

$$\det(G) = \det(B - A)AB \neq 0.$$

When column i, j and parity 1 are erased, we have

$$\begin{aligned} \det(G) &= \det(B^2 - A^2) \\ &= \det \begin{bmatrix} b^2 & -a_1a_3 & \\ & b^2 & -a_1a_2 \\ -a_2a_3 & & b^2 \end{bmatrix} \\ &= b^6 - (a_1a_2a_3)^2 = b^6 - c^2. \end{aligned}$$

When $b = 1$ or c, the above value is $1 - c^2$ or $c^6 - c^2$. So we need $c \neq 0, c^2 \neq 1, c^4 \neq 1$. Again, for a finite field of size 4, these conditions are satisfied. Hence we can rebuild any two systematic and one parity erasures.

Suppose 3 systematic columns i, j, l are erased, and $1 \leq i < j < l \leq k$. We will show that each time we need to solve 27 equations, and then reduce it to the case of two systematic erasures. In order to rebuild any row v in these three columns, consider the following set of 9 rows (and therefore

27 unknown elements):

$$V = v + \text{span}\{e_i - e_l, e_j - e_l\}.$$

These unknowns correspond to rows V in parity 0. In parity 1, they correspond to rows $V + e_i = V + e_j = V + e_l$, which are equal to each other since they are cosets of $\text{span}\{e_i - e_l, e_j - e_l\}$ and $v + e_l$ is a member in all of them. Similarly, the unknowns correspond to rows $V + 2e_i = V + 2e_j = V + 2e_l$ in parity 2. Altogether we have 27 parity elements (equations). Next we write these equations in a matrix form:

$$Gx = y,$$

where G, y are coefficients, x are unknowns. We are going to show that $\det(G) \neq 0$. Order the set $\text{span}\{e_i - e_l, e_j - e_l\}$ arbitrarily as $(v_0, v_1, \ldots, v_8)$. And order the coset $V + te_l$ as $(v + te_l + v_0, v + te_l + v_1, \ldots, v + te_l + v_8)$, for $t = 0, 1, 2$. Now the coefficient matrix G will be

$$\begin{array}{l} \\ \text{parity } 0, V \\ \text{parity } 1, V + e_l \\ \text{parity } 2, V + 2e_l \end{array} \begin{array}{c} \begin{array}{ccc} \text{info } i, V & \text{info } j, V & \text{info } l, V \end{array} \\ \begin{pmatrix} I & I & I \\ A' & B' & C' \\ A'^2 & B'^2 & C'^2 \end{pmatrix} \end{array},$$

where each sub-block is of size 9×9. The first, second, and third block rows correspond to parity 0,1, and 2, respectively. And the first, second and third block columns correspond to erased column i, j, l, respectively. Since parity 0 is row sum, the first block rows contain identity submatrices. What is C' for parity 1? By Construction 4.24, row u in column l corresponds to row $u + e_l$ in parity 1. So C' should be diagonal. By (4.32) the values in C are determined by $u \cdot \sum_{t=1}^{l} e_t$. And for some constants g, h, we have $u = v + g(e_i - e_l) + h(e_j - e_l) \in V$, and thus $u \cdot \sum_{t=1}^{l} e_t = (v + g(e_i - e_l) + h(e_j - e_l)) \sum_{t=1}^{l} e_t = v \sum_{t=1}^{l} e_t + g - g + h - h = v \sum_{t=1}^{l} e_t$ is a constant for V. So

$$C' = I \text{ or } cI,$$

for a primitive element c. Now notice that C' is commutative with A' and B', we have $\det(G) = \det(B' - A') \det(C' - A') \det(C' - B')$ (without commutativity, this equation may not hold).

Moreover, since V is the union of $W(v), W(v+e_l-e_i), W(v+2(e_l-e_i))$, and

$$\det(B'-A') = \det \begin{bmatrix} I & I \\ A' & B' \end{bmatrix},$$

we know that $\det(B'-A')$ is simply the multiplication of three determinants in (4.34) with starting point $v, v+e_l-e_i, v+2(e_l-e_i)$, which are always nonzero. Similarly, we can conclude that $\det(C'-A'), \det(C'-B')$ are also nonzero. Hence the code an correct any three erasures and is an MDS code.

□

4.6 Concluding Remarks

In this chapter, we described explicit constructions of the first known systematic (n,k) MDS array codes with $n-k$ equal to some constant, such that the amount of information needed to rebuild an erased column equals to $1/(n-k)$, matching the information-theoretic lower bound. While the codes are new and interesting from a theoretical perspective, they also provide an exciting practical solution, specifically, when $n-k=2$, our zigzag codes are the best known alternative to RAID-6 schemes. RAID-6 is the most prominent scheme in storage systems for combating disk failures. Our new zigzag codes provide a RAID-6 scheme that has optimal update (important for write efficiency), small finite-field size (important for computational efficiency) and optimal access of information for rebuilding - cutting the current rebuilding time by a factor of two.

We note that one can add redundancy for the sake of lowering the rebuilding ratio. For instance, one can use three parity nodes instead of two. The idea is that the third parity is not used for protecting data from erasures, since in practice, three concurrent failures are unlikely. However, with three parity nodes, we are able to rebuild a single failed node by accessing only $1/3$ of the remaining information (instead of $1/2$). An open problem is to construct codes that can be extended in a simple way, namely, codes with three parity nodes such that the first two nodes ensure a rebuilding ratio of $1/2$ and the third node further lowers the ratio to $1/3$. Hence, we can first construct an array with two parity nodes and when needed, extend the array by adding an additional parity node to obtain additional improvement in the rebuilding ratio.

Another future research direction is to consider the ratio of read accesses in the case of a write (update) operation. For example, in an array code with two parity nodes, in order to update a single

information element, one needs to read at least three elements and write three elements, because we need to know the values of the old information and old parities and compute the new parity elements (by subtracting the old information from the parity and adding the new information). However, an interesting observation, in our optimal code construction with two parity nodes, is if we update all the information in the first column and the rows in the top half of the array (see Figure 4.4), we do not need to read for computing the new parities, because we know the values of all the information elements needed for computing the parities. These information elements take about half the size of the entire array. So in a storage system we can cache the information to be written until most of these elements need to be updated (we could arrange the information in a way that these elements are often updated at the same time), hence, the ratio between the number of read operations and the number of new information elements is relatively very small. Clearly, we can use a similar approach for any other systematic column. In general, given r parity nodes, we can avoid redundant read operations if we update about $1/r$ of the array.

Chapter 5

Rebuilding Any Single-Node Erasure

5.1 Introduction

In this chapter, we define the *rebuilding ratio* as the ratio of accessed information to the remaining information in case of *any* single erasure. Different from the previous two chapters, we consider systematic as well as parity erasure. For example, it is easy to check that for the code in Figure 5.1, if any two columns are erased, we can still recover all the information, namely, it is an MDS code. Here all elements are in finite field F_3. Now suppose column C_1 is erased, it can be rebuilt by accessing $a_{0,2}, a_{1,2}$ from column C_2, r_0, r_1 from column C_3, and z_0, z_1 from column C_4, as follows:

$$\begin{aligned} a_{0,1} &= r_0 - a_{0,2} = 2a_{0,2} + r_0 \\ a_{1,1} &= 2a_{1,2} + r_1 \\ a_{2,1} &= 2a_{1,2} + z_0 \\ a_{3,1} &= a_{0,2} + z_1 \end{aligned}$$

Hence, by accessing only 6 elements out of 12 remaining elements, i.e., only half of the remaining information, the erased node can be rebuilt. Similarly, if column C_2 is erased, only half elements need to be accessed. However, if column C_3 or C_4 is erased, one has to access all elements in column C_1, C_2, a total of 8 elements, in order to rebuild. Details on this code will be discussed in Section 5.2.

As mentioned before, when a single erasure occurs and all the remaining nodes are accessible, the lower bound for the repair bandwidth, and therefore the rebuilding ratio, is $1/r$ [DGW$^+$10]. In the previous chapter we presented an explicit construction of MDS array codes that achieve the lower bound $1/r$ on the ratio for rebuilding any *systematic node*.

Systematic nodes | Parity nodes

C_1	C_2	C_3	C_4
$a_{0,1}$	$a_{0,2}$	$r_0 = a_{0,1} + a_{0,2}$	$z_0 = a_{2,1} + a_{1,2}$
$a_{1,1}$	$a_{1,2}$	$r_1 = a_{1,1} + a_{1,2}$	$z_1 = a_{3,1} + 2a_{0,2}$
$a_{2,1}$	$a_{2,2}$	$r_2 = a_{2,1} + a_{2,2}$	$z_2 = 2a_{0,1} + 2a_{3,2}$
$a_{3,1}$	$a_{3,2}$	$r_3 = a_{3,1} + a_{3,2}$	$z_3 = 2a_{1,1} + a_{2,2}$

Figure 5.1: An MDS array code with two systematic and two parity nodes. All the elements are in finite field F_3. The first parity column C_3 is the row sum and the second parity column C_4 is generated by the zigzags. For example, zigzag z_0 contains the elements $a_{i,j}$ that satisfy $f_j^1(i) = 0$.

The main result of this chapter is an explicit construction of MDS array codes with r parity nodes, that achieves the lower bound $1/r$ for rebuilding *any systematic or parity node*. The rebuilding of a single erasure has an efficient implementation as computations within nodes are not required. Moreover, our codes have simple encoding and decoding procedures - when $r = 2$ and $r = 3$, the codes require finite-field sizes of 3 and 4, respectively.

We would like to point out here that the constructed code achieves optimal ratio in the cost of update complexity. An MDS code with r parities is called *optimal update* if each information element is contained in exactly r parity elements. If we update the value of an information element, we only need to change the value of r parity elements. And this is the minimum number of changes required for an MDS code. For example, in Figure 5.1 the information element $a_{0,1}$ is contained in only $r = 2$ parity elements: r_0, z_2. While the construction in last chapter is optimal update, the code in this chapter is not. Each information element is contained in $2r - 1$ parity elements.

5.2 Rebuilding Ratio Problem

In this section we formally define the rebuilding ratio problem and review the code construction in the previous chapter, which has optimal rebuilding for systematic erasure. We then show that the construction can be made an MDS code, in fact, this will be the basis for proving that our newly proposed construction described in Section 5.3 is also an MDS code.

We first define the framework of a systematic MDS array code. Let $A = (a_{i,j})$ be an information array of size $p \times q$. A column is also called a node, and an entry is called an element. Each of the q columns is a systematic node in the code. We add r parity columns to this array on the right, such that from any q columns, we can recover the entire information. In the previous chapter, it was shown that if the code has *optimal update*, i.e., each information element is protected by exactly r parity elements, then each parity node corresponds to q permutations acting on $[0, p-1]$. More specifically, suppose the permutations are $f_1, f_2, \ldots, f_q$. Then the t-th element in this parity node is a linear combination of all elements $a_{i,j}$ such that $f_j(i) = t$. The set of information elements contained in this linear combination is called a *zigzag set*. For the t-th element in the l-th parity, $t \in [0, p-1], l \in [0, r-1]$, denote by $f_1^l, \ldots, f_q^l$ the set of associated permutations, and Z_t^l the zigzag set.

Because the ordering of the elements in each node can be arbitrary, we can assume that the first parity node is always a linear combination of each row (corresponding to identity permutations). If we write a permutation in the vector notation, we have

$$f_1^0 = f_2^0 = \cdots = f_q^0 = (0, 1, \ldots, p-1).$$

Figure 5.1 is an example of such codes. The first parity C_3 corresponds to identity permutations, or sum of each row. The second parity C_4 corresponds to the permutations

$$f_1^1 = (2, 3, 0, 1),$$
$$f_2^1 = (1, 0, 3, 2).$$

For instance, assume $t = 0$. Since $f_1^1(2) = 0, f_2^1(1) = 0$, the zigzag set $Z_0^1 = \{a_{2,1}, a_{1,2}\}$, and z_0 is a linear combination of these two elements.

For a given MDS code with parameters q, r, we ask what is the accessed fraction in order to rebuild a single node (in the average case)? Hence, the *rebuilding ratio* of a code is:

$$R = \frac{\sum_{i=1}^{q+r} (\#\text{ accessed elements to rebuild node } i)}{(q+r)(\#\text{ remaining elements})}.$$

Notice that the ratio is averaged among all of the single-node erasures.

When a systematic node is erased, we assume that each unknown element is rebuilt by one of the parity nodes. That is, we access one parity element containing the unknown, and access all

the elements in the corresponding zigzag set except the unknown. In order to lower the number of accesses, we would like to find

1. Good permutations such that the accessed zigzag sets intersect as much as possible.

2. Proper coefficients in the linear combinations such that the code is MDS.

For example, in Figure 5.1, in order to rebuild column C_1, we access the zigzag sets $A = \{Z_0^0, Z_1^0\}$, $B = \{Z_0^1, Z_1^1\}$, corresponding to parities $\{r_0, r_1\}, \{z_0, z_1\}$. Since the surviving elements in A and in B are both $\{a_{0,2}, a_{1,2}\}$, they are identical and have maximal intersection. As a result, only $1/2$ of the elements are accessed. Besides, the coefficients $\{1, 2\}$ in the parity linear combinations guarantee that any two nodes are sufficient to recover all the information. Hence the code is MDS.

Now we repeat the result of Construction 4.24. We formed permutations based on r-ary vectors. Let $e_1, e_2, \ldots, e_k$ be the standard vector basis of $\mathbb{Z}_r^k$. We will use x to represent both an integer in $[0, r^k - 1]$ and its r-ary expansion (the r-ary vector of length k). It will be clear from the context which meaning is used. All the calculations are done over $\mathbb{Z}_r$.

Construction 5.1 *Let the information array be of size $r^k \times k$. Define permutation f_j^l on $[0, r^k - 1]$ as $f_j^l(x) = x + le_j$, for $j \in [1, k], l \in [0, r-1]$. For example, if $r = 2, k = 2, j = 2, l = 1, x = 3$, then $f_2^1(3) = 3 + 1 \cdot e_2 = (1,1) + (0,1) = (1,0) = 2$. And $f_2^1 = (1,0,3,2)$ is the entire permutation. For $t \in [0, r^k - 1]$, we define the zigzag set Z_t^l in parity node l as the elements $a_{i,j}$ such that their coordinates satisfy $f_j^l(i) = t$. Let $Y_j = \{x \in [0, r^k - 1] : x \cdot e_j = 0\}$ be the set of vectors whose j-th coordinate is 0. If column j is erased, rebuild by accessing rows Y_j in all the remaining columns.*

We point here that the above construction does not use the zero vector as Construction 4.24. Notice that $|Y_j| = r^k/r$ is only $1/r$ of the remaining elements. The previous chapter tells us that Y_j is sufficient to rebuild node j and therefore the construction has optimal ratio $1/r$ for any systematic node.

Figure 5.1 is an example of Construction 5.1 with $k = 2, r = 2$. As mentioned before, only $1/2$ of the information is accessed in order to rebuild C_1. The accessed elements are in rows $Y_1 = \{x \in [0,3] : x \cdot e_1 = 0\} = \{0,1\}$.

Next, we show that by assigning the coefficients in the parities properly, we can make the code MDS. Let $P_j = (p_{i,l})$ be the permutation matrix corresponding to $f_j = f_j^1$, namely, $p_{i,l} = 1$ if $l + e_j = i$, and $p_{i,l} = 0$ otherwise. Assigning the coefficients is the same as modifying $p_{i,l} = 1$ to

some other non-zero value. When $r \geq 4$, modify all $p_{i,l} = 1$ to $p_{i,l} = \lambda_j$, for some λ_j in a finite field F. Let the generator matrix of the code be

$$G' = \begin{bmatrix} I & & \\ & \ddots & \\ & & I \\ I & \cdots & I \\ P_1^1 & \cdots & P_k^1 \\ \vdots & & \vdots \\ P_1^{r-1} & \cdots & P_k^{r-1} \end{bmatrix}_{(k+r)\times k}. \tag{5.1}$$

Here each submatrix is of size $r^k \times r^k$.

When $r = 2, 3$, modify $p_{i,l} = 1$ to

$$p_{i,l} = c, \text{ if } l \cdot \sum_{t=1}^{j} e_t = 0, \tag{5.2}$$

where c is a primitive element of F_3, F_4, respectively. And keep $p_{i,l} = 1$ if $\sum_{t=1}^{j} e_t \neq 0$. And the generator matrix is also G' in (5.1). For example, the coefficients in Figure 5.1 is assigned according to (5.2), with

$$P_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}, P_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The following theorem shows that using the above assignment the code can be MDS.

Theorem 5.2 *(1) Construction* 5.1 *can be made an MDS code for a large enough finite field. (2) When $r = 2, 3$, field of size* 3 *and* 4 *is sufficient to make the code MDS.*

Proof: Part (2) are shown in Theorems 4.6 and 4.30. We only prove part (1). An MDS code means that it can recover any r erasures. Suppose t systematic nodes and $r - t$ parity nodes are erased, $1 \leq t \leq r$. Thus suppose we delete from G' in (5.1) the systematic rows $\{j_1, j_2, \ldots, j_t\}$ and the remaining parity nodes are $\{i_1, i_2, \ldots, i_t\}$. Then the following $t \times t$ block matrix should be

invertible:

$$G = \begin{bmatrix} P_{j_1}^{i_1} & \cdots & P_{j_t}^{i_1} \\ \vdots & & \vdots \\ P_{j_1}^{i_t} & \cdots & P_{j_t}^{i_t} \end{bmatrix} \tag{5.3}$$

Its determinant $\det(G)$ is a polynomial with unknowns $\lambda_{j_1}, \ldots, \lambda_{j_t}$. All terms have highest degree $r^k(i_1 + \cdots + i_t)$. One term with highest degree is $\prod_{s=1}^{t} \lambda_{j_s}^{i_s r^k}$ with non-zero coefficient 1 or -1. So $\det(G)$ is a non-zero polynomial. Up to now we only showed one possible case of erasures. For any r erasures, we can find the corresponding $\det(G)$ as a non-zero polynomial. The product of all these polynomials is again a non-zero polynomial. Hence by [Alo99] for a large enough field there exist assignments of $\{\lambda_j\}$ such that the value of the polynomial is not 0. Then for any case of r erasures, the corresponding matrix G is invertible, and the code is MDS. □

5.3 Code Construction

The code in Construction 4.24 has optimal rebuilding for systematic nodes. However, in order to rebuild a parity node, one has to access all the information elements. In this section we construct MDS array codes with optimal rebuilding ratio for rebuilding both the systematic and the parity nodes. The code has $k-1$ systematic nodes and r parities nodes, for any k, r.

Consider the permutation $f_j = f_j^1$ in Construction 5.1. It is clear that f_j is a permutation of order r, i.e., f_i^r is the identity permutation. For $i \in [0, r-1]$, define X_i as the set of vectors of weight i, namely, $X_i = \{v \in \mathbb{Z}_r^k : v \cdot (1, \ldots, 1) = i\}$. X_0 is a subgroup of $\mathbb{Z}_r^k$ and $X_i = X_0 + ie_k$ is its coset, where $e_k = (0, \ldots, 0, 1)$. Assume the elements in X_i are ordered, $i \in [0, r-1]$, and the ordering is

$$X_0 = (v_1, \ldots, v_{r^{k-1}}),$$
$$X_i = (v_1 + ie_k, \ldots, v_{r^{k-1}} + ie_k).$$

Since the ordering of the elements in each node does not matter, we can reorder them as $(X_0, X_1, \ldots, X_{r-1})$, with each X_i ordered as above. We are going to write our generator matrix of the code in this new ordering. One can check that $f_j(X_i) = X_{i+1}$, where the subscript is added mod r. So the

matrix P_j corresponding to f_j can be written as

$$P_j = \begin{array}{c} \\ X_0 \\ X_1 \\ \vdots \\ X_{r-1} \end{array} \begin{array}{c} \begin{array}{cccc} X_0 & X_1 & \dots & X_{r-1} \end{array} \\ \begin{pmatrix} & & & p_j \\ p_j & & & \\ & \ddots & & \\ & & p_j & \end{pmatrix} \end{array}, \tag{5.4}$$

where p_j corresponds to the mapping of $f_j : X_i \mapsto X_{i+1}$. In particular, if p_j is viewed as a permutation acting on X_0, then for $x \in X_0$,

$$p_j(x) = x + e_j - e_k.$$

Next we assign coefficients in P_j. When $r = 2, 3$, modify the 1 entries of p_i into c if its corresponding column l satisfies $l \cdot \sum_{t=1}^{j} e_t = 0$. Here c is a primitive element in F_3, F_4. When $r \geq 4$, modify 1 entries into λ_j.

In the following, we will use block matrices the same as single elements. When referring to row or column indices, we mean block row or column indices. We refer to p_j as a small block, and the corresponding block row or column as a small block row or column. And P_j is called a big block with big block row or column. Moreover, we assume the elements in each column are in order $(X_0, \dots, X_{r-1})$.

Construction 5.3 *Suppose the information array is of size $r^k \times (k-1)$. For $j \in [1, k-1]$, define a big block matrix*

$$A_j^0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ \vdots \\ r-2 \\ r-1 \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & & & & \end{array} \\ \begin{pmatrix} I & & & & & & \\ p_j & \alpha p_j^{r-1} & & & & & \\ p_j^2 & & \alpha p_j^{r-2} & & & & \\ \vdots & & & & \ddots & & \\ p_j^{r-2} & & & & & p_j^2 & \\ p_j^{r-1} & & & & & & p_j \end{pmatrix} \end{array}$$

where $\alpha \neq 0, 1$ is an element of the finite field and is multiplied to the diagonal in rows $1, \dots, \lfloor \frac{r}{2} \rfloor$. And define A_j^i by cyclically shifting the rows and columns of A_j^0 to the right and bottom by i posi-

$$A^0 = \begin{array}{|c|c|} \hline \underline{I} & \\ \hline \underline{p} & 2p \\ \hline \end{array} \qquad A^1 = \begin{array}{|c|c|} \hline \underline{p} & p \\ \hline & I \\ \hline \end{array}$$

$$A^0 = \begin{array}{|c|c|c|} \hline \underline{I} & & \\ \hline \underline{p} & \alpha p^2 & \\ \hline \underline{p^2} & & p \\ \hline \end{array} \qquad A^1 = \begin{array}{|c|c|c|} \hline \underline{p} & p^2 & \\ \hline & I & \\ \hline & p & \alpha p^2 \\ \hline \end{array} \qquad A^2 = \begin{array}{|c|c|c|} \hline \underline{\alpha p^2} & & p \\ \hline & p & p^2 \\ \hline & & I \\ \hline \end{array}$$

Figure 5.2: Parity matrices A^i for $r = 2$ (left) and $r = 3$ (right) parities. When the first parity node is erased, the underlined elements are accessed from systematic nodes. The remaining unknown elements are recovered by the shaded elements from parity nodes.

tions:

$$A_j^i = \begin{bmatrix} \beta p_j^i & & & p_j^{r-i} & & \\ & \ddots & & \vdots & & \\ & & p_j & p_j^{r-1} & & \\ & & & I & & \\ & & & p_j & \alpha p_j^{r-1} & \\ & & & \vdots & & \ddots \end{bmatrix},$$

where $\beta = \alpha$ or 1. If $x - i < \frac{r}{2}$ or $x - i = \frac{r}{2}, i < \frac{r}{2}$, coefficient α is multiplied to the diagonal in row x. Construct the code as follows. Let the first $k-1$ nodes be systematic, and the last r nodes be parities. Parity i is defined by $A_1^i, \ldots, A_{k-1}^i$. The generator matrix of the code is

$$\begin{bmatrix} I & & \\ & \ddots & \\ & & I \\ A_1^0 & \cdots & A_{k-1}^0 \\ \vdots & & \vdots \\ A_1^{r-1} & \cdots & A_{k-1}^{r-1} \end{bmatrix}.$$

$a_{0,1}$	$a_{0,2}$	$a_{0,1}+a_{0,2}$	$a_{5,1}+a_{4,1}+a_{3,2}+a_{2,2}$
$a_{1,1}$	$a_{1,2}$	$a_{5,1}+2a_{4,1}+a_{3,2}+2a_{2,2}$	$a_{1,1}+a_{1,2}$
$a_{2,1}$	$a_{2,2}$	$a_{6,1}+2a_{7,1}+2a_{0,2}+a_{1,2}$	$a_{2,1}+a_{2,2}$
$a_{3,1}$	$a_{3,2}$	$a_{3,1}+a_{3,2}$	$a_{6,1}+a_{7,1}+2a_{0,2}+2a_{1,2}$
$a_{4,1}$	$a_{4,2}$	$2a_{0,1}+a_{1,1}+2a_{6,2}+a_{7,2}$	$a_{4,1}+a_{4,2}$
$a_{5,1}$	$a_{5,2}$	$a_{5,1}+a_{5,2}$	$2a_{0,1}+2a_{1,1}+2a_{6,2}+2a_{7,2}$
$a_{6,1}$	$a_{6,2}$	$a_{6,1}+a_{6,2}$	$2a_{3,1}+2a_{2,1}+a_{5,2}+a_{4,2}$
$a_{7,1}$	$a_{7,2}$	$2a_{3,1}+a_{2,1}+a_{5,2}+2a_{4,2}$	$a_{7,1}+a_{7,2}$

Figure 5.3: An MDS array code with two systematic and two parity nodes by Construction 5.3. The finite field used is F_3. The shaded elements are accessed to rebuild the first parity node.

Sometimes we will omit the subscript j when it is not important, and the superscript is computed mod r.

Example 5.4 *For two and three parities, the matrices A^i are shown in Figure* 5.2. *When $r=2$, as finite field F_3 is used, we can take $\alpha=2\neq 1$. Coefficient $\alpha=2$ is multiplied to only the second diagonal in A^0. When $r=3$, finite field F_4 is used and we choose some $\alpha\neq 0,1$. We multiply α to one diagonal block in each A^i. It can be seen that A^1, A^2 are simply shifted versions of A^0. An example of a code with* 2 *parities is shown in Figure* 5.3.

It can be seen from Construction 5.3 and Figure 5.3 that this code is not optimal update. In fact each information element appears $2r-1$ times in the parities.

Next we show that the code in Construction 5.3 has optimal ratio. We first observe that in A^i, the x-th row ($x\neq i$) is

$$\begin{array}{c} \\ x \text{ in } A^i \end{array}\begin{array}{c} \begin{array}{cccccc} & & i & & x & \\ \end{array} \\ \begin{pmatrix} \cdots & p^{x-i} & \cdots & \beta p^{i-x} & \cdots \end{pmatrix}, \end{array}$$

where the values above are the column indices and omitted blocks are all zero. Here $\beta=\alpha$ if $x-i<\frac{r}{2}$ or $x-i=\frac{r}{2}, i<\frac{r}{2}$, and $\beta=1$ otherwise. Therefore, suppose $i'-i<\frac{r}{2}$ or $i'-i=\frac{r}{2}, i<\frac{r}{2}$, then the i'-th small block row in A^i and the i-th small block row in $A^{i'}$ are the same except for the coefficients:

$$\begin{array}{c} \\ i' \text{ in } A^i \\ i \text{ in } A^{i'} \end{array}\begin{array}{c} \begin{array}{cccccc} & & i & & i' & \\ \end{array} \\ \begin{pmatrix} \cdots & p^{i'-i} & \cdots & \alpha p^{i-i'} & \cdots \\ \cdots & p^{i'-i} & \cdots & p^{i-i'} & \cdots \end{pmatrix}. \end{array} \tag{5.5}$$

Theorem 5.5 *The code has optimal ratio* $1/r$ *for rebuilding any node. More specifically, when the systematic node* e_i *is erased,* $i \in [1, k-1]$*, we only need to access rows* $Y_i = \{v \in \mathbb{Z}_r^k : v \cdot e_i = 0\}$*. When the parity* i *is erased,* $i \in [0, r-1]$*, we only need to access rows* $X_i = \{v \in \mathbb{Z}_r^k : v \cdot (1, 1, \cdots, 1) = i\}$.

Proof: **Systematic rebuilding:** W.l.o.g. assume column e_1 is erased. Access equations $Y = \{v \in \mathbb{Z}_r^k : v \cdot e_1 = 0\}$ from each parity. We will show that all the unknowns $(x_0, \ldots, x_{r^k-1})$ in column e_1 are solvable from these equations. First notice that Y is a subgroup of $\mathbb{Z}_r^k$, and $Y - te_k = Y$ for any $t \in [0, r-1]$. So any index in $\mathbb{Z}_r^k$ can be written as one of the following three cases:

$$l, l - te_1, l + t(e_1 - e_k),$$

where $l \in Y$ and $1 \leq t \leq \lfloor \frac{r}{2} \rfloor$. So we need to show that an unknown element indexed by these three cases is solvable.

For any $l \in Y$, assume $l \in Y \cap X_{i'}$ for some i'. Then x_l is contained in equation

$$x_l$$

because of the i'-th small row block $[\cdots I \cdots]$ in $A_1^{i'}$. Notice that $l + (i - i')e_k \in Y \cap X_i$ for all $i \in [0, r-1]$. In (5.5) consider row l in A^i and row $l + (i - i')e_k$ in $A^{i'}$, and write $t = i' - i \leq \lfloor \frac{r}{2} \rfloor$. Then we have equations

$$\begin{aligned} bx_{l-te_1} + \alpha c x_{l+t(e_1-e_k)} &= g \\ bx_{l-te_1} + cx_{l+t(e_1-e_k)} &= h \end{aligned}$$

for some coefficients $\alpha \neq 0, 1$, $b, c \neq 0$ and g, h. These equations are obviously independent. Hence all unknowns are solvable.

Next we show that the fraction of elements accessed in the remaining columns is $1/r$. For a parity node A^i, only rows Y are accessed, which is a fraction of $1/r$. The corresponding columns in A^i of theses equations are accessed from the systematic nodes. For a surviving systematic node $j \in [2, k-1]$ and parity i, by definition of p_j^i, rows Y in A_j^i are mapped to columns $Y' = Y + i(e_k - e_j) + se_k$ for some s. However, Y' is a coset of Y and since $i(e_k - e_j) + se_k \in Y$, we have $Y' = Y$. Thus only elements with indices Y are accessed from each node.

Parity rebuilding: Since the parities are all symmetric, w.l.o.g. suppose the 0-th parity is erased. Access X_0 from each node, which is the set of vectors of weight 0. Need to show this is sufficient to recover

$$A = [A_1^0, A_2^0, \ldots, A_{k-1}^0],$$

where A_j^0 is defined in Construction 5.3. Since X_0 is sent from the systematic nodes, the 0-th column in each A_j^0 is known, and we can remove them from the equations. By (5.5), from parity i' we can access row

$$[\underline{\beta' p_1^{i'}} \cdots p_1^{-i'} \cdots \underline{\beta' p_2^{i'}} \cdots p_2^{-i'} \cdots\cdots \underline{\beta' p_k^{i'}} \cdots p_k^{-i'} \cdots],$$

where the underlined elements are known from the systematic nodes and can be treated as 0. Here β' is 1 or α. Multiplying this row by β, we can rebuild the i'-th row of A:

$$[\underline{p_1^{i'}} \cdots \beta p_1^{-i'} \cdots \underline{p_2^{i'}} \cdots \beta p_2^{-i'} \cdots\cdots \underline{p_k^{i'}} \cdots \beta p_k^{-i'} \cdots],$$

where $\beta\beta' = \alpha$, and again the underlined elements are known and treated as 0. So far we have rebuilt row i' in A, with $i' = 1, 2, \ldots, r-1$. The 0-th row in A is

$$[\underline{I} \cdots \underline{I} \cdots\cdots \underline{I} \cdots]$$

and can be rebuilt from the systematic nodes directly. Thus the erased node is rebuilt by accessing X_0, which is $1/r$ of the elements. □

It can be seen from the above proof that the rebuilding of any single erasure can be easily implemented. If a systematic node is eared, we only need to solve at most two linear equations at a time, and the computation can be done in parallel. If a parity is erased the rebuilding is even simpler: we only need to subtract/add information elements, and multiply by β. Also, the rebuilding above is different from Construction 4.24, where only one linear equation is solved at a time.

Example 5.6 *Consider the code with two or three parities in Figure* 5.2*. When the first parity node is erased, one can access X_0 from the systematic nodes, and the underlined elements are known. Then access the shaded elements from the surviving parity nodes. It is easy to see that the first parity can be rebuilt from the accessed elements.*

For the specific example of Figure 5.3*, when the first systematic node is erased, one can access*

rows $Y_1 = \{v : v \cdot e_1 = 0\} = \{0, 1, 2, 3\}$ *from all the surviving nodes. When the first parity node is erased, one can access rows* $X_0 = \{0, 3, 5, 6\}$ *from all the remaining nodes (the shaded elements). Then it is easy to check that in both cases it is sufficient to rebuild the erased column.*

Next we show the construction is indeed an MDS code. We prove this by reducing this problem to the fact that Construction 5.1 is MDS. First we make an observation on the small blocks.

Lemma 5.7 *Construction* 5.1 *is MDS iff any* $t \times t$ *sub-block matrix of*

$$H' = \begin{bmatrix} p_1^0 & \cdots & p_k^0 \\ \vdots & & \vdots \\ p_1^{r-1} & \cdots & p_k^{r-1} \end{bmatrix}_{k \times k}$$

is invertible, for all $t \in [1, r]$.

Proof: First define a $t \times t$ sub-block matrix of H':

$$H = \begin{bmatrix} p_1^0 & \cdots & p_t^0 \\ \vdots & & \vdots \\ p_1^{t-1} & \cdots & p_t^{t-1} \end{bmatrix}.$$

We showed in the appendix that Construction 5.1 is MDS iff any G in (5.3) is invertible. W.l.o.g.

suppose $\{i_1, \ldots, i_t\} = \{0, \ldots, t-1\}, \{j_1, \ldots, j_t\} = \{1, \ldots, t\}$. By (5.4), G can be rewritten as

$$G = \left[\begin{array}{ccc|ccc|c} I & & & I & & & \\ & I & & & I & & \cdots \\ & & \ddots & & & \ddots & \\ & & I & & & I & \\ \hline & & p_1 & & & p_2 & \\ p_1 & & & p_2 & & & \cdots \\ & \ddots & & & \ddots & & \\ & & p_1 & & & p_2 & \\ \hline & & p_1^2 & & & p_2^2 & \\ & & & p_1^2 & & & p_2^2 \quad \cdots \\ p_1^2 & & & p_2^2 & & & \\ & \ddots & & & \ddots & & \\ \hline & \vdots & & & \vdots & & \end{array}\right],$$

where each big block is composed of $r \times r$ small blocks. We can see that the shaded small blocks are the only non-zero blocks in their corresponding rows and columns, and they form the submatrix H. Therefore G being invertible is equivalent to H and the remaining submatrix both being invertible. Moreover the remaining submatrix has a similar form as G and we can again find t rows and t columns corresponding to H. Continue this we get

$$\det(G) \neq 0 \Leftrightarrow (\det(H))^r \neq 0 \Leftrightarrow \det(H) \neq 0.$$

The same conclusion holds for any submatrix of H'. Thus completes the proof. □

The method of taking out sub-block matrices to compute the determinant as above is also used in the proof of the following theorem, which shows that Construction 5.3 is indeed an MDS code.

Theorem 5.8 *If the coefficients in the linear combinations of the parities are chosen such that Construction* 5.1 *is MDS, then Construction* 5.3 *is also MDS.*

Proof: Similar to the proof of Theorem 5.2 in the appendix, Construction 5.3 being MDS

means any of the following matrix is invertible:

$$A = \begin{bmatrix} A_{j_1}^{i_1} & \cdots & A_{j_t}^{i_1} \\ \vdots & & \vdots \\ A_{j_1}^{i_t} & \cdots & A_{j_t}^{i_t} \end{bmatrix}_{t\times t},$$

where each submatrix is of size $r \times r$ and $t \in [1,r], I = \{i_1, \ldots, i_t\} \subseteq [0, r-1], \{j_1, \ldots, j_t\} \subseteq [1, k-1]$. Let the complement of I be $\overline{I} = [0, r-1] \backslash I$. In each big block consider the small block column $x \in \overline{I}$. Only small block rows x in each big block are non-zero. Thus we can take out this $t \times t$ sub-block matrix:

$$\begin{bmatrix} \beta_1 p_{j_1}^{i_1 - x} & \cdots & \beta_1 p_{j_t}^{i_1 - x} \\ \vdots & & \vdots \\ \beta_t p_{j_1}^{i_t - x} & \cdots & \beta_t p_{j_t}^{i_t - x} \end{bmatrix},$$

where $\{\beta_i\}$ are 1 or α. But by Lemma 5.7, the above matrix is invertible. So we only need to look at the remaining submatrix. Again, we can take out another small block column and row from $\overline{I}$ from each big block, and it is invertible by Lemma 5.7. Continue this process, we are left with only columns and rows of I in each big block. For all $i, i' \in I, 1 \leq i' - i < \frac{r}{2}$ or $i' - i = \frac{r}{2}, i < \frac{r}{2}$, consider row i' in A^i and row i in $A^{i'}$. They are shown in (5.5). One can do row operations and keep the invertibility of the matrix, and get

$$\begin{array}{c} \\ i' \text{ in } A^i \\ i \text{ in } A^{i'} \end{array} \begin{array}{c} \begin{array}{ccccccccc} i & & i' & & i & & i' & & \end{array} \\ \begin{pmatrix} \cdots & 0 & \cdots & p_{j_1}^{i-i'} & \cdots & 0 & \cdots & p_{j_t}^{i-i'} & \cdots \\ \cdots & p_{j_1}^{i'-i} & \cdots & 0 & \cdots & p_{j_t}^{i'-i} & \cdots & 0 & \cdots \end{pmatrix}. \end{array}$$

Proceed this for all $i, i' \in I$, we are left with block diagonal matrix in each big block and the matrix left is of size $t^2 \times t^2$. Taking out the i_1-th column and row in each big block, we have the following $t \times t$ submatrix:

$$\begin{bmatrix} p_{j_1}^0 & \cdots & p_{j_t}^0 \\ p_{j_1}^{i_2 - i_1} & \cdots & p_{j_t}^{i_2 - i_1} \\ \vdots & & \vdots \\ p_{j_1}^{i_t - i_1} & \cdots & p_{j_t}^{i_t - i_1} \end{bmatrix},$$

which is invertible by Lemma 5.7. Similarly, we can take out the i_2-th column and row, and so on, and each submatrix is again invertible. Thus, any matrix A is invertible and Construction 5.3 is

MDS. □

For example, one can easily check that the code in Figure 5.3 is able to recover the information from any two nodes. Therefore it is an MDS code.

Theorem 5.8 says that once we have an MDS code in Construction 5.1, we can use its coefficients and design a new code by Construction 5.3. And the new code is guaranteed to be an MDS code.

5.4 Summary

In this chapter, we presented constructions of MDS array codes that achieve the optimal rebuilding ratio $1/r$, where r is the number of redundancy nodes. The new codes are constructed based on our previous Construction 4.24 and improve the efficiency of the rebuilding access.

Now we mention a couple of open problems. First, in our construction each information element is contained in $2r-1$ parity elements. This means if we update this element, we need to update $2r-1$ times in the parities. But an optimal-update code will require only r updates. So are there codes that achieve optimal rebuilding ratio and also optimal update?

Besides, if there are $k-1$ systematic nodes and r parity nodes, then our code has r^k rows. Namely, the code length is limited. Given the number of rows, are there codes that are longer? For example, when $r=2$, we know a construction with r^k rows and k systematic nodes:

$$A_j^0 = \begin{bmatrix} I & 0 \\ p_j & I \end{bmatrix}, A_j^1 = \begin{bmatrix} I & p_j \\ 0 & I \end{bmatrix}.$$

Here A_j^0, A_j^1 are the matrices that generate the parities, and we can take all $j \in [1,k]$. This code has one more information column than Construction 5.3, and also achieves optimal ratio. On the other hand, however, given r^k rows, it can be proven that any systematic and linear code with optimal ratio has no more than $k+1$ systematic nodes. Thus the code length $k-1$ can be improved by at most 2 nodes.

Finally, using Construction 4.24 one is able to rebuild any $e, 1 \leq e \leq r$, *systematic* erasures with an access ratio of e/r. However, it is an open problem to construct a code that can rebuild any e erasures with optimal access.

Chapter 6

Rebuilding Multiple Failures

6.1 Introduction

So far we have discussed codes with optimal rebuilding for a single systematic node erasure in Chapter 4, for any single-node erasure in Chapter 5. In this chapter, we discuss the rebuilding for other types of failures.

For a zigzag code with two parities, as the minimum distance is 3, the code is able to correct one column erasure, two-column erasures, or any single column error. Column erasures happen when a node is dysfunctional, not connected, or when errors in the node are treated as an erasure. We will present decoding algorithm for erasures. On the other hand, errors may happen in a solid state drive, or when errors are not treated as node erasures in a storage system. We assume in this case that a column error corresponds to an arbitrary number of symbol errors in one column. By computing syndromes one can locate and correct such errors.

In some applications such as solid state drives, one may encounter entire node erasure as well as some bit flips, or single bit or symbol errors. In the presence of an error, rebuilding another node erasure becomes more complicated: one needs to gather information and decide the error location, error value, as well as to compute the lost node values. Therefore, we will study the capability of the zigzag code in terms of correcting errors and erasures. In particular, we focus on the codes with two parities. It is surprising that even though the minimum distance of this code is only 3, it still corrects a node erasure plus some symbol errors, which in general would require larger minimum distance.

Even though single-node erasure is the most common scenario in distributed or centralized storage, it is possible to find two or more erasures at the same time as the storage system scales. If the code has only two parities, one needs to read all the remaining file. However, we also consider

the following generalization: Suppose that we have an MDS code with three parity nodes, if we have a single erasure, using our codes, we can rebuild the erasure with rebuilding ratio of $1/3$. What happens if we have two erasures? What is the rebuilding ratio in this case? We will show our zigzag codes can achieve the optimal rebuilding ratio of $2/3$. In general, if we have $r \geq 3$ parity nodes and e erasures happen, $1 \leq e \leq r$, we will prove that the lower bound of repair bandwidth is e/r (normalized by the size of the remaining array), and so is the rebuilding ratio. And the code we constructed achieves this lower bound for any e.

6.2 Decoding of the Codes

In this section, we will discuss decoding algorithms of the zigzag codes in case of column erasures as well as a column error. The algorithms work for both Construction 4.1 and its duplication code.

Let $\mathcal{C}$ be a $(k+2,k)$ MDS array code defined by Construction 4.1 (and possibly duplication). The code has array size $2^m \times (k+2)$. Let the zigzag permutations be f_j, $j \in [0,k-1]$, which are not necessarily distinct. Let the information elements be $a_{i,j}$, and the row and zigzag parity elements be r_i and z_i, respectively, for $i \in [0,2^m-1], j \in [0,k-1]$. W.l.o.g. assume the row coefficients are $\alpha_{i,j} = 1$ for all i,j. And let the zigzag coefficients be $\beta_{i,j}$ in some finite field $\mathbb{F}$.

For convenience we rewrite Construction 4.1 here.

Construction 6.1 (Construction 4.1) *Let A be the information array of size $2^m \times k$. Let $T = \{v_0, v_1, \ldots, v_{k-1}\} \subseteq \mathbb{F}_2^m$ be a set of vectors of size k. For $v \in T$, we define the permutation $f_v : [0,2^m-1] \to [0,2^m-1]$ by $f_v(x) = x + v$. Construct the two parities as row and zigzag parities.*

The following is a summary of the erasure decoding algorithms.

Algorithm 6.2 *(Erasure Decoding)*

One erasure.

1) One parity node is erased. Rebuild the row parity by

$$r_i = \sum_{j=0}^{k-1} a_{i,j}, \tag{6.1}$$

and the zigzag parity by

$$z_i = \sum_{j=0}^{k-1} \beta_{f_j^{-1}(i),j} a_{f_j^{-1}(i),j}. \tag{6.2}$$

2) One information node j is erased. Rebuild the elements in rows X_j by rows, and those in rows $\overline{X_j}$ by zigzags. Here $X_v = \{x \in [0, 2^{m-1}] : x \cdot v = 0\}$.

Two erasures.

1) Two parity nodes are erased. Rebuild by (6.1) *and* (6.2).

2) One parity node and one information node is erased. If the row parity node is erased, rebuild by zigzags; otherwise rebuild by rows.

3) Two information nodes j_1 and j_2 are erased.

- If $f_{j_1} = f_{j_2}$, for any $i \in [0, 2^m - 1]$, compute

$$\begin{array}{rcl} x_i & = & r_i - \sum_{j \neq j_1, j_2} a_{i,j} \\ y_i & = & z_{f_{j_1}(i)} - \sum_{j \neq j_1, j_2} \beta_{f_j^{-1} f_{j_1}(i), j} a_{f_j^{-1} f_{j_1}(i), j}. \end{array} \tag{6.3}$$

Solve a_{i,j_1}, a_{i,j_2} from the equations

$$\begin{bmatrix} 1 & 1 \\ \beta_{i,j_1} & \beta_{i,j_2} \end{bmatrix} \begin{bmatrix} a_{i,j_1} \\ a_{i,j_2} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

- Else, for any $i \in [0, 2^m - 1]$, set $i' = i + f_{j_1}(0) + f_{j_2}(0)$, and compute $x_i, x_{i'}, y_i, y_{i'}$ according to (6.3). *Then solve $a_{i,j_1}, a_{i,j_2}, a_{i',j_1}, a_{i',j_2}$ from equations*

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ \beta_{i,j_1} & 0 & 0 & \beta_{i',j_2} \\ 0 & \beta_{i,j_2} & \beta_{i',j_1} & 0 \end{bmatrix} \begin{bmatrix} a_{i,j_1} \\ a_{i,j_2} \\ a_{i',j_1} \\ a_{i',j_2} \end{bmatrix} = \begin{bmatrix} x_i \\ x_{i'} \\ y_i \\ y_{i'} \end{bmatrix}.$$

In case of a column error, we first compute the syndrome, then locate the error position, and at last correct the error. Let $x_0, x_1, \ldots, x_{p-1} \in \mathbb{F}$. Denote $f^{-1}(x_0, x_1, \ldots, x_{p-1}) = (x_{f^{-1}(0)}, x_{f^{-1}(1)}, \ldots, x_{f^{-1}(p-1)})$ for a permutation f on $[0, p-1]$. The detailed algorithm is as follows.

Algorithm 6.3 *(Error Decoding)*

Compute for all $i \in [0, 2^m - 1]$*:*

$$\begin{aligned} s_{i,0} &= \sum_{j=0}^{k-1} a_{i,j} - r_i \\ s_{i,1} &= \sum_{j=0}^{k-1} \beta_{f_j^{-1}(i),j} a_{f_j^{-1}(i),j} - z_i. \end{aligned}$$

Let the syndrome be $S_0 = (s_{0,0}, s_{1,0}, \ldots, s_{2^m-1,0})$ *and* $S_1 = (s_{0,1}, s_{1,1}, \ldots, s_{2^m-1,1})$.

- If $S_0 = 0$ *and* $S_1 = 0$*, there is no error.*

- Else if one of S_0, S_1 *is* 0*, there is an error in the parity. Correct it by* (6.1) *or* (6.2)*.*

- Else, find the error location. For $j = 0$ *to* $k - 1$*:*

Compute for all $i \in [0, 2^m - 1]$, $x_{i,j} = \beta_{i,j} s_{i,0}$.

Let $X_j = (x_{0,j}, \ldots, x_{2^m-1,j})$ *and* $Y_j = f_j^{-1}(X_j)$.

If $Y_j = S_1$*, subtract* S_0 *from column* j*. Stop.*

If no such j *is found, there are more than one error.*

If there is only one error, the above algorithm is guaranteed to find the error location and correct it, since the code is MDS, as the following theorem states.

Theorem 6.4 *Algorithm* 6.3 *can correct one column error.*

Proof: Notice that each zigzag permutation f_j is the inverse of itself by Construction 4.1, or $f_j = f_j^{-1}$. Suppose there is error in column j, and the error is $E = (e_0, e_1, \ldots, e_{2^m-1})$. So the received column j is the sum of the original information and E. Thus the syndromes are $s_{i,0} = e_i$ and

$$s_{i,1} = \beta_{f_j(i),j} e_{f_j(i)}.$$

For column t, $t \in [0, k-1]$, we have $x_{i,t} = \beta_{i,t} s_{i,0} = \beta_{i,t} e_i$. Write $Y_t = f_j^{-1}(X_j) = (y_{0,t}, \ldots, y_{2^m-1,t})$ and then

$$y_{i,t} = x_{f_t(i),t} = \beta_{f_t(i),t} e_{f_t(i)}.$$

We will show the algorithm finds $Y_t = S_1$ iff $t = j$, and therefore subtracting $S_0 = E$ from column j will correct the error. When $t = j$, $y_{i,t} = s_{i,1}$, for all $i \in [0, 2^m - 1]$, so $Y_j = S_1$. Now suppose there is $t \neq j$ such that $Y_t = S_1$. Since the error E is nonzero, there exists i such that $e_{f_j(i)} \neq 0$.

Consider the indices i and $i' = f_t f_j(i)$. $y_{i,t} = s_{i,1}$ yields

$$\beta_{f_t(i),t} e_{f_t(i)} = \beta_{f_j(i),j} e_{f_j(i)}. \tag{6.4}$$

Case 1: When $f_t = f_j$, set $r = f_t(i) = f_j(i)$, then (6.4) becomes $\beta_{r,t} e_r = \beta_{r,j} e_r$ with $e_r \neq 0$. Hence $\beta_{r,t} = \beta_{r,j}$ which contradicts (4.19).

Case 2: When $f_t \neq f_j$, since f_t, f_j are commutative and are inverse of themselves, $f_t(i') = f_t f_t f_j(i) = f_j(i)$ and $f_j(i') = f_j f_t f_j(i) = f_t(i)$. Therefore $y_{i',t} = s_{i',1}$ yields

$$\beta_{f_j(i),t} e_{f_j(i)} = \beta_{f_t(i),j} e_{f_t(i)}. \tag{6.5}$$

The two equations (6.4) (6.5) have nonzero solution $(e_{f_j(i)}, e_{f_t(i)})$ iff

$$\beta_{f_t(i),t} \beta_{f_j(i),t} = \beta_{f_j(i),j} \beta_{f_t(i),j},$$

which contradicts (10.2) with $r = f_t(i), r' = f_j(i)$. Hence the algorithm finds the unique erroneous column. □

If the computations are done in parallel for all $i \in [0, 2^m - 1]$, then Algorithm 6.3 can be done in time $O(k)$. Moreover, since the permutations f_i's only change one bit of a number in $[0, 2^m - 1]$ in the optimal code in Theorem 4.3, the algorithm can be easily implemented.

6.3 Correcting Column Erasure and Element Error

In array codes for storage systems, data is arranged in a 2D array. Each column in the array is typically stored in a separate disk and is called *a node*, and each entry in the array is call ed *an element*. In the conventional error model, disk failures correspond to an erasure or an error of an entire node. Therefore, array codes are usually designed to correct such entire node failures. However, if we consider different applications, such as the case of flash memory as storage nodes, element error is also possible. In other words, we may encounter only a few errors in a column as well as entire node erasures. For an MDS array code with two parities, the minimum Hamming distance is 3, therefore, it is not possible to correct a node erasure and a node error at the same time. However, since zigzag code has very long column lengths, we ask ourselves: is it capable of correcting a node erasure and some element errors?

Given a $(k+2,k)$ zigzag code generated by distinct binary vectors $T = \{v_0, v_1, \ldots, v_{k-1}\}$, the following algorithm corrects a node erasure and an element error. Here we assume that the erasure and error are in different columns, and there is only a single element error in the systematic part of the array. The code has two parities and 2^m rows, and the zigzag permutations are $f_j = v_j, j \in [0, k-1]$. The original array is denoted by $(a_{i,j})$, the erroneous array is $(\hat{a}_{i,j})$. The row coefficients are all ones, and the zigzag coefficients are $\beta_{i,j}$. Let $x_0, x_1, \ldots, x_{p-1} \in \mathbb{F}$. Denote $f^{-1}(x_0, x_1, \ldots, x_{p-1}) = (x_{f^{-1}(0)}, x_{f^{-1}(1)}, \ldots, x_{f^{-1}(p-1)})$ for a permutation f on $[0, p-1]$.

Algorithm 6.5 *Suppose column t is erased, and there is at most one element error in the remaining array. Compute for all $i \in [0, 2^m - 1]$ the syndromes:*

$$s_{i,0} = \sum_{j \neq t} \hat{a}_{i,j} - r_i,$$

$$s_{i,1} = \sum_{j \neq t} \beta_{f_j^{-1}(i),j} \hat{a}_{f_j^{-1}(i),j} - z_i.$$

Let the syndrome be $S_0 = (s_{0,0}, s_{1,0}, \ldots, s_{2^m-1,0})$ and $S_1 = (s_{0,1}, s_{1,1}, \ldots, s_{2^m-1,1})$.
Compute for all $i \in [0, 2^m - 1]$, $x_i = \beta_{i,t} s_{i,0}$. Let $X = (x_0, \ldots, x_{2^m-1})$, $Y = f_t^{-1}(S_1)$, $W = X - Y$.
- If $W = 0$, there is no element error. Assign column t as $-S_0$.
- Else, there will be two rows r, r' such that $w_r, w_{r'}$ are nonzero. Find j such that $v_j = r + r' + v_t$. The error is in column j.

- *If $\frac{w_r}{w_{r'}} = -\frac{\beta_{r,t}}{\beta_{r,j}}$, then the error is at row r, and assign $a_{r,j} = \hat{a}_{r,j} - \frac{W_r}{\beta_{r,t}}$.*
- *Else if $\frac{w_r}{w_{r'}} = -\frac{\beta_{r',j}}{\beta_{r',t}}$, then the error is at row r', and assign $a_{r',j} = \hat{a}_{r',j} - \frac{W_{r'}}{\beta_{r',t}}$.*
- *Else there are more than one errors.*

Theorem 6.6 *The above algorithm can correct a node erasure and a systematic element error.*

Proof: Suppose column t is erased and there is an error at column j and row r. Define $r' = r + v_t + v_j$. Let $\hat{a}_{r,j} = a_{r,j} + e$. It is easy to see that $x_i = y_i = -\beta_{i,t} a_{i,t}$ except when $i = r, r'$. Since the set of binary vectors $\{v_0, v_1, \ldots, v_{k-1}\}$ are distinct, we know that the error is in column j. Moreover,we have

$$x_r = -\beta_{r,t} a_{r,t} + \beta_{r,t} e,$$

$$y_r = -\beta_{r,t} a_{r,t},$$

$$x_{r'} = -\beta_{r',t} a_{r',t},$$

$$y_{r'} = -\beta_{r',t} a_{r',t} + \beta_{r,j} e.$$

Therefore, the difference between X and Y is

$$w_r = x_r - y_r = \beta_{r,t} e,$$

$$w_{r'} = x_{r'} - y_{r'} = -\beta_{r,j} e.$$

And we can see that no matter what e is, we always have

$$\frac{w_r}{w_{r'}} = -\frac{\beta_{r,t}}{\beta_{r,j}}.$$

Similarly, if the error is at row r', we will get

$$\frac{w_r}{w_{r'}} = -\frac{\beta_{r',j}}{\beta_{r',t}}.$$

By the MDS property of the code, we know that $\beta_{r,t}\beta_{r',t} \neq \beta_{r,j}\beta_{r',j}$ (see the remark after the proof of the finite field size 3). Therefore, we can distinguish between the two cases of an error in row r and in row r'. □

Example 6.7 *Consider the zigzag code in Figure* 6.1. *Suppose all of column* 0 *is erased. And suppose there is an error in the 0-th element in column* 1. *Namely, the erroneous symbol we read is* $\hat{b}_0 = b_0 + e$ *for some error* $e \neq 0 \in \mathbb{F}_3$, *see Figure* 6.2. *We can simply compute the syndrome, locate this error, and recover the original array. Since the erased column corresponds to the zero vector, and all the coefficients in column* 0 *are ones. The algorithm is simplified. For* $i \in [0,3]$, *we compute the syndromes and subtract them, we get zeros in all places except row* 0 *and* 2, *which satisfy* $0 + 2 = (0,0) + (1,0) = (1,0) = e_1$. *Therefore, we know the location of the error is in column* 1 *and row* 0 *or* 2. *But since* $W_0 = -W_2$, *we know the error is in* $\hat{b}_0$ *(If* $W_0 = W_2$, *the error is in* $\hat{b}_2$*).*

In practice, when we are confident that there are no element errors besides the node erasure, we can use the optimal rebuilding algorithm in Section 4.2.2 and access only half of the array to rebuild the failed node. However, we can also try to rebuild this node by accessing the other half of the array. Thus we will have two recovered version for the same node. If they are equal to each other, there are no element errors; if not, there are element errors. Thus, we have the flexibility of

	0	1	2	R	Z
0	a_0	b_0	c_0	$r_0 = a_0 + b_0 + c_0$	$z_0 = a_0 + 2b_2 + 2c_1$
1	a_1	b_1	c_1	$r_1 = a_1 + b_1 + c_1$	$z_1 = a_1 + 2b_3 + c_0$
2	a_2	b_2	c_2	$r_2 = a_2 + b_2 + c_2$	$z_2 = a_2 + b_0 + c_3$
3	a_3	b_3	c_3	$r_3 = a_3 + b_3 + c_3$	$z_3 = a_3 + b_1 + 2c_2$

Figure 6.1: $(5,3)$ zigzag code generated by the standard basis and the zero vector. All elements are over $\mathbb{F}_3$.

	0	1	2	R	Z	S_0	S_1	$W = S_0 - S_1$
0		$b_0 + e$	c_0	r_0	z_0	$-a_0 + e$	$-a_0$	e
1		b_1	c_1	r_1	z_1	$-a_1$	$-a_1$	0
2		b_2	c_2	r_2	z_2	$-a_2$	$-a_2 + e$	$-e$
3		b_3	c_3	r_3	z_3	$-a_3$	$-a_3$	0

Figure 6.2: An erroneous array of the $(5,3)$ zigzag code. There is a node erasure in column 0 and an element error in column 1. All the other elements are not corrupted. S_0, S_1 are the syndromes.

achieving optimal rebuilding ratio or correcting extra errors.

When there is one node erasure and more than one element errors in column j and row $R = \{r_1, r_2, \ldots, r_l\}$, following the same techniques, it is easy to see that the code is able to correct systematic errors if

$$R \cup (R + v_j) \neq R' \cup (R' + v_i)$$

for any set of rows R' and any other column index i, and $r_i \neq r_t + v_j$ for any $i, t \in [l]$.

When the code has more than two parities, the zigzag code can again correct element errors exceeding the bound by the Hamming distance. To detect errors, one can either compute the syndromes, or rebuild the erasures multiple times by accessing different e/r parts of the array.

Finally, it should be noted that if a node erasure and a single error happen in a parity column, then we cannot correct this error in the $(k+2,k)$ code.

6.4 Rebuilding Multiple Erasures

In this section, we discuss the rebuilding of e erasures, $1 \leq e \leq r$. We will first prove the lower bound for rebuilding ratio and repair bandwidth. Then we show a construction achieving the lower bound for systematic nodes. At last we generalize this construction and Construction 4.24, and propose a rebuilding algorithm using an arbitrary subgroup and its cosets. We repeat the $(k+r,k)$ zigzag Construction 4.24 below.

Construction 6.8 (Construction 4.24) *Let $A = (a_{i,j})$ be the information array of size $r^m \times k$, for some integers k, m. Let $T = \{v_0, ..., v_{k-1}\} \subseteq \mathbb{Z}_r^m$ be a subset of vectors of size k, where for each $v = (v_1, ..., v_m) \in T$,*

$$\gcd(v_1, ..., v_m, r) = 1, \tag{6.6}$$

and gcd *is the greatest common divisor. For any l, $0 \leq l \leq r-1$, and $v \in T$ we define the permutation $f_v^l : [0, r^m - 1] \to [0, r^m - 1]$ by $f_v^l(x) = x + lv$, where by abuse of notation we use $x \in [0, r^m - 1]$ both to represent the integer and its r-ary representation, and all the calculations are done over $\mathbb{Z}_r$. For simplicity denote the permutation $f_{v_j}^l$ as f_j^l for $v_j \in T$. For $t \in [0, r^m - 1]$, we define the zigzag set Z_t^l in parity node l, as the elements $a_{i,j}$ such that their coordinates satisfy $f_j^l(i) = t$. In a rebuilding of systematic node i the elements in rows $X_i^l = \{x \in [0, r^m - 1] : x \cdot v_i = r - l\}$ are rebuilt by parity node l, $l \in [0, r-1]$, where the inner product in the definition is done over $\mathbb{Z}_r$. From* (6.6) *we get that for any i and l, $|X_i^l| = r^{m-1}$.*

In this section, in order to simplify some of the results we will assume that r is a prime and the calculations are done over $\mathbb{F}_r$. Note that all the result can be generalized with minor changes for an arbitrary integer r and the ring $\mathbb{Z}_r$.

6.4.1 Lower Bounds

The next theorem shows that the rebuilding ratio for Construction 4.24 is at least e/r.

Theorem 6.9 *Let A be an array with r parity nodes constructed by Construction* 4.24*. In an erasure of $1 \leq e \leq r$ systematic nodes, the rebuilding ratio is at least $\frac{e}{r}$.*

Proof: In order to recover the information in the systematic nodes we need to use at least er^m zigzag sets from the r^{m+1} sets (There are r parity nodes, r^m zigzag sets in each parity). By the pigeonhole principle there is at least one parity node, such that at least er^{m-1} of its zigzag sets are used. Hence each remaining systematic node has to access its elements that are contained in these zigzag sets. Therefore each systematic node accesses at least er^{m-1} of its information out of r^m, which is a portion of $\frac{e}{r}$.

Since we use at least er^m zigzag sets, we use at least er^m elements in the r parity nodes, which is again a portion of $\frac{e}{r}$. Hence the overall rebuilding ratio is at least $\frac{e}{r}$. □

In a general code (not necessary MDS, systematic, or optimal update), what is the amount of information needed to transmit in order to rebuild e nodes? Assume that in the system multiple nodes

are erased, and we rebuild these nodes *simultaneously* from information in the remaining nodes. It should be noted that this model is a bit different from the distributed repair problem, where the recovery of each node is done separately. We follow the definitions and notations of [SRVKR12]. An *exact-repair reconstructing code* satisfies the following two properties: (i)Reconstruction: any k nodes can rebuild the total information. (ii)Exact repair: if e nodes are erased, they can be recovered exactly by transmitting information from the remaining nodes.

Suppose the total amount of information is $\mathcal{M}$, and the n nodes are $[n]$. For e erasures, $1 \leq e \leq r$, denote by α, d_e, β_e the amount of information stored in each node, the number of nodes connected to the erased nodes, and the amount of information transmitted by each of the nodes, respectively. For subsets $A, B \subseteq [n]$, W_A is the amount of information stored in nodes A, and S_A^B is the amount of information transmitted from nodes A to nodes B in the rebuilding.

The following results give lower bound of repair bandwidth for e erasures, and the proofs are based on [SRVKR12].

Lemma 6.10 *Let $B \subseteq [n]$ be a subset of nodes of size $|e|$, then for an arbitrary set of nodes A, $|A| \leq d_e$ such that $B \cap A = \emptyset$,*

$$H(W_B|W_A) \leq \min\{|B|\alpha, (d_e - |A|)\beta_e\}.$$

Proof: If nodes B are erased, consider the case of connecting to them nodes A and nodes C, $|C| = d_e - |A|$. Then the exact-repair condition requires

$$\begin{aligned} 0 &= H(W_B|S_A^B, S_C^B) \\ &= H(W_B|S_A^B) - I(W_B, S_C^B|S_A^B) \\ &\geq H(W_B|S_A^B) - H(S_C^B) \\ &\geq H(W_B|S_A^B) - (d - |A|)\beta_e \\ &\geq H(W_B|W_A) - (d - |A|)\beta_e. \end{aligned}$$

Moreover, it is clear that $H(W_B|W_A) \leq H(W_B) \leq |B|\alpha$ and the result follows. □

Theorem 6.11 *Any reconstructing code with file size $\mathcal{M}$ must satisfy for any $1 \leq e \leq r$*

$$\mathcal{M} \leq s\alpha + \sum_{i=0}^{\lfloor \frac{k}{e} \rfloor - 1} \min\{e\alpha, (d_e - ie - s)\beta_e\}$$

where $s = k \mod e, 0 \leq s < e$. *Moreover for an MDS code,* $\beta_e \geq \frac{e\mathcal{M}}{k(d-k+e)}$.

Proof: The file can be reconstructed from any set of k nodes, hence

$$\begin{aligned}\mathcal{M} &= H(\mathcal{W}_{[k]}) \\ &= H(W_{[s]}) + \sum_{i=0}^{\lfloor \frac{k}{e} \rfloor - 1} H(\mathcal{W}_{[ie+s+1,(i+1)e+s]} | \mathcal{W}_{[ie+s]}) \\ &\leq s\alpha + \sum_{i=0}^{\lfloor \frac{k}{e} \rfloor - 1} \min\{e\alpha, (d_e - ie - s)\beta_e\}.\end{aligned}$$

In an MDS code $\alpha = \frac{\mathcal{M}}{k}$, hence in order to satisfy the inequality any summand of the form $\min\{e\alpha, (d_e - ie - s)\beta_e\}$ must be at least $e\frac{\mathcal{M}}{k}$, which occurs if and only if $(d_e - (\lfloor \frac{k}{e} \rfloor - 1)e - s)\beta_e \geq \frac{e\mathcal{M}}{k}$. Hence we get

$$\beta_e \geq \frac{e\mathcal{M}}{k(d-k+e)}.$$

And the proof is completed. □

Therefore, the lower bound of the repair bandwidth for an MDS code is $\frac{e\mathcal{M}}{k(d-k+e)}$, which is the same as the lower bound of the rebuilding ratio in Theorem 6.9.

6.4.2 Rebuilding Algorithms

Next we discuss how to rebuild in case of e erasures, $1 \leq e \leq r$, for an MDS array code with optimal update. Theorem 6.11 gives the lower bound e/r on the rebuilding ratio for e erasures. Is this achievable? Let us first look at an example.

Example 6.12 *Consider the code in Figure* 4.8 *with* $r = 3$. *When* $e = 2$ *and columns* C_0, C_1 *are erased, we can access rows* $\{0,1,3,4,6,7\}$ *in column* C_2, C_3, *rows* $\{1,2,4,5,7,8\}$ *in column* C_4, *and rows* $\{2,0,5,3,8,6\}$ *in column* C_5. *One can check that the accessed elements are sufficient to rebuild the two erased columns, and the ratio is* $2/3 = e/r$. *It can be shown that similar rebuilding can be done for any two systematic node erasures. Therefore, in this example the lower bound is achievable.*

Consider an information array of size $p \times k$ and an (n,k) MDS code with $r = n - k$ parity nodes. Each parity node $l \in [0, r-1]$ is constructed from the set of permutations $\{f_i^l\}$ for $i \in [0, k-1]$. Notice that in the general case the number of rows p in the array is not necessarily a power of r. We will assume columns $[0, e-1]$ are erased. In an erasure of e columns, ep elements need

rebuilt, hence we need ep equations (zigzags) that contain these elements. In an optimal rebuilding, each parity node contributes ep/r equations by accessing the values of ep/r of its zigzag elements. Moreover, the union of the zigzag sets that create these zigzag elements, constitute an e/r portion of the elements in the surviving systematic nodes. In other words, assume that we access rows X from the surviving columns $[e, k-1]$, $X \subseteq [0, p-1]$, then $|X| = ep/r$ and

$$f_j^l(X) = f_i^l(X)$$

for any parity node $l \in [0, r-1]$ and $i, j \in [e, k-1]$. Note that it is equivalent that for any parity node $l \in [0, r-1]$ and surviving systematic node $j \in [e, k-1]$

$$f_j^l(X) = f_e^l(X).$$

Let G^l be the subgroup of the symmetric group S_p that is generated by the set of permutations $\{f_e^{-l} \circ f_j^l\}_{j=e}^{k-1}$. It is easy to see that the previous condition is also equivalent to that for any parity $l \in [0, r-1]$ the group G^l *stabilizes* X, i.e., for any $f \in G^l$, $f(X) = X$.

Assuming there is a set X that satisfies this condition, we want to rebuild the ep elements from the chosen ep equations, i.e., the ep equations with the ep variables being solvable. A necessary condition is that each element in the erased column will appear at least once in the chosen zigzag sets (equations). parity $l \in [0, r-1]$ accesses its zigzag elements $f_e^l(X)$, and these zigzag sets contain the elements in rows $(f_i^l)^{-1} f_e^l(X)$ of the erased column $i \in [0, e-1]$. Hence the condition is equivalent to that for any erased column $i \in [0, e-1]$

$$\cup_{l=0}^{r-1} (f_i^l)^{-1} f_e^l(X) = [0, p-1].$$

These two conditions are necessary for optimal rebuilding ratio. In addition, we need to make sure that the ep equations are linearly independent, which depends on the coefficients in the linear combinations that created the zigzag elements. We summarize:

Sufficient and necessary conditions for optimal rebuilding ratio in e erasures: There exists a set $X \subseteq [0, p-1]$ of size $|X| = ep/r$, such that

1. For any parity node $l \in [0, e-1]$ the group G^l stabilizes the set X, i.e., for any $g \in G^l$

$$g(X) = X, \tag{6.7}$$

where G^l is generated by the set of permutations $\{f_e^{-l} \circ f_j^l\}_{j=e}^{k-1}$.

2. For any erased column $i \in [0, e-1]$,

$$\cup_{l=0}^{r-1} (f_i^l)^{-1} f_e^l(X) = [0, p-1]. \tag{6.8}$$

3. The ep equations (zigzag sets) defined by the set X are linearly independent.

The previous discussion gave the condition for optimal rebuilding ratio in an MDS optimal-update code with e erasures in general. Next will interpret these conditions in the special case where the number of rows $p = r^m$, and the permutations are generated by $T = \{v_0, v_1, \ldots, v_{k-1}\} \subseteq \mathbb{F}_r^m$ and Construction 4.24, i.e., $f_i^l(x) = x + lv_i$ for any $x \in [0, r^m - 1]$. Note that in the case of r a prime

$$G^1 = G^2 = ... = G^{r-1},$$

and in that case we simply denote the group as G. The following theorem gives a simple characterization for sets that satisfy condition 1.

Theorem 6.13 *Let $X \subseteq \mathbb{F}_r^m$ and G defined above then G stabilizes X, if and only if X is a union of cosets of the subspace*

$$Z = \text{span}\{v_{e+1} - v_e, \ldots, v_{k-1} - v_e\}. \tag{6.9}$$

Proof: It is easy to check that any coset of Z is stabilized by G, hence if X is a union of cosets it is also a stabilized set. For the other direction let $x, y \in \mathbb{F}_r^m$ be two vectors in the same coset of Z, it is enough to show that if $x \in X$ then also $y \in X$. Since $y - x \in Z$ there exist $\alpha_1, ..., \alpha_{k-1-e} \in [0, r-1]$ such that $y - x = \sum_{i=1}^{k-1-e} \alpha_i (v_{e+i} - v_e)$. Since $f(X) = X$ for any $f \in G$ we get that $f(x) \in X$ for any $x \in X$ and $f \in G$, hence

$$\begin{aligned} y &= x + y - x \\ &= x + \sum_{i=1}^{k-1-e} \alpha_i (v_{e+i} - v_e) \\ &= f_e^{-\alpha_{k-1-e}} f_{k-1}^{\alpha_{k-1-e}} ... f_e^{-\alpha_1} f_{e+1}^{\alpha_1}(x), \end{aligned}$$

for $f_e^{-\alpha_{k-1-e}} f_{k-1}^{\alpha_{k-1-e}} ... f_e^{-\alpha_1} f_{e+1}^{\alpha_1} \in G$. So $y \in X$ and the result follows. □

Remark: For any set of vectors S and $v, u \in S$,

$$\text{span}\{S - v\} = \text{span}\{S - u\}.$$

Here $S - v = \{v_i - v | v_i \in S\}$. Hence, the subspace Z defined in the previous theorem does not depend on the choice of the vector v_e. By the previous theorem we interpret the **necessary and sufficient conditions of an optimal code** as follows:
There exists a set $X \subseteq \mathbb{F}_r^m$ of size $|X| = er^{m-1}$, such that

1. X is a union of cosets of

$$Z = \text{span}\{v_{e+1} - v_e, \ldots, v_{k-1} - v_e\}.$$

2. For any erased column $i \in [0, e-1]$,

$$\cup_{l=0}^{r-1} (X + l(v_i - v_e)) = \mathbb{F}_r^m. \tag{6.10}$$

3. The er^m equations (zigzag sets) defined by the set X are linearly independent.

The following theorem gives a simple equivalent condition for conditions $1, 2$.

Theorem 6.14 *There exists a set $X \subseteq \mathbb{F}_r^m$ of size $|X| = er^{m-1}$ such that conditions $1, 2$ are satisfied if and only if*

$$v_i - v_e \notin Z, \tag{6.11}$$

for any erased column $i \in [0, e-1]$.

Proof: Assume conditions $1, 2$ are satisfied. If $v_i - v_e \in Z$ for some erased column $i \in [0, e-1]$ then $X = \cup_{l=0}^{r-1}(X + l(v_i - v_e)) = \mathbb{F}_r^m$, which is a contradiction to $X \subsetneq \mathbb{F}_r^m$. On the other hand, If (6.11) is true, then $v_i - v_e$ can be viewed as a permutation that acts on the cosets of Z. The number of cosets of Z is $r^m/|Z|$ and this permutation (when it is written in cycle notation) contains $r^{m-1}/|Z|$ cycles, each with length r. For each $i \in [0, e-1]$ choose $r^{m-1}/|Z|$ cosets of Z, one from each cycle of the permutation $v_i - v_e$. In total $er^{m-1}/|Z|$ cosets are chosen for the e erased nodes. Let X be the union of the cosets that were chosen. It is easy to see that X satisfies condition 2. If $|X| < er^{m-1}$ (Since there might be cosets that were chosen more than once) add arbitrary $(er^{m-1} - |X|)/|Z|$ other cosets of Z, and also condition 1 is satisfied. □

In general, if (6.11) is not satisfied, the code does not have an optimal rebuilding ratio. However we can define

$$Z = \text{span}\{v_i - v_e\}_{i \in I}, \tag{6.12}$$

where we assume w.l.o.g. $e \in I$ and $I \subseteq [e, k-1]$ is a maximal subset of surviving nodes that satisfies for any erased node $j \in [0, e-1], v_j - v_e \notin Z$. Hence from now on we assume that Z is defined by a subset of surviving nodes I. This set of surviving nodes will have an optimal rebuilding ratio (see Corollary 6.18), i.e., in the rebuilding of columns $[0, e-1]$, columns I will access a portion of e/r of their elements. The following theorem gives a sufficient condition for the er^m equations defined by the set X to be solvable linear equations.

Theorem 6.15 *Suppose that there exists a subspace X_0 that contains Z such that for any erased node $i \in [0, e-1]$*

$$X_0 \oplus \text{span}\{v_i - v_e\} = \mathbb{F}_r^m, \tag{6.13}$$

then the set X defined as an union of some e cosets of X_0 satisfies conditions $1, 2$ and 3 over a field large enough.

Proof: Condition 1 is trivial. Note that by (6.13), $l(v_i - v_e) \notin X_0$ for any $l \in [1, r-1]$ and $i \in [0, e-1]$, hence $\{X_0 + l(v_i - v_e)\}_{l \in [0, r-1]}$ is the set of cosets of X_0. Let $X_j = X_0 + j(v_i - v_e)$ be a coset of X_0 for some $i \in [0, e-1]$ and suppose $X_j \subset X$. Now let us check condition 2:

$$\begin{aligned}
\cup_{l=0}^{r-1}(X + l(v_i - v_e)) &\supseteq \cup_{l=0}^{r-1}(X_j + l(v_i - v_e)) \\
&= \cup_{l=0}^{r-1}(X_0 + j(v_i - v_e) + l(v_i - v_e)) \\
&= \cup_{l=0}^{r-1}(X_0 + (j+l)(v_i - v_e)) \\
&= \cup_{t=0}^{r-1}(X_0 + t(v_i - v_e)) \qquad (6.14) \\
&= \mathbb{F}_r^m. \qquad (6.15)
\end{aligned}$$

(6.14) holds since $j + l$ is computed mod r. So condition 2 is satisfied. Next we prove condition 3. There are er^m unknowns and er^m equations. Writing the equations in a matrix form we get $AY = b$, where A is an $er^m \times er^m$ matrix. Y, b are vectors of length er^m, and $Y = (y_1, ..., y_{er^m})^T$ is the unknown vector. The matrix $A = (a_{i,j})$ is defined as $a_{i,j} = x_{i,j}$ if the unknown y_j appears in the i-th equation, otherwise $a_{i,j} = 0$. Hence we can solve the equations if and only if there is assignment for the unknowns $\{x_{i,j}\}$ in the matrix A such that $\det(A) \neq 0$. By (6.15), accessing

rows corresponding to any coset X_j will give us equations where each unknown appears exactly once. Since X is a union of e cosets, each unknown appears e times in the equations. Thus each column in A contains e unknowns. Moreover, each equation contains one unknown from each erased node, thus any row in A contains e unknowns. Then by Hall's Marriage Theorem [Hal35] we conclude that there exists a permutation f on the integers $[1, er^m]$ such that

$$\prod_{i=1}^{er^m} a_{i,f(i)} \neq 0.$$

Hence the polynomial $\det(A)$ when viewed as a symbolic polynomial, is not the zero polynomial, i.e.,

$$\det(A) = \sum_{f \in S_{er^m}} \operatorname{sgn}(f) \prod_{i=1}^{er^m} a_{i,f(i)} \neq 0.$$

By Theorem 4.9 we conclude that there is an assignment from a field large enough for the unknowns such that $\det(A) \neq 0$, and the equations are solvable. Note that this proof is for a specific set of erased nodes. However if (6.13) is satisfied for any set of e erasures, multiplication of all the nonzero polynomials $\det(A)$ derived for any set of erased nodes is again a nonzero polynomial and by the same argument there is an assignment over a field large enough such that any of the matrices A is invertible, and the result follows. □

In order to use Theorem 6.15, we need to find a subspace X_0 as in (6.13). The following theorem shows that such a subspace always exists, moreover it gives an explicit construction of it.

Theorem 6.16 *Suppose $1 \leq e < r$ erasures occur. Let Z be defined by* (6.12) *and $v_i - v_e \notin Z$ for any erased node $i \in [0, e-1]$. Then there exists $u \perp Z$ such that for any $i \in [0, e-1]$,*

$$u \cdot (v_i - v_e) \neq 0. \tag{6.16}$$

Moreover the orthogonal subspace $X_0 = (u)^\perp$ satisfies (6.13)*.*

Proof: First we will show that such vector u exists. Let $u_1, \ldots u_t$ be a basis for $(Z)^\perp$ the orthogonal subspace of Z. Any vector u in $(Z)^\perp$ can be written as $u = \sum_{j=1}^{t} x_j u_j$ for some x_j's. We claim that for any $i \in [0, e-1]$ there exists j such that $u_j \cdot (v_i - v_e) \neq 0$. Because otherwise, $(Z)^\perp = \operatorname{span}\{u_1, \ldots, u_t\} \perp v_i - v_e$, which means $v_i - v_e \in Z$ and reaches a contradiction. Thus

the number of solutions for the linear equation

$$\sum_{j=1}^{t} x_j u_j \cdot (v_i - v_e) = 0$$

is r^{t-1}, which equals the number of u such that $u \cdot (v_i - v_e) = 0$. Hence by the union bound there are at most er^{t-1} vectors u in $(Z)^{\perp}$ such that $u \cdot (v_i - v_e) = 0$ for some erased node $i \in [0, e-1]$. Since $|(Z)^{\perp}| = r^t > er^{t-1}$ there exists u in $(Z)^{\perp}$ such that for any erased node $i \in [0, e-1]$,

$$u \cdot (v_i - v_e) \neq 0.$$

Define $X_0 = (u)^{\perp}$, and note that for any erased node $i \in [0, e-1], v_i - v_e \notin X_0$, since $u \cdot (v_i - v_e) \neq 0$ and X_0 is the orthogonal subspace of u. Moreover, since X_0 is a hyperplane we conclude that

$$X_0 \oplus \text{span}\{v_i - v_e\} = \mathbb{F}_r^m,$$

and the result follows. □

Theorems 6.15 and 6.16 give us **an algorithm to rebuild multiple erasures:**

1. Find Z by (6.12) satisfying (6.11).
2. Find $u \perp Z$ satisfying (6.16). Define $X_0 = (u)^{\perp}$ and X as a union of e cosets of X_0.
3. Access rows $f_e^l(X)$ in parity $l \in [0, r-1]$ and all the corresponding information elements.

We know that under a proper selection of coefficients the rebuilding is possible.

In the following we give two examples of rebuilding using this algorithm. The first example shows an optimal rebuilding for any set of e node erasures. As mentioned above, the optimal rebuilding is achieved since (6.11) is satisfied, i.e., $I = [e, k-1]$.

Example 6.17 *Let $T = \{v_0, v_1, \ldots, v_m\}$ be a set of vectors that contains an orthonormal basis of $\mathbb{F}_r^m$ together with the zero vector. Suppose columns $[0, e-1]$ are erased. Note that in that case $I = [e, m]$ and Z is defined as in* (6.12)*. Define*

$$u = \sum_{j=e}^{m} v_j,$$

and $X_0 = (u)^\perp$. *When* $m = r$ *and* $e = r-1$, *modify* u *to be*

$$u = \sum_{i=1}^{m} v_i.$$

It is easy to check that $u \perp Z$ *and for any erased column* $i \in [0, e-1], u \cdot (v_i - v_e) = -1$. *Therefore by Theorems* 6.15 *and* 6.16 *a set* X *defined as a union of an arbitrary* e *cosets of* X_0 *satisfies conditions* 1, 2 *and* 3, *and optimal rebuilding is achieved.*

If we take the orthonormal vectors in T as the standard basis and zero vector, we get the code in Theorem 4.27 and we know for any e erasures, the rebuilding ratio is e/r. When columns C_0, C_1 are erased in Figure 4.8, $u = e_2$ and $X_0 = (u)^\perp = \text{span}\{e_1\} = \{0, 3, 6\}$. Take X as the union of X_0 and its coset $\{1, 4, 7\}$, which is the same as Example 6.12. One can check that each erased element appears exactly 3 times in the equations and the equations are solvable in $\mathbb{F}_4$. Similarly, the equations are solvable for other 2 systematic erasures.

Before we proceed to the next example, we give an upper bound for the rebuilding ratio using Theorem 6.15 and a set of nodes I.

Corollary 6.18 *Theorem* 6.15 *requires rebuilding ratio at most*

$$\frac{e}{r} + \frac{(r-e)(k-|I|-e)}{r(k+r-e)}$$

Proof: By Theorem 6.15, the fraction of accessed elements in columns I and the parity columns is e/r of each column. Moreover, the accessed elements in the rest columns are at most an entire column. Therefore, the ratio is at most

$$\frac{\frac{e}{r}(|I|+r) + (k-|I|-e)}{k+r-e} = \frac{e}{r} + \frac{(r-e)(k-|I|-e)}{r(k+r-e)}$$

and the result follows. □

Note that as expected when $|I| = k-e$ the rebuilding ratio is optimal, i.e. e/r. In the following example the code has $O(m^2)$ columns. The set I does not contain all the surviving systematic nodes, hence the rebuilding is not optimal but is at most $\frac{1}{2} + O(\frac{1}{m})$.

Example 6.19 *Suppose* $2|m$. *Let* $T = \{v = (v_1, \ldots, v_m) : \|v\|_1 = 2, v_i = 1, v_j = 1,$ *for some* $i \in [1, m/2], j \in [m/2+1, m]\} \subset \mathbb{F}_2^m$ *be the set of vectors generating the code with* $r = 2$ *parities, hence the number of systematic nodes is* $|T| = k = m^2/4$. *Suppose column* $w = (w_1, \ldots, w_m)$,

$w_1 = w_{m/2+1} = 1$ *is erased. Define the set* $I = \{v \in T : v_1 = 0\}$*, and*

$$Z = \text{span}\{v_i - v_e | i \in I\}$$

for some $e \in I$*. Thus* $|I| = m(m-2)/4$*. It can be seen that* Z *defined by the set* I *satisfies* (6.11)*, i.e.,* $w - v_e \notin Z$ *since the first coordinate of a vector in* Z *is always* 0*, as oppose to* 1 *for the vector* $w - v_e$*. Define* $u = (0, 1, ..., 1)$ *and* $X_0 = (u)^{\perp}$*. It is easy to check that* $u \perp Z$ *and* $u \cdot (w - v_e) = 1 \neq 0$*. Hence, the conditions in Theorem* 6.16 *are satisfied and rebuilding can be done using* X_0*. Moreover by Corollary* 6.18 *the rebuilding ratio is at most*

$$\frac{1}{2} + \frac{1}{2}\frac{(m/2)-1}{(m^2/4)+1} \approx \frac{1}{2} + \frac{1}{m},$$

which is a little better than Theorem 4.14 *in the constants. Note that by similar coefficients assignment of Construction* 4.13*, we can use a field of size* 5 *or* 8 *to assure that the code will be an MDS code.*

6.4.3 Minimum Number of Erasures with Optimal Rebuilding

Next we want to point out a surprising phenomenon. We say that a set of vectors S satisfies *property* e for $e \geq 1$ if for any subset $A \subseteq S$ of size e and any $u \in A$,

$$u - v \notin \text{span}\{w - v : w \in S \backslash A\},$$

where $v \in S \backslash A$. Recall that by Theorem 6.14 any set of vectors that generates a code $\mathcal{C}$ and can rebuild optimally any e erasures, satisfies property e. The following theorem shows that this property is monotonic, i.e., if S satisfies property e then it also satisfies property a for any $e \leq a \leq |S|$.

Theorem 6.20 *Let* S *be a set of vectors that satisfies property* e*, then it also satisfies property* a*, for any* $e \leq a \leq |S|$*.*

Proof: Let $A \subseteq S, |A| = e + 1$ and assume to the contrary that $u - v \in \text{span}\{w - v : w \in S \backslash A\}$ for some $u \in A$ and $v \in S \backslash A$. $|A| \geq 2$ hence there exists $x \in A \backslash u$. It is easy to verify that $u - v \in \text{span}\{w - v : w \in S \backslash A^*\}$, where $A^* = A \backslash x$ and $|A^*| = e$ which contradicts the property e for the set S. □

Hence, from the previous theorem we conclude that a code $\mathcal{C}$ that can rebuild optimally e erasures, is able to rebuild optimally any number of erasures greater than e as well. However, as pointed out already there are codes with r parities that cannot rebuild optimally from some $e < r$ erasures. Therefore, one might expect to find a code $\mathcal{C}$ with parameter $e^* \geq 1$ such that it can rebuild optimally e erasures *only* when $e^* \leq e \leq r$. For example, for $r = 3, m = 2$ let $\mathcal{C}$ be the code constructed by the vectors $\{0, e_1, e_2, e_1 + e_2\}$. We know that any code with more than 3 systematic nodes cannot rebuild one erasure optimally, since the size of a family of orthogonal permutations over the integers $[0, 3^2 - 1]$ is at most 3. However, one can check that for any two erased columns, the conditions in Theorem 6.15 are satisfied hence the code can rebuild optimally for any $e = 2$ erasures and we conclude that $e^* = 2$ for this code.

The phenomena that some codes has a threshold parameter e^*, such that *only* when the number of erasures e is at least as the threshold e^* then the code can rebuild optimally, is a bit counter intuitive and surprising. This phenomena gives rise to another question. We know that for a code constructed with vectors from $\mathbb{F}_r^m$, the maximum number of systematic columns for optimal rebuilding of $e = 1$ erasures is $m + 1$ (Theorem 4.28). Can the number of systematic columns in a code with an optimal rebuilding of $e > 1$ erasures be increased? The previous example shows a code with 4 systematic columns can rebuild optimally any $e = 2$ erasures. But Theorem 4.28 shows that when $r = 3, m = 2$, optimal rebuilding for 1 erasure implies no more than 3 systematic columns. Hence the number of systematic columns is increased by at least 1 compared to codes with 9 rows and optimal rebuilding of 1 erasure. The following theorem gives an upper bound for the maximum systematic columns in a code that rebuilds optimally any e erasures.

Theorem 6.21 *Let $\mathcal{C}$ be a code constructed by Construction* 4.24 *and vectors from $\mathbb{F}_r^m$. If $\mathcal{C}$ can rebuild optimally any e erasures, for some $1 \leq e < r$, then the number of systematic columns k in the code satisfies*

$$k \leq m + e.$$

Proof: Consider a code with length k and generated by vectors $v_0, v_1, \ldots, v_{k-1}$. If these vectors are linearly independent then $k \leq m$ and we are done. Otherwise they are dependent. Suppose e columns are erased, $1 \leq e < r$. Let v_e be a surviving column. Consider a new set a of vectors: $T = \{v_i - v_e : i \in [0, k-1], i \neq e\}$. We know that the code can rebuild optimally only if (6.11) is satisfied for all possible e erasures. Thus for any $i \neq e$, $i \in [0, k-1]$, if column i is erased and column e is not, we have $v_i - v_e \notin Z$ and thus $v_i - v_e \neq 0$. So every vector

in T is nonzero. Let s be the minimum number of dependent vectors in T, that is, the minimum number of vectors in T such that they are dependent. For nonzero vectors, we have $s \geq 2$. Say $\{v_{e+1} - v_e, v_{e+2} - v_e, \ldots, v_{e+s} - v_e\}$ is a minimum dependent set of vector. Since any $m+1$ vectors are dependent in $\mathbb{F}_r^m$,

$$s \leq m+1.$$

We are going to show $k - e \leq s - 1$. Suppose to the contrary that the number of remaining columns satisfies $k - e \geq s$ and e erasures occur. When column v_{e+s} is erased and the s columns $\{v_e, v_{e+1}, \ldots, v_{e+s-1}\}$ are not, we should be able to rebuild optimally. However since we chose a dependent set of vectors, $v_{e+s} - v_e$ is a linear combination of $\{v_{e+1} - v_e, v_{e+2} - v_e, \ldots, v_{e+s-1} - v_e\}$, whose span is contained in Z in (6.11). Hence (6.11) is violated and we reach a contradiction. Therefore,

$$k - e \leq s - 1 \leq m.$$

□

Notice that this upper bound is tight. For $e = 1$ we already gave codes with optimal rebuilding of 1 erasure and $k = m+1$ systematic columns. Moreover, for $e = 2$ the code already presented in this section and constructed by the vectors $0, e_1, e_2, e_1 + e_2$, reaches the upper bound with $k = 4$ systematic columns.

6.4.4 Generalized Rebuilding Algorithms

The rebuilding algorithms presented in Constructions 4.1,4.24 and Theorem 6.15 all use a specific subspace and its cosets in the rebuilding process. This method of rebuilding can be generalized by using an arbitrary subspace as explained below.

Let $T = \{v_0, \ldots, v_{k-1}\}$ be a set of vectors generating the code in Construction 4.24 with r^m rows and r parities. Suppose e columns $[0, e-1]$ are erased. Let Z be a proper subspace of $\mathbb{F}_r^m$. In order to rebuild the erased nodes, in each parity column $l \in [0, r-1]$, access the zigzag elements z_i^l for $i \in X_l$, and X_l is a union of cosets of Z. In each surviving node, access all the elements that are in the zigzag sets X_l of parity l. More specifically, access element $a_{i,j}$ in the surviving column $j \in [e, k-1]$ if $i + lv_j \in X_l$. Hence, in the surviving column j and parity l, we access elements in rows $X_l - lv_j$. In order to make the rebuilding successful we impose the following conditions on the sets $X_0, ..., X_l$. Since the number of equations needed is at least as the number of erased elements,

we require

$$\sum_{l=0}^{r-1} |X_l| = er^m. \tag{6.17}$$

Moreover we want the equations to be solvable, hence for any erased column $i \in [0, e-1]$,

$$\cup_{l=0}^{r-1} X_l - lv_i = [0, r^m - 1] \text{ multiplicity } e, \tag{6.18}$$

which means if the union is viewed as a multiset, then each element in $[0, r^m - 1]$ appears exactly e times. This condition makes sure that the equations are solvable by Hall's theorem (see Theorem 6.15). Under these conditions we would like to minimize the ratio, i.e., the number of accesses which is,

$$\min_{X_0,\ldots,X_{r-1}} \sum_{j=e}^{k-1} |\cup_{l=0}^{r-1} (X_l - lv_j)|. \tag{6.19}$$

In summary, for the **generalized rebuilding algorithm** one first chooses a subspace Z, and then solves the minimization problem in (6.19) subject to (6.17) and (6.18).

The following example interprets the minimization problem for a specific case.

Example 6.22 *Let $r = 2, e = 1$, i.e., two parities and one erasure, then equations* (6.17),(6.18) *becomes*

$$|X_0| + |X_1| = 2^m, X_0 \cup X_1 + v_0 = [0, 2^m - 1].$$

Therefore $X_1 + v_0 = \overline{X_0}$. The objective function in (6.19) *becomes,*

$$\min_{X_0,X_1} \sum_{j=1}^{k-1} |X_0 \cup X_1 + v_j| = \min_{X_0} \sum_{j=1}^{k-1} |X_0 \cup (\overline{X_0} + v_0 + v_j)|.$$

Each $v_0 + v_j$ defines a permutation $f_{v_0+v_j}$ on the cosets of Z by $f_{v_0+v_j}(A) = A + v_0 + v_j$ for a coset A of Z. If $v_0 + v_j \in Z$ then $f_{v_0+v_j}$ is the identity permutation and $|X_0 \cup (\overline{X_0} + v_0 + v_j)| = 2^m$, regardless of the choice of X_0. However, if $v_0 + v_j \notin Z$, then $f_{v_0+v_j}$ is of order 2, i.e., it is composed of disjoint cycles of length 2. Note that if $f_{v_0+v_j}$ maps A to B and only one of the cosets A, B is contained in X_0, say A, then only A is contained in $X_0 \cup (\overline{X_0} + v_0 + v_j)$. On the other hand, if both $A, B \in X_0$ or $A, B \notin X_0$ then,

$$A, B \subseteq X_0 \cup (\overline{X_0} + v_0 + v_j).$$

In other words, (A, B) is a cycle in $f_{v_0+v_j}$ which is totally contained in X_0 or in $\overline{X_0}$. Define N_j^X as

the number of cycles (A, B) *in the permutation* $f_{v_0+v_j}$ *that are totally contained in* X *or in* $\overline{X}$*, where* X *is a union of some cosets of* Z*. It is easy to see that the minimization problem is equivalent to minimizing*

$$\min_{X} \sum_{j=1}^{k-1} N_j^X. \tag{6.20}$$

In other words, we want to find a set X *which is a union of cosets of* Z*, such that the number of totally contained or totally not contained cycles in the permutations defined by* $v_j + v_0$*,* $j \in [1, k-1]$ *is minimized.*

From the above example, we can see that given a non-optimal code with two parities and one erasure, finding the solution in (6.20) requires minimizing for the sum of these $k-1$ permutations, which is an interesting combinatorial problem. Moreover, by choosing a different subspace Z we might be able to get a better rebuilding algorithm than that in Construction 4.1 or Theorem 6.15.

6.5 Concluding Remarks

In this chapter, we have discussed a variety of failure scenarios in storage systems. The failures can be in an entire column or node, in some specific element or symbol, and we developed algorithms to reconstruct the file. There are quite a few other failure or rebuilding models one might encounter in practice. For example, when the request is for part of a node instead of an entire node, or *degraded read*, the rebuilding ratio or code constructions are unknown.

For another example, during the rebuilding of one node, another node erasure may occur. Having rebuilt some of the first failed node, it is interesting to find out how to rebuild the rest lost information. This problem is also related to the cache size allocated during rebuilding. If the cache size is as large as the file, we need to do nothing but simply rebuild from the cache. However, this is not practical when the file or the number of nodes is large. Then it is not obvious what the necessary access and computation is.

Chapter 7

Long MDS Array Codes with Optimal Bandwidth

7.1 Introduction

We have studied MDS array codes with optimal rebuilding access ratio. However, one may find that the number of systematic nodes is small compared to the size of each node. We showed some constructions to lengthen the code in Section 4.4, however, these constructions do not have optimal ratio exactly. In this chapter, we relax the constraint of optimal access, instead we construct long MDS codes with optimal bandwidth.

The *repair bandwidth (fraction)* is defined as the fraction of the remaining data transmitted in order to correct e erasures. In the chapter, we focus on rebuilding of a single systematic node, since it is more commonly seen and relates directly to retrieving information. Denote the size of each node or the column length as l. In other words, each column is a vector of length l and all the elements are from a finite filed $\mathbb{F}$. Let $n, k, r = n - k$ be the number of all, systematic, and parity nodes, respectively. Assume that a code is optimal bandwidth, then we study the *code length*, i.e., the number of systematic nodes k given l in this chapter. We write a code in the notation (n, k, l) to emphasize these parameters.

For example, in Figure 7.1, we show an MDS code with 4 systematic nodes, $r = 2$ parity nodes, and column length $l = 2$. One can check that this code can correct any two erasures, therefore it is an MDS code. In order to repair any systematic node, only $1/r = 1/2$ fraction of the remaining information is transmitted. Thus this code has optimal repair.

If we are interested in the code length, low-rate ($k/n \leq 1/2$) codes have a linear code length $l + 1$ [SRKR10, SR10a]; on the other hand, high-rate ($k/n > 1/2$) constructions are relatively

N1	N2	N3	N4	P1	P2
a	b	c	d	$a+b+c+d$	$2a+w+2b+3c+d$
w	x	y	z	$w+x+y+z$	$3w+b+3x+2y+z$

Figure 7.1: $(n=6,k=4,l=2)$ MDS code over finite field $\mathbb{F}_4$ generated by primitive polynomial x^2+x+1. Here 2 is a primitive element of the field. The first 4 nodes are systematic and the last 2 are parities. To repair $N1$ transmit the first row from every remaining node. To repair $N2$ transmit the second row. To repair $N3$ transmit the sum of both rows. And to repair $N4$ transmit the sum of the first row and 2 times the second row from nodes $N1, N2, N3, P1$, and the sum of the first row and 3 times the second row from node $P2$.

short. For example, suppose that we have 2 parity nodes, then the number of systematic nodes is only $\log l$ in all of the constructions, except for [CHLM11] it is $2\log l$. In [TWB12] it is shown that an upper bound for the code length is $k \leq 1 + l\binom{l}{l/2}$, but the tightness of this bound is not known. It is obvious that there is a big gap between this upper bound and the constructed codes.

The main contribution of this chapter is to construct codes with 2 parity nodes and $3\log l$ systematic nodes. The code uses a finite field of size $1+2\log l$. Moreover, we will give a general construction of high-rate codes with $(r+1)\log_r l$ systematic nodes for arbitrary number of parities r. It turns out that this construction is a combination of the code in [CHLM11] and also [CHL11, PDC11a, TWB11].

In this chapter, we will introduce the bandwidth and code length problem, and present long MDS code constructions first for two parities and then for arbitrary number of parities.

7.2 Problem Settings

An (n,k,l) MDS array code is an $(n-k)$-erasure-correcting code such that each symbol is a column of length l. The number of systematic symbols is k and the number of parity symbols is $r=n-k$. We call each symbol a column or a node, and k the *code length*. We assume that the code is systematic, hence the first k nodes of the code are information or systematic nodes, and the last r nodes are parity or redundancy nodes.

Suppose the columns of the code are $C_1, C_2, \ldots, C_n$, each being a column vector in $\mathbb{F}^l$, for some finite field $\mathbb{F}$. We assume that for parity node $k+i$, information node j, the *coding matrix* is $A_{i,j}$ of size $l \times l$, $i \in [r]$, $j \in [k]$. And the parity columns are computed as

$$C_{k+i} = \sum_{j=1}^{k} A_{i,j} C_j,$$

for all $i \in [r]$. For example, in Figure 7.1, the coding matrices are $A_{1,j} = I$ for all $j \in [k]$ and $A_{2,j}$, $j = 1, 2, 3, 4$ are

$$\begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}, \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Here the finite field is $\mathbb{F}_4$ generated by $x^2 + x + 1$. In our constructions, we require that $A_{1,j} = I$ for all $j \in [k]$. Hence the first parity is the row sum of the information array. Even though this assumption is not necessarily true for an arbitrary linear MDS array code, it can be shown that any linear code can be equivalently transformed into one with such coding matrices [TWB12].

Suppose a code has optimal repair for any systematic node i, $i \in [k]$, meaning only a fraction of $1/r$ data is transmitted in order to repair it. When a systematic node i is erased, we are going to use size $l/r \times l$ matrices $S_{i,j}$, $j \neq i, j \in [n]$, to repair the node: From a surviving node j, we are going to compute and transmit $S_{i,j}C_j$, which is only $1/r$ of the information in this node.

Notations: In order to simplify the notations, we write $S_{i,j}$ and $S_{i,k+t}A_{t,j}$ both as matrices of size $l/r \times l$ and the subspaces of their row spans.

Optimal repair of a systematic node i is equivalent to the following **subspace property**: There exist matrices $S_{i,j}$, $j \neq i, j \in [n]$, all with size $l/r \times l$, such that for all $j \neq i, j \in [k], t \in [r]$,

$$S_{i,j} = S_{i,k+t}A_{t,j}, \tag{7.1}$$

where the equality is defined on the row spans instead of the matrices. And

$$\sum_{t=1}^{r} S_{i,k+t}A_{t,i} = \mathbb{F}^l. \tag{7.2}$$

Here the sum of two subspaces A, B of $\mathbb{F}^l$ is defined as $A + B = \{a + b : a \in A, b \in B\}$. Obviously, the dimension of each subspace $S_{i,k+t}A_{t,i}$ is no more than l/r, and the sum of r such subspaces has dimension no more than l. This means these subspaces intersect only on the zero vector. Therefore, the sum is actually the direct sum of vector spaces. Moreover, we know that each $S_{i,k+t}$ has full rank l/r.

We claim that (7.1) (7.2) are necessary and sufficient conditions for optimal repair. The sketch of the proof is as follows: suppose the code has optimal repair-bandwidth, then we need to transmit l/r elements from each surviving column. Suppose we transmit $S_{i,j}C_j$ from a systematic node $j \neq i, j \in [k]$, and $S_{i,k+t}C_{k+t} = \sum_{z=1}^{k} S_{i,k+t}A_{t,z}C_z$ from a parity node $k + t \in [k + 1, k + r]$.

Our goal is to recover C_i and cancel out all C_j, $j \neq i, j \in [k]$. In order to cancel out C_j, (7.1) must be satisfied. In order to solve C_i, all equations related to C_i must have full rank l, so (7.2) is satisfied. One the other hand, if (7.1) (7.2) are satisfied, one can transmit $S_{i,j}C_j$ from each node j, $j \neq i, j \in [n]$ and optimally repair the node i. Similar interference alignment technique was first introduced in [SR10b, CJM10] for the repair problem. Also, [SRKR10] was the first to formally prove similar conditions.

It is shown in [TWB12] that we can further simplify our repair strategy of node i and assume $S_{i,j} = S_i$, for all $j \neq i, j \in [n]$ by equivalent transformation of the coding matrices (probably with an exception of the strategy of one node). Then the **subspace property** becomes for any $j \neq i, j \in [k], t \in [r]$,

$$S_i = S_i A_{t,j}. \tag{7.3}$$

Again the equality means equality of row spans. And the sum of subspaces satisfies

$$\sum_{t=1}^{r} S_i A_{t,i} = \mathbb{F}^l. \tag{7.4}$$

Notice that if (7.3) is satisfied, we can say that S_i is an invariant subspace of $A_{t,j}$ (multiplied on the left) for all parity nodes $k+t$ and all information nodes $j \neq i$. If $A_{t,j}$ is diagonalizable and has l linearly independent left eigenvectors, an invariant subspace has a set of basis which are all eigenvectors of $A_{t,j}$. As a result, our goal is to find matrices $A_{t,j}$ and their invariant subspaces. And by using sufficiently large finite field and varying the eigenvalues of the coding matrices, we are able to ensure that the codes are MDS. Therefore, we will first focus on finding eigenvectors of the coding matrices and then discuss about the eigenvalues.

For example, in Figure 7.1, the matrices S_i, $i = 1, 2, 3$ are

$$(1, 0), (0, 1), (1, 1).$$

One can check that the subspace property (7.3)(7.4) is satisfied for $i \in [3]$. For instance, since $S_3 = (1, 1)$ is an eigenvector for $A_{t,j}$, $t = 1, 2, j = 1, 2, 4$, we have $S_3 = S_3 A_{t,j}$. And it is easy to check that $S_3 \oplus S_3 A_{2,3} = \text{span}(1, 1) \oplus \text{span}(3, 2) = \mathbb{F}^2$. For the node $N4$, the matrices $S_{4,j}$'s are not equal. In fact $S_{4,j} = (1, 2)$ for $j = 1, 2, 3, 5$ and $S_{4,6} = (1, 3)$.

7.3 Code Constructions with Two Parities

In this section, we are going to construct codes with column length $l = 2^m$, $k = 3m$ systematic nodes, and $r = 2$ parity nodes. Here m is some integer. As we showed in the previous section, we can assume the coding matrices are

$$\begin{pmatrix} I & \cdots & I \\ A_1 & \cdots & A_k \end{pmatrix}, \tag{7.5}$$

where $A_{1,i} = I$ and $A_{2,i} = A_i$ correspond to parity 1 and 2, respectively.

Now we only need to find coding matrices A_i's, and subspaces S_i's. For now we only care about eigenvectors of A_i, not its eigenvalues because eigenvectors determine the repair bandwidth. Later we will show that using a large enough finite field, we can choose the eigenvalues such that the code is indeed MDS. In the following construction, for any $i \in [k]$, A_i has two different eigenvalues $\lambda_{i,0}, \lambda_{i,1}$, each corresponding to $l/2 = 2^{m-1}$ eigenvectors. Denote these eigenvectors as

$$V_{i,0} = \begin{pmatrix} v_{i,1} \\ v_{i,2} \\ \vdots \\ v_{i,l/2} \end{pmatrix}$$

for eigenvalue $\lambda_{i,0}$, and

$$V_{i,1} = \begin{pmatrix} v_{i,l/2+1} \\ v_{i,l/2+2} \\ \vdots \\ v_{i,l} \end{pmatrix}$$

for eigenvalue $\lambda_{i,1}$. Therefore, A_i can be computed as

$$A_i = \begin{pmatrix} V_{i,0} \\ V_{i,1} \end{pmatrix}^{-1} \begin{pmatrix} \lambda_{i,0} I_{\frac{l}{2}\times\frac{l}{2}} & \\ & \lambda_{i,1} I_{\frac{l}{2}\times\frac{l}{2}} \end{pmatrix} \begin{pmatrix} V_{i,0} \\ V_{i,1} \end{pmatrix}.$$

By abuse of notations, we also use $V_{i,0}, V_{i,1}$ to represent the eigenspace corresponding to $\lambda_{i,0}, \lambda_{i,1}$, respectively. Namely, $V_{i,0} = \text{span}\{v_{i,1}, \ldots, v_{i,l/2}\}$ and $V_{i,1} = \text{span}\{v_{i,l/2+1}, \ldots, v_{i,l}\}$.

When a systematic node i is erased, $i \in [k]$, we are going to use S_i to rebuild it. The **subspace**

property becomes

$$S_i = S_i A_j, \quad \forall j \neq i, j \in [k], \tag{7.6}$$

$$S_i + S_i A_i = \mathbb{F}^l. \tag{7.7}$$

In the following construction, e_a, $a \in [0, l-1]$, are some basis of $\mathbb{F}^l$, for example, one can think of them as the standard basis. The subscript a is represented by its binary expansion, $a = (a_1, a_2, \ldots, a_m)$. For example, if $l = 16, m = 4, a = 5$, then $e_5 = e_{(0,1,0,1)}$ and $a_1 = a_3 = 0, a_2 = a_4 = 1$.

In order to construct the code, we first define 3 sets of vectors for $i \in [m]$:

$$\begin{aligned} P_{i,0} &= \{e_a : a_i = 0\}, \\ P_{i,1} &= \{e_a : a_i = 1\}, \\ Q_i &= \{e_a + e_b : a_i + b_i = 1, a_j = b_j, \forall j \neq i\}. \end{aligned}$$

For example, if $m = 2, i = 1$, then $P_{1,0} = \{e_{(0,0)}, e_{(0,1)}\} = \{e_0, e_1\}$, $P_{1,1} = \{e_{(1,0)}, e_{(1,1)}\} = \{e_2, e_3\}$, and $Q_1 = \{e_{(0,0)} + e_{(1,0)}, e_{(0,1)} + e_{(1,1)}\} = \{e_0 + e_2, e_1 + e_3\}$. **Notation**: The subscript i for sets $P_{i,u}, Q_i$ and a_i (the i-th digit of vector a) is written modulo m. For example, if $i \in [tm+1, (t+1)m]$ for some integer t, then $P_{i,u} := P_{i-tm,u}$.

Construction 7.1 *The $(n = 3m + 2, k = 3m, l = 2^m)$ code has coding matrices A_i, $i \in [k]$, each with two distinct eigenvalues, and eigenvectors $V_{i,0}, V_{i,1}$. When node i is erased, we are going to use S_i to rebuild. We construct the code as follows:*

1. *For $i \in [m]$, $V_{i,0} = \text{span}(Q_i)$, $V_{i,1} = \text{span}(P_{i,1})$, $S_i = \text{span}(P_{i,0})$.*
2. *For $i \in [m+1, 2m]$, $V_{i,0} = \text{span}(P_{i,0})$, $V_{i,1} = \text{span}(Q_i)$, $S_i = \text{span}(P_{i,1})$.*
3. *For $i \in [2m+1, 3m]$, $V_{i,0} = \text{span}(P_{i,0})$, $V_{i,1} = \text{span}(P_{i,1})$, $S_i = \text{span}(Q_i)$.*

Example 7.2 *Deleting the node N4, Figure* 7.1 *is a code using Construction* 7.1 *and $l = 2$. Another example of $l = 4$ is shown in Figure* 7.2*. One can check* (7.6) *holds. For instance, $S_1 = \text{span}\{e_0, e_1\} = \text{span}\{e_0 + e_1, e_1\}$ is an invariant subspace of A_2. So $S_1 = S_1 A_2$. If the two eigenvalues of A_i are distinct, it is easy to show that $S_i \oplus S_i A_i = \mathbb{F}^4$, $\forall i \in [6]$.*

The above example shows that for $m = 1, 2$, the constructed code has optimal repair. It is true in general, as the following theorem suggests.

	N1	N2	N3	N4	N5	N6
1st eigenspace of A_i	e_0+e_2 e_1+e_3	e_0+e_1 e_2+e_3	e_0 e_1	e_0 e_2	e_0 e_1	e_0 e_2
2nd eigenspace of A_i	e_2 e_3	e_1 e_3	e_0+e_2 e_1+e_3	e_0+e_1 e_2+e_3	e_2 e_3	e_1 e_3
S_i	e_0 e_1	e_0 e_2	e_2 e_3	e_1 e_3	e_0+e_2 e_1+e_3	e_0+e_1 e_2+e_3

Figure 7.2: $(n=8,k=6,l=4)$ code. The first parity node is assumed to be the row sum, and the second parity is computed using coding matrices A_i. In order to rebuild node i, S_i is multiplied to each surviving node. The first $2m=4$ nodes have optimal access, and the last $m=2$ nodes have optimal update.

Theorem 7.3 *Construction* 7.1 *is a code with optimal repair-bandwidth* $1/2$ *for rebuilding any systematic node.*

Proof: By symmetry of the first two cases in the construction, we are only going to show that the rebuilding of node i, $i \in [m] \cup [2m+1, 3m]$ is optimal. Namely, the subspace property (7.6)(7.7) is satisfied. Recall that $S_iA_j = S_i$ is equivalent to S_i being an invariant subspace of A_j.
Case 1: $i \in [m]$.

- When $j \in [tm+1,(t+1)m], j-tm \neq i, t \in \{0,1\}$, define $B = \{e_a : a_j = 1-t, a_i = 0\} \cup \{e_a+e_b : a_j+b_j = 1, a_i = b_i = 0, a_z = b_z, \forall z \neq i,j\}$. Then it is easy to see that $S_i = \text{span}(P_{i,0}) = \text{span}(B)$. Moreover, each vector in set B is an eigenvector of A_j, therefore S_i is an invariant subspace of A_j.

- When $j-m=i$, $S_i = V_{j,0} = \text{span}(P_{i,0})$, so S_i is an eigenspace of A_j.

- When $j \in [2m+1, 3m]$, we can see that every vector in $P_{i,0}$ is a vector in $V_{j,0} = \text{span}(P_{j,0})$ or in $V_{j,1} = \text{span}(P_{j,1})$, hence it is an eigenvector of A_j.

- When $j = i$, consider a vector $e_a \in P_{i,0}$, then $a_i = 0$. And $e_a = (e_a+e_b) - e_b$ where $b_i = 1, b_j = a_j$ for all $j \neq i$. Here both e_a+e_b and e_b are eigenvectors of A_i.

$$\begin{aligned} e_aA_i &= (e_a+e_b)A_i - e_bA_i \\ &= \lambda_{i,0}(e_a+e_b) - \lambda_{i,1}e_b \\ &= (\lambda_{i,0}-\lambda_{i,1})e_b + \lambda_{i,0}e_a. \end{aligned}$$

Because $\lambda_{i,0} \neq \lambda_{i,1}$, we get $\text{span}\{e_aA_i, e_a\} = \text{span}(e_a, e_b)$. Hence $S_iA_i + S_i = \text{span}\{e_a, e_b :$

$a_i = 0, b_i = 1, a_j = b_j, \forall j \neq i\} = \mathbb{F}^l$.

Case 2: $i \in [2m+1, 3m]$.

- When $j = i - m$ or $j = i - 2m$, $S_i = \text{span}(Q_i)$ is an eigenspace of A_j.
- When $j \in [tm+1, (t+1)m]$, and $j \neq i - tm$ for $t \in \{0, 1\}$, define $D = \{e_a + e_b : a_j = b_j = 1 - t, a_i + b_i = 1, a_z = b_z, \forall z \neq i, j\} \cup \{e_a + e_b + e_c + e_d : a_j = b_j = 0, c_j = d_j = 1, a_i + b_i = 1, c_i + d_i = 1, a_z = b_z = c_z = d_z, \forall z \neq i, j\}$. We can see that $S_i = \text{span}(Q_i) = \text{span}(D)$ and every vector in D is an eigenvector of A_j.
- When $j \in [2m+1, 3m], j \neq i$. We can see that $Q_i = \{e_a + e_b : a_j = b_j = 0, a_i + b_i = 1, a_z = b_z, \forall z \neq i, j\} \cup \{e_a + e_b : a_j = b_j = 1, a_i + b_i = 1, a_z = b_z, \forall z \neq i, j\}$. Apparently, every vector in Q_i is a sum of two vectors in $P_{j,0}$ or two vectors in $P_{j,1}$. So $S_i = \text{span}(Q_i)$ is an invariant subspace of A_j.
- When $j = i$, consider any $e_a + e_b \in Q_i$, where $a_i = 1, b_i = 0, a_z = b_z, \forall z \neq i$. We have

$$(e_a + e_b)A_i = \lambda_{i,1} e_a + \lambda_{i,0} e_b.$$

  Because $\lambda_{i,0} \neq \lambda_{i,1}$, we get $\text{span}\{(e_a + e_b)A_i, e_a + e_b\} = \text{span}\{e_a, e_b\}$. Thus $S_i A_i + S_i = \text{span}\{e_a, e_b : a_i = 1, b_i = 0, a_z = b_z, \forall z \neq i\} = \mathbb{F}^l$.

□

It should be noted that if we shorten the code and keep only the first $2m$ systematic nodes in the code, then it is actually equivalent to the code in [CHLM11]. The repairing of the first $2m$ nodes does not require computation within each remaining node, since only standard bases are multiplied to the surviving columns (e.g., Figure 7.2). We call such repair *optimal access*. It is shown in [TWB12] that if a code has optimal access, then the code has no more than $2m$ nodes. On the other hand, the shortened code with the last m systematic nodes in the above construction is equivalent to that of [CHL11, PDC11a, TWB11]. Since the coding matrices A_i, $i \in [2m+1, 3m]$ are all diagonal, every information entry is included in only $r+1$ entries in the code. We say such a code has *optimal update*. In [TWB12] it is proven that an optimal-update code with diagonal coding matrices has no more than m nodes. Therefore, our code is a combination of the longest optimal-access code and the longest optimal-update code, which provides tradeoff among access,

update, and the code length. The shortening technique was also used in [SRKR10] in order to get optimal-repair code with different code rates.

In addition, if we try to extend an optimal-access code $\mathcal{C}$ with length $2m$ to a code $\mathcal{D}$ with length k, so that $\mathcal{C}$ is a shortened code of $\mathcal{D}$, then the following theorem shows that $k = 3m$ is largest code length. Therefore, our construction is longest in the sense of extending $\mathcal{C}$.

Theorem 7.4 *Any extended code of an optimal-access code of length* $2m$ *will have no more than* $3m$ *systematic nodes.*

Proof: Let $\mathcal{C}$ be an optimal-access code of length $2m$. Let $\mathcal{D}$ be an extended code of $\mathcal{C}$. By equivalently transforming the coding matrices (see [TWB12]), we can always assume the coding matrices of the parities in $\mathcal{D}$ are

$$\begin{pmatrix} I & \cdots & I & I & \cdots & I \\ A_1 & \cdots & A_{2m} & A_{2m+1} & \cdots & A_k \end{pmatrix}.$$

Here the first $2m$ column blocks corresponds to the coding matrices of $\mathcal{C}$. First consider the code $\mathcal{C}$, that is, the first $2m$ nodes. If $\mathcal{C}$ has optimal access, then S_i is the span of $l/2$ standard basis, for $i \in [2m]$. Since there are $2m$ systematic nodes, on average each e_z appears $2m \times \frac{l}{2} \times \frac{1}{l} = m$ times, for $z \in [0, l-1]$. We claim that each e_z appears exactly m times. Otherwise, there exists one e_z that appears in $\{S_i : i \in I\}$, for some $|I| > m, I \subset [2m]$. So $|\cap_{i\in I} S_i| \geq 1$. However, by [TWB12] we know when $|I| > m$, $|\cap_{i\in I} S_i| = 0$. So every e_z, $z \in [0, l-1]$, must appear in m of the S_i's, say $e_z \in S_i, \forall i \in J, |J| = m, J \subset [2m]$. Again by [TWB12] when $|J| = m$, we have $|\cap_{i\in J} S_i| = 1$, so $\cap_{i\in J} S_i = e_z$. So these m subspaces intersect only on e_z.

Now consider the extended code $\mathcal{D}$. Since every S_i, $i \in J$, is an invariant subspace of A_j, $j \in [2m+1, k]$ by the subspace property, we know their intersection, e_z is also an invariant subspace of A_j. In other words, e_z is an eigenvector of A_j. This result is true for all $z \in [0, l-1]$. Hence, we know the standard basis are all the eigenvectors of A_j, $j \in [2m+1, k]$. Equivalently, A_j are all diagonal. So the last $k - 2m$ nodes in $\mathcal{D}$ are optimal update. By [TWB12], there are only m nodes that are all optimal update. So $k \leq 3m$. □

Next let us discuss about the finite-field size of the code. In order to make the code MDS, it is equivalent that we should be able to recover from any two column erasures. In other words, any 1×1 or 2×2 submatrices of the matrix (7.5) should be invertible. Therefore, all eigenvalues $\lambda_{i,s}$ should be nonzero, $i \in [k], s \in \{0, 1\}$. Moreover, the following matrix should be invertible for all

$i \neq j$:

$$\begin{bmatrix} I & I \\ A_i & A_j \end{bmatrix}.$$

Or equivalently, $A_i - A_j$ should be invertible.

Let us first look at an example. Suppose $m = 2, i = 1, j = 2$ (see Figure 7.2), then $A_1 - A_2$ is

$$\begin{bmatrix} \lambda_{1,0} - \lambda_{2,0} & \lambda_{2,1} - \lambda_{2,0} & \lambda_{1,0} - \lambda_{1,1} & 0 \\ 0 & \lambda_{1,0} - \lambda_{2,1} & 0 & \lambda_{1,0} - \lambda_{1,1} \\ 0 & 0 & \lambda_{1,1} - \lambda_{2,0} & \lambda_{2,1} - \lambda_{2,0} \\ 0 & 0 & 0 & \lambda_{1,1} - \lambda_{2,1} \end{bmatrix} \tag{7.8}$$

We can simply compute the determinant by expanding along the first column and the last row. The remaining 2×2 submatrix in the middle is diagonal:

$$\begin{bmatrix} \lambda_{1,0} - \lambda_{2,1} & 0 \\ 0 & \lambda_{1,1} - \lambda_{2,0} \end{bmatrix} \tag{7.9}$$

Hence, the determinant $\det(A_1 - A_2)$ is

$$(\lambda_{1,0} - \lambda_{2,0})(\lambda_{1,0} - \lambda_{2,1})(\lambda_{1,1} - \lambda_{2,0})(\lambda_{1,1} - \lambda_{2,1}).$$

For another example, let $m = 2, i = 1, j = 3$, then $A_1 - A_3$ is

$$\begin{bmatrix} \lambda_{1,0} - \lambda_{3,0} & 0 & \lambda_{1,0} - \lambda_{1,1} & 0 \\ 0 & \lambda_{1,0} - \lambda_{3,0} & 0 & \lambda_{1,0} - \lambda_{1,1} \\ \lambda_{3,0} - \lambda_{3,1} & 0 & \lambda_{1,1} - \lambda_{3,1} & 0 \\ 0 & \lambda_{3,0} - \lambda_{3,1} & 0 & \lambda_{1,1} - \lambda_{3,1} \end{bmatrix} \tag{7.10}$$

Since we can permutate rows and columns of a matrix and not change its rank, the above matrix can be changed into:

$$\begin{bmatrix} \lambda_{1,0} - \lambda_{3,0} & \lambda_{1,0} - \lambda_{1,1} & 0 & 0 \\ \lambda_{3,0} - \lambda_{3,1} & \lambda_{1,1} - \lambda_{3,1} & 0 & 0 \\ 0 & 0 & \lambda_{1,0} - \lambda_{3,0} & \lambda_{1,0} - \lambda_{1,1} \\ 0 & 0 & \lambda_{3,0} - \lambda_{3,1} & \lambda_{1,1} - \lambda_{3,1} \end{bmatrix}. \tag{7.11}$$

And its determinant is

$$\det(A_1 - A_3) = (\lambda_{1,0} - \lambda_{3,1})^2(\lambda_{3,0} - \lambda_{1,1})^2.$$

Now let us discuss in general the finite-field size of the code.

Construction 7.5 *Let the elements of the code be over $\mathbb{F}_q$, with $q \geq 2m+1$. Let c be a primitive element in $\mathbb{F}_q$ and write $< i >:= i \mod m$. Assign the eigenvalues of the coding matrices to be*

$$\lambda_{i,s} = \begin{cases} c^{<i>+sm}, & i \in [2m] \\ c^{<i>+(1-s)m}, & i \in [2m+1, 3m] \end{cases} \tag{7.12}$$

If we have an extra systematic column with $A_{3m+1} = I$ (see column $N4$ in Figure 7.1), we can use a field of size $2m+2$ and simply modify the above construction by

$$\lambda_{i,s} = \begin{cases} c^{<i>+sm+1}, & i \in [2m] \\ c^{<i>+(1-s)m+1}, & i \in [2m+1, 3m] \end{cases}$$

For example, when $m = 1$, the coefficients in Figure 7.1 are assigned using the above formula, where the field size is 4 and $c = 2$. For another example, if $m = 2$, we can use finite field $\mathbb{F}_5$ and $c = 2$, then assign the eigenvalues to be

$$(\lambda_{1,0}, \ldots, \lambda_{6,0}) = (1, 2, 1, 2, 4, 3),$$

$$(\lambda_{1,1}, \ldots, \lambda_{6,1}) = (4, 3, 4, 3, 1, 2).$$

Theorem 7.6 *The above construction guarantees that the constructed code is MDS and has optimal repair-bandwidth. The finite-field size is $q \geq 2m+1$.*

Proof: We claim that if we check any two indices $i \neq j \in [3m]$, then the following conditions are necessary and sufficient for $A_i - A_j$ to be invertible. Assume $r, s \in \{0, 1\}$.

1. $\lambda_{i,s} \neq \lambda_{j,r}$, for any $i \neq j \mod m$.
2. $\lambda_{i,s} \neq \lambda_{j,1-s}$, for $i \in [m], j = i + m$.
3. $\lambda_{i,s} \neq \lambda_{j,s}$, for $i \in [2m], j \in [2m+1, 3m], i = j \mod m$.

If we have an extra systematic column with $A_{3m+1} = I$, then $A_i - I$ is invertible iff

4) $\lambda_{i,s} \neq 1$.

By the proof of Theorem 7.3 we already know that optimal repair-bandwidth is equivalent to

5) $\lambda_{i,0} \neq \lambda_{i,1}$.

It can be easily checked that the above conditions are satisfied by Construction 7.5. Here we only prove condition 1 for $i, j \in [m]$ and condition 2. The rest cases all follow similar ideas. Without loss of generality we can assume $\{e_i\}$ is standard basis, because the basis will not change the value of $\det(A_i - A_j)$.

When $i, j \in [m]$, $V_{i,0} = Q_i, V_{i,1} = P_{i,1}$, and $V_{j,0} = Q_j, V_{j,1} = P_{i,1}$. So $V_{i,1}, V_{j,1}$ share the same eigenvectors $B = \{e_a : a_i = a_j = 1\}$. If we view each element in B as an integer in $[0, 2^m - 1]$ (each vector in B is the binary representation of an integer), we can say A_i, A_j both have only one nonzero element in each row in B. On the other hand, columns of V_i^{-1}, V_j^{-1} correspond to the right eigenvectors of A_i, A_j, respectively. And it is easy to show that they share the right eigenvectors $C = \{e_a^T : a_i = a_j = 0\}$, where the superscript T means transpose. Hence, A_i, A_j both have only one nonzero element in each column in C. To compute the determinant of $A_i - A_j$, we can expand along rows B and columns C. The remaining submatrix will be diagonal since we already eliminated all the non-diagonal elements. Then it is easy to verify condition 1. See (7.8)(7.9) for an example.

When $i \in [m], j = i + m$, $V_{i,0} = Q_i, V_{i,1} = P_{i,1}$ and $V_{j,0} = P_{i,0}, V_{j,1} = Q_i$. Therefore both A_i, A_j have nonzero elements at the diagonal locations. Also A_i has nonzero elements at row $P_{i,0}$ and column $P_{i,1}$. Similarly A_j has nonzero elements at row $P_{i,1}$ and column $P_{i,0}$. Let $a = (0, \ldots, 0, 1, 0, \ldots, 0)$ be a binary vector of length m and the only '1' is at location i. And let us view e_0, e_a as the corresponding integers $0, 2^{m-i}$. Then we can see that rows $\{e_0, e_a\}$ and columns $\{e_0, e_a\}$ have only four nonzero elements. We can permutate the rows/columns of a matrix and not change its rank. Therefore move these two rows/columns to rows/columns $0, 1$, and we get a block diagonal matrix. Following the same procedure, we will get block diagonal matrix, where each block is of size 2×2. And the determinant is simple to compute. See (7.10)(7.11) for an example.

□

We can see that the field size q is about $2/3$ of the number of systematic nodes and is not a constant. Also the code has parameters $(n = 3m + 2, k = 3m, l = 2^m)$. On the other hand, the

$(n = m+3, k = m+1, l = 2^m)$ code in [TWB13] has constant field of size $q = 3$. So the proposed code has longer k but longer (actual) column length $l \log q$ as well. Nonetheless, it may be possible to alter the structure of A_i's a bit (for example, do not require A_i to be diagonalizable) and obtain a constant field size. And this will be one of our future work directions.

7.4 Codes with Arbitrary Number of Parities

In this section, we will give constructions of codes with arbitrary number of parity nodes. Our code will have $l = r^m$ rows, $k = (r+1)m$ systematic nodes, and r parity nodes, for any $r \geq 2$, $m \geq 1$.

Suppose $A_{s,i}$ is the coding matrix for parity node $k+s$ and information node i. From Section 7.2, we assume $A_{1,i} = I$ for all i. In our construction, we are going to add the following assumptions. Every $A_{s,i}$ has r distinct eigenvalues, each corresponding to $l/r = r^{m-1}$ linearly independent eigenvectors, for $s \in [2,r]$. Moreover, given an information node $i \in [k]$, all matrices $A_{s,i}$, $s \in [2,r]$, share the same eigenspaces $V_{i,0}, V_{i,1}, \ldots, V_{i,r-1}$. If these eigenspaces correspond to eigenvalues $\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,r-1}$ for $A_{2,i}$, then we assume they correspond to eigenvalues $\lambda_{i,0}^{s-1}, \lambda_{i,1}^{s-1}, \ldots, \lambda_{i,r-1}^{s-1}$ for $A_{s,i}$. By abuse of notations, $V_{i,u}$ represents both the eigenspace and the $l/r \times l$ matrix containing l/r independent eigenvectors. Under these assumptions, it is easy to see that if we write $A_{s,i}$ as

$$\begin{pmatrix} V_{i,0} \\ \vdots \\ V_{i,r-1} \end{pmatrix}^{-1} \begin{pmatrix} \lambda_{i,0}^{s-1} I & & \\ & \ddots & \\ & & \lambda_{i,r-1}^{s-1} I \end{pmatrix} \begin{pmatrix} V_{i,0} \\ \vdots \\ V_{i,r-1} \end{pmatrix},$$

where the identity matrices are of size $\frac{l}{r} \times \frac{l}{r}$, then $A_{s,i} = A_{2,i}^{s-1}$, for all $s \in [r]$. Hence, we are going to write $A_i = A_{2,i}$, thus $A_{s,i} = A_i^{s-1}$, and our construction will only focus on the matrix A_i. As a result, the **subspace property** becomes

$$S_i = S_i A_j, \forall j \neq i, j \in [k] \tag{7.13}$$

$$S_i + S_i A_i + S_i A_i^2 + \cdots + S_i A_i^{r-1} = \mathbb{F}^l \tag{7.14}$$

Note that such choice of eigenvalues is not the unique way to construct the matrices, but it guarantees that the code has optimal repair-bandwidth. Also, when the finite-field size is large

enough, we can find appropriate values of $\lambda_{i,u}$'s such that the code is MDS. At last, since each $V_{i,u}$ has dimension l/r and corresponds to l/r independent eigenvectors, we know that any vector in the subspace $V_{i,u}$ is an eigenvector of A_i.

Let $\{e_0, e_1, \ldots, e_{r^m-1}\}$ be the standard basis of $\mathbb{F}^l$. And we are going to use the r-ary expansion to represent the index of a base. An index $a \in [0, r^m - 1]$ is written as $a = (a_1, a_2, \ldots, a_m)$, where a_i is its i-th digit. For example, when $r = 3, m = 4$, we have $e_5 = e_{(0,0,1,2)}$. Define for $i \in [k], u \in [0, r-1]$ the following sets of vectors:

$$\begin{aligned} P_{i,u} &= \{e_a : a_i = u\}, \\ Q_i &= \{\sum_{a_i=0}^{r-1} e_a : a_j \in [0, r-1], j \neq i\}. \end{aligned}$$

So $P_{i,u}$ is the set of bases whose index is u in the i-th digit. The sum in Q_i is over all e_a such that the j-th digit of a is some fixed value for all $j \neq i$, and the i-th digit varies in $[0, r-1]$. In other words, a vector in Q_i is the summation of the corresponding bases in $P_{i,u}, \forall u$. For example, when $r = 3, m = 2$, $P_{1,0} = \{e_{(0,0)}, e_{(0,1)}, e_{(0,2)}\} = \{e_0, e_1, e_2\}$, $P_{1,1} = \{e_3, e_4, e_5\}$, $P_{1,2} = \{e_6, e_7, e_8\}$, and $Q_1 = \{e_0 + e_3 + e_6, e_1 + e_4 + e_7, e_2 + e_5 + e_8\}$.

Notations: If $a = (a_1, a_2, \ldots, a_m)$ is an r-ary vector, denote by $a_i(u) = (a_1, \ldots, a_{i-1}, u, a_{i+1}, \ldots, a_m)$ the vector that is the same as a except digit i, $u \in [0, r-1]$. In the following, all of the subscript i for sets $P_{i,u}, Q_i$ and for digit a_i are computed modulo m. For example, if $i \in [tm+1, (t+1)m]$ for some integer t, then $Q_i := Q_{i-tm}$.

Construction 7.7 *The $(n = (r+1)m + r, k = (r+1)m, l = r^m)$ code is constructed as follows. For information node $i \in [tm+1, (t+1)m]$, $t \in [0, r-1]$, the u-th eigenspace ($u \in [0, r-1]$) of coding matrix A_i and the rebuilding subspace S_i are defined as*

$$\begin{aligned} V_{i,u} &= \text{span}(P_{i,u}), \forall u \neq t, \\ V_{i,t} &= \text{span}(Q_i), \\ S_i &= \text{span}(P_{i,t}). \end{aligned}$$

For information node $i \in [rm+1, (r+1)m]$, the eigenspaces and rebuilding subspaces are

$$\begin{aligned} V_{i,u} &= \text{span}(P_{i,u}), \forall u \in [0, r-1] \\ S_i &= \text{span}(Q_i). \end{aligned}$$

i	$P_{i,0}$	$P_{i,1}$	$P_{i,2}$	Q_i
1	e_0	e_3	e_6	$e_0+e_3+e_6$
	e_1	e_4	e_7	$e_1+e_4+e_7$
	e_2	e_5	e_8	$e_2+e_5+e_8$
2	e_0	e_1	e_2	$e_0+e_1+e_2$
	e_3	e_4	e_5	$e_3+e_4+e_5$
	e_6	e_7	e_8	$e_6+e_7+e_8$

Figure 7.3: Sets of vectors used to construct a code with $r = 3$ parities and column length $l = 3^2 = 9$.

i	1	2	3	4	5	6	7	8
$V_{i,0}$	Q_1	Q_2	$P_{1,0}$	$P_{2,0}$	$P_{1,0}$	$P_{2,0}$	$P_{1,0}$	$P_{2,0}$
$V_{i,1}$	$P_{1,1}$	$P_{2,1}$	Q_1	Q_2	$P_{1,1}$	$P_{2,1}$	$P_{1,1}$	$P_{2,1}$
$V_{i,2}$	$P_{1,2}$	$P_{2,2}$	$P_{1,2}$	$P_{2,2}$	Q_1	Q_2	$P_{1,2}$	$P_{2,2}$
S_i	$P_{1,0}$	$P_{2,0}$	$P_{1,1}$	$P_{2,1}$	$P_{1,2}$	$P_{2,2}$	Q_1	Q_2

Figure 7.4: An $(n = 11, k = 8, l = 9)$ code. Sets $P_{i,u}$ and Q_i are listed in Figure 7.3. $V_{i,u}$ is the u-th eigenspace of the coding matrix A_i. S_i is the subspace used to rebuild systematic node i.

Example 7.8 *Figure* 7.3 *illustrated the subspaces $P_{i,u}, Q_i$ for $r = 3$ parities and column length $l = 9$. Figure* 7.4 *is a code constructed from these subspaces and has* 8 *systematic nodes. One can see that if a node is erased, one can transmit only a subspace of dimension* 3 *to rebuild, which corresponds to only* $1/3$ *repair bandwidth fraction. The three coding matrices for systematic node i are I, A_i, A_i^2, for $i \in [8]$.*

The following theorem shows that the code indeed has optimal repair-bandwidth $1/r$.

Theorem 7.9 *Construction* 5.3 *has optimal repair-bandwidth* $1/r$ *when rebuilding one systematic node.*

Proof: By symmetry of the construction, we are only going to show that the subspace property (7.13)(7.14) is satisfied for $i \in [1, m] \cup [rm+1, (r+1)m]$. Also $S_i A_j = S_i$ implies that S_i has a basis that are all eigenvectors of A_j.

Case 1: $i \in [1, m]$. Before we begin to explore the different cases, let us define the following sets of vectors

$$\begin{aligned} B_u &= \{e_a : a_i = 0, a_j = u\}, u \in [0, r-1], \\ C_t &= \{\sum_{a_j=0}^{r-1} e_a : a_i = 0, a_z \in [0, r-1], z \neq i, j\}. \end{aligned}$$

In the definition of C_t, the sum is over all e_a such that the i-th digit of a is 0, the z-th digit is some fixed value, $z \neq i, j$, and the j-th digit varies in $[0, r-1]$. Then one can see that

$$B_u \subset P_{j,u}, C_t \subset Q_j.$$

- $j \in [tm+1, (t+1)m]$, for some $t \in [0, r-1]$ and $j - tm \neq i$. Then the eigenspaces of A_j are $V_{j,u} = \text{span}(P_{j,u})$, $u \neq t$, and $V_{j,t} = \text{span}(Q_j)$. Then it is clear that $S_i = \text{span}(P_{i,0}) = \text{span}(\{B_u : u \neq t\} \cup C_t)$. Also every vector of $B_u, u \neq t$ and C_t is an eigenvector of A_j.

- $j \in [rm+1, (r+1)m]$, $j - rm \neq i$. The eigenspaces of A_j are $V_{j,u} = \text{span}(P_{j,u})$, $u \in [0, r-1]$. And $S_i = \text{span}(P_{i,0}) = \text{span}\{B_u : u \in [0, r-1]\}$ and every vector in $B_u, \forall u$ is an eigenvector of A_j.

- $j - tm = i$, $t \in [1, r]$. Then the first eigenspace of A_j is $V_{j,0} = \text{span}(P_{i,0}) = S_i$.

- $j = i$. In this case we want to check (7.14) in the subspace property. Suppose the distinct eigenvalues of A_i are $\lambda_0, \lambda_1, \ldots, \lambda_{r-1}$. Then the eigenvalues for A_i^s will be $\lambda_0^s, \lambda_1^s, \ldots, \lambda_{r-1}^s$, for $s \in [0, r-1]$. Notice that $S_i = \text{span}(P_{i,0}) = \text{span}\{e_{a_i(0)} : \forall a \in \mathbb{Z}_r^m\}$ and

$$\begin{aligned} & e_{a_i(0)} A_i^s \\ = & (\sum_{u=0}^{r-1} e_{a_i(u)} - e_{a_i(1)} - \cdots - e_{a_i(r-1)}) A_i \\ = & \lambda_0^s \sum_{u=0}^{r-1} e_{a_i(u)} - \lambda_1^s e_{a_i(1)} - \cdots - \lambda_{r-1}^s e_{a_i(r-1)} \\ = & \lambda_0^s e_{a_i(0)} + \sum_{u=1}^{r-1} (\lambda_0^s - \lambda_u^s) e_{a_i(u)}. \end{aligned}$$

Writing the equations for all $s \in [0, r-1]$ in a matrix, we get

$$\begin{pmatrix} e_{a_i(0)} \\ e_{a_i(0)} A_i \\ e_{a_i(0)} A_i^2 \\ \vdots \\ e_{a_i(0)} A_i^{r-1} \end{pmatrix} = M \begin{pmatrix} e_{a_i(0)} \\ e_{a_{i(1)}} \\ \vdots \\ e_{a_i(r-1)} \end{pmatrix},$$

with

$$M = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \lambda_0 & \lambda_0 - \lambda_1 & \cdots & \lambda_0 - \lambda_{r-1} \\ \lambda_0^2 & \lambda_0^2 - \lambda_1^2 & \cdots & \lambda_0^2 - \lambda_{r-1}^2 \\ \vdots & \vdots & & \vdots \\ \lambda_0^{r-1} & \lambda_0^{r-1} - \lambda_1^{r-1} & \cdots & \lambda_0^{r-1} - \lambda_{r-1}^{r-1} \end{pmatrix}.$$

After a sequence of elementary column operations, M becomes the following Vandermonde matrix

$$M' = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \lambda_0 & \lambda_1 & \cdots & \lambda_{r-1} \\ \lambda_0^2 & \lambda_1^2 & \cdots & \lambda_{r-1}^2 \\ \vdots & \vdots & & \vdots \\ \lambda_0^{r-1} & \lambda_1^{r-1} & \cdots & \lambda_{r-1}^{r-1} \end{pmatrix}.$$

Since λ_i's are distinct, we know M' and hence M is non-singular. Therefore, $\text{span}\{e_{a_i(0)}, e_{a_i(0)}A_i, \ldots, e_{a_i(0)}A_i^{r-1}\} = \text{span}\{e_{a_i(0)}, e_{a_i(1)}, \ldots, e_{a_i(r-1)}\}$. Since S_i contains $e_{a_i(0)}$ for all r-ary vector a, we know $S_i + S_iA_i + \cdots + S_iA_i^{r-1} = \mathbb{F}^l$.

Case 2: $i \in [rm, (r+1)m]$. Again, we first define some sets of vectors to help with our arguments.

$$\begin{aligned} B'_u &= \{\sum_{a_i=0}^{r-1} e_a : a_j = u, a_z \in [0, r-1], z \neq i, j\} \\ C'_t &= \{\sum_{a_i=0}^{r-1}\sum_{a_j=0}^{r-1} e_a : a_z \in [0, r-1], z \neq i, j\}. \end{aligned}$$

Here the sum in B'_u has fixed values of $a_j = u$ and a_z, $z \neq i, j$, and the i-th digit varies in $[0, r-1]$. The sum in C'_t has fixed values of a_z, $z \neq i, j$, and the i-th and j-th digit both vary in $[0, r-1]$. Then one can check that

$$B'_u \subset \text{span}(P_{j,u}), C'_t \subset \text{span}(Q_j).$$

- $j \in [tm+1, (t+1)m]$, $t \in [0, r-1]$, and $j - tm \neq i - rm$. The eigenspaces of A_j are $\text{span}(P_{j,u})$, $u \neq t$ and $\text{span}(Q_j)$. And $S_i = \text{span}(Q_i) = \text{span}(\{B'_u : u \neq t\} \cup C'_u)$. We can see that every vector in $B'_u, u \neq t$ and C'_t is an eigenvector of A_j.

- $j \in [rm+1, (r+1)m]$, $j \neq i$. The eigenspaces of A_j are $P_{j,u}$, $u \in [0, r-1]$. And $S_i =$

$\text{span}(Q_i) = \text{span}\{B'_u : u \in [0, r-1]\}$. We can see that every vector of B'_u is an eigenvector of A_j.

- $j - tm = i - rm$, $t \in [0, r-1]$. Then the t-th eigenspace of A_j is $\text{span}(Q_i)$, which is equal to S_i.

- $j = i$. Take $\sum_{u=0}^{r-1} e_{a_i(u)} \in S_i$ for arbitrary a, then

$$\sum_{u=0}^{r-1} e_{a_i(u)} A_i^s = \sum_{u=0}^{r-1} \lambda_u^s e_{a_i(u)}.$$

Written in a matrix form, we have

$$\begin{pmatrix} e_{a_i(0)} \\ e_{a_i(0)} A_i \\ e_{a_i(0)} A_i^2 \\ \vdots \\ e_{a_i(0)} A_i^{r-1} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \lambda_0 & \lambda_1 & \cdots & \lambda_{r-1} \\ \lambda_0^2 & \lambda_1^2 & \cdots & \lambda_{r-1}^2 \\ \vdots & \vdots & & \vdots \\ \lambda_0^{r-1} & \lambda_1^{r-1} & \cdots & \lambda_{r-1}^{r-1} \end{pmatrix} \begin{pmatrix} e_{a_i(0)} \\ e_{a_{i(1)}} \\ \vdots \\ e_{a_i(r-1)} \end{pmatrix}.$$

So similar to Case 1, we know $S_i + S_i A_i + \ldots S_i A_i^{r-1}$ spans the entire space $\mathbb{F}^l$.

□

Again, this construction can be shortened to an optimal-access code of length rm [CHLM11] and an optimal-update code of length m [CHL11, PDC11a, TWB11].

The finite-field size of this code can be bounded by the following theorem. In the following, we do not assume that the eigenvalue of $A_{s,i}$ is the s-th power of $A_{2,i}$, and $A_{1,i}$ is not necessarily identity. Hence, we only assume that $A_{1,i}, \ldots, A_{r,i}$ share the same eigenspaces for all i.

Theorem 7.10 *A finite field of size $k^{r-1}r^{m-1} + 1$ suffices for the code to be MDS and optimal repair-bandwidth. Here $k = (r+1)m$.*

Proof: Let $\{\lambda_{i,j}^{(s)}\}$ be the j-th eigenvalue of $A_{s,i}$, $i \in [k], j \in [0, r-1], s \in [r]$. In order to show that the code is MDS, we need to check if all $x \times x$ submatrices of the following matrix are invertible, for all $x \in [1, r]$:

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,k} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,k} \\ \vdots & \vdots & & \vdots \\ A_{r,1} & A_{r,2} & \cdots & A_{r,k} \end{bmatrix}. \tag{7.15}$$

Note that each $A_{s,i}$ can be written as $V_i^{-1}\Lambda_{s,i}V_i$ for some diagonal matrix $\Lambda_{s,i}$, where the rows of V_i are eigenvectors and the diagonal of $\Lambda_{s,i}$ are eigenvalues. Since $A_{1,i}, \ldots, A_{r,i}$ share the same eigenvectors V_i, we can multiply V_i^{-1} on the right of the i-th block column, and not change the rank of the above matrix:

$$M = \begin{bmatrix} V_1^{-1}\Lambda_{1,1} & V_2^{-1}\Lambda_{1,2} & \cdots & V_k^{-1}\Lambda_{1,k} \\ V_1^{-1}\Lambda_{2,1} & V_2^{-1}\Lambda_{2,2} & \cdots & V_k^{-1}\Lambda_{2,k} \\ \vdots & \vdots & & \vdots \\ V_1^{-1}\Lambda_{r,1} & V_2^{-1}\Lambda_{r,2} & \cdots & V_k^{-1}\Lambda_{r,k} \end{bmatrix}.$$

Here all $\lambda_{i,j}^{(s)}$ are unknowns in the finite field $\mathbb{F}_q$. We are going to show that if we write the determinants of each $x \times x$ submatrix as a polynomial, and take the product of all these polynomials, then it is an nonzero polynomial. Moreover, by Combinatorial Nullstellensatz [Alo99] we can find assignments of the unknowns over a large enough finite field, such that this polynomial is not zero. Then we are guaranteed to have all the $x \times x$ submatrices invertible. In [Alo99] it is proved that if the degree of a polynomial $f(x_1, \ldots, x_s)$ is $\deg(f) = \sum_{i=1}^{s} t_i$, and the coefficient of $\prod_{i=1}^{s} x_i^{t_i}$ is nonzero, then a finite field of size $\max_i\{t_i\}$ is sufficient for an assignment $c_1, \ldots, c_s$ such that $f(c_1, \ldots, c_s) \neq 0$.

By the symmetry of the $\lambda_{i,j}^{(s)}$, we consider only the degree of $\lambda := \lambda_{1,0}^{(1)}$. We will find its maximum degree in the polynomial of determinants. This unknown variable only appears in the matrix

$$\Lambda_{1,1} = \begin{bmatrix} \lambda_{1,0}^{(1)} I & & \\ & \ddots & \\ & & \lambda_{1,r-1}^{(1)} I \end{bmatrix},$$

where I is the identity matrix of size $r^{m-1} \times r^{m-1}$. Let $B = V_1^{-1}\Lambda_{1,1}$. Then we know that λ appears

only in the first r^{m-1} columns of B. For the determinant of any $x \times x$ submatrix of M, only the ones containing B needs to be considered, because we are only interested in the degree of λ. Therefore, there are $\binom{k-1}{x-1}\binom{r-1}{x-1}$ submatrices of size $x \times x$ that has λ in its determinant, and its degree is r^{m-1} for each submatrix. So the total degree of λ is

$$r^{m-1} \sum_{x=1}^{r} \binom{k-1}{x-1}\binom{r-1}{x-1}.$$

Moreover, we know from the proof of Theorem 7.9 that optimal repair-bandwidth is achieved for the first systematic node iff the following matrix is invertible

$$\begin{pmatrix} \lambda_{1,0}^{(1)} & \cdots & \lambda_{1,r-1}^{(1)} \\ \vdots & & \vdots \\ \lambda_{1,0}^{(r)} & \cdots & \lambda_{1,r-1}^{(r)} \end{pmatrix}$$

Hence, we need to multiply its determinant to our polynomial. The total degree of λ is

$$\begin{aligned} & 1 + r^{m-1} \sum_{x=1}^{r} \binom{k-1}{x-1}\binom{r-1}{x-1} \\ = \; & 1 + r^{m-1} \sum_{x=0}^{r-1} \binom{k-1}{x}\binom{r-1}{x} \\ < \; & 1 + r^{m-1} \sum_{x=0}^{r-1} (k-1)^x \binom{r-1}{x} \\ = \; & 1 + r^{m-1} k^{r-1}. \end{aligned}$$

Hence the proof is completed. □

We can see that in the above theorem, for high-rate codes the field size is expositional in the number of systematic nodes. But we believe that there is still a large space to improve this bound.

7.5 Lowering the Access Ratio

In this section, we are going to construct equivalent codes of Construction 5.3, such that the further lower the average number of accesses of a random erasure. Given an (n,k,l) code $\mathcal{C}$, define the *access ratio* as

$$R = \frac{\sum_{i=1}^{k} \#\text{ accesses from all surviving nodes to rebuild } i}{k(n-1)l}.$$

Recall that for the $(n = (r+1)m + r, k = (r+1)m, l = r^m)$ code in the previous section, the access ratio is

$$R = \frac{rm \cdot (n-1)\frac{l}{r} + m \cdot (n-1)l}{(r+1)m \cdot (n-1)l} = \frac{2}{r+1}.$$

First let us define the code and its rebuilding subspaces. We will multiply a block diagonal matrix on the right of (7.15) and get the new coding matrix:

$$\begin{aligned} C &= \begin{bmatrix} C_{2,1} & \cdots & C_{2,k} \\ \vdots & \ddots & \vdots \\ C_{r,1} & \cdots & C_{r,k} \end{bmatrix} \\ = AB &= \begin{bmatrix} A_{2,1} & \cdots & A_{2,k} \\ \vdots & \ddots & \vdots \\ A_{r,1} & \cdots & A_{r,k} \end{bmatrix} \begin{bmatrix} B_1 & & \\ & \ddots & \\ & & B_k \end{bmatrix}. \end{aligned}$$

Namely, for $i \in [r], j \in [k]$ the new coding matrices are

$$C_{i,j} = A_{i,j} B_j, \tag{7.16}$$

where B_j is an invertible matrix of size $l \times l$. For an erased node $i \in [k]$, the rebuilding subspaces are still $S_{i,j} = S_i$ for parity nodes $j \in [k+1, k+r]$, and $S_{i,j} = S_i B_j$ for systematic nodes $j \in [k]$.

Theorem 7.11 *The new coding matrices $C_{i,j}$ and rebuilding subspaces $S_{i,j}$ satisfy the subspace property, and is still an MDS code.*

Proof: We know for $A_{i,j}$, the row spans satisfies for all $j \neq i, j \in [k], t \in [r]$,

$$\text{span}\{S_i\} = \text{span}\{S_i A_{t,j}\}.$$

Therefore, there exists a invertible matrix M of size $l/r \times l/r$ such that the matrices satisfies

$$S_i = M S_i A_{t,j}.$$

Hence

$$S_i B_j = M S_i A_{t,j} B_j,$$

or equivalently

$$\text{span}\{S_i B_j\} = \text{span}\{S_i A_{t,j} B_j\}.$$

By definition, we have

$$\text{span}\{S_{i,j}\} = \text{span}\{S_{i,k+t} C_{t,j}\},$$

so (7.1) is satisfied. We also know that the sum of subspaces satisfies

$$\mathbb{F}^l = \sum_{t=1}^{r} S_i A_{t,i},$$

or

$$\begin{aligned} l &= \text{rank} \begin{bmatrix} S_i A_{1,i} \\ \vdots \\ S_i A_{r,i} \end{bmatrix} = \text{rank} \begin{bmatrix} S_i A_{1,i} \\ \vdots \\ S_i A_{r,i} \end{bmatrix} B_i \\ &= \text{rank} \begin{bmatrix} S_i A_{1,i} B_i \\ \vdots \\ S_i A_{r,i} B_i \end{bmatrix} = \text{rank} \begin{bmatrix} S_{i,k+1} C_{1,i} \\ \vdots \\ S_{i,k+r} C_{r,i} \end{bmatrix}. \end{aligned}$$

Therefore, (7.2) is satisfied:

$$\mathbb{F}^l = \sum_{t=1}^{r} S_{i,k+t} C_{t,i}.$$

Since $C = AB$ and any $x \times x$ subblock matrix of A is invertible, we can see that any $x \times x$ subblock matrix of C is also invertible, for any $x \in [r]$. Therefore, the new code is also MDS. □

Now let us find a code such that the number of accesses will be decreased. Notice that if $S_{i,j} = S_i B_j$ can be written as the span of l/r standard bases, then node j has optimal access when node i is erased. So we need to look for proper B_j's such that the above condition is satisfied by as many (i, j) pairs as possible. In particular, let us assume

$$B_j = V_j^{-1} = \begin{pmatrix} V_{j,0} \\ \vdots \\ V_{j,r-1} \end{pmatrix}^{-1} \tag{7.17}$$

is the inverse of the matrix of the eigenspaces.

Theorem 7.12 *The access ratio of the* $(n = (r+1)m + r, k = (r+1)m, l = r^m)$ *code using*

(7.17) *is*

$$\frac{2-\frac{r-1}{n-1}}{r+1}.$$

Proof:

□

7.6 Conclusions

In this chapter, we presented a family of codes with parameters $(n = (r+1)m + r, k = (r+1)m, l = r^m)$ and they are so far the longest high-rate MDS code with optimal repair. The codes were constructed using eigenspaces of the coding matrices, such that they satisfy the subspace property. This property gives more insights on the structure of the codes, and simplifies the proof of optimal repair.

If we require that the code rate approaches 1, i.e., r being a constant and m goes to infinity, then the column length l is *exponential* in the code length k. However, if we require the code rate to be roughly a constant fraction, i.e., m being a constant and r goes to infinity, then l is *polynomial* in k. Therefore, depending on the application, we can see a tradeoff between the code rate and the code length.

It is still an open problem what is the longest optimal-repair code one can build given the column length l. Also, the bound of the finite-field size used for the codes may not be tight enough. Unlike the constructions in this chapter, the field size may be reduced when we assume that the coding matrices do not have eigenvalues or eigenvectors (are not diagonalizable). These are our future work directions.

Part II

Coding for Flash Memory

Chapter 8

Introduction to Rank Modulation

The most commonly used storage media for computers and data centers has been hard disks for several decades thanks to its low cost and stable performance. However, recently flash memory emerged as a new form of non-volatile storage technology. It has high capacity density, good random access ability, high power efficiency, and promising scalability. Its market has grown noticeably because of the development of mobile devices such cell phones, cameras, and tablets. The requirement of lighter and faster hard drives of personal computers also stimulated the research and technology of flash. Moreover, data centers are facing more and more problems of high energy cost, low random access speed, and large space occupancy with hard disks. Therefore besides the vast demand in the consumer products, flash memory is without doubt one of the most prominent substitute of hard disks for industrial products.

However, flash memory has its own disadvantages. Compared to hard disks, it is more costly per storage unit, the programming process is harder to control, and data is more prone to errors during reading and retention. Facing such problems, coding could be one of the solutions. By representing data differently from current state of art, we are able to store more bits on the same flash device, speed up the programming process, and improve data reliability. The main coding technique we are going to discuss in this part of the thesis is rank modulation.

In flash memories, floating-gate cells use their charge levels to store data [BG07] (See Figure 8.1). For higher capacity, multi-level cells (MLCs) with an increasing number of levels are being developed. To increase a cell level, charge is injected into the cell by the Fowler-Nordheim tunneling mechanism or the hot-electron injection mechanism. To erase a cell, tunnel release mechanism is performed on a whole block (typically 512K cells). Moreover, to lower any cell level, one must erase a whole cell block and reprogram them starting at the lowest level. Block erasure not only costs time and energy, also decreases the lifetime of a device. This asymmetric property caused by

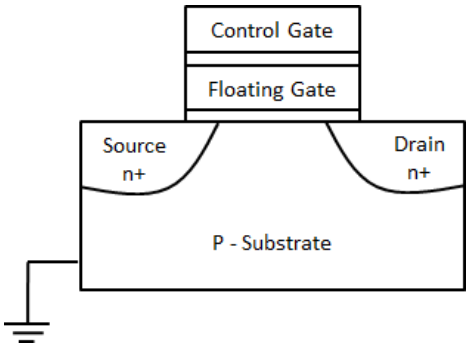

Figure 8.1: A flash memory cell.

block erasure is a prominent feature of flash memories and presents a bottleneck of flash memories in terms of speed and reliability.

Since lowering cell levels is very expensive, the programming process is iterative to avoid over-injection, as we have seen in Figure 1.6. Therefore, writing data requires many programming cycles, and each cell can only support a certain number of levels so that adjacent levels are distinguishable. In addition, multi-level cells may suffer more data errors compared to single-level ones, due to their higher requirement on cell-level resolution. In particular, charge leakage, or threshold level drift in aging cells will cause erroneous data.

Rank modulation is a new information representation scheme that uses the relative order of cell levels to represent data [JMSB09]. Physical realizations were recently presented in [KPT12] and also in [PPM$^+$11] for phase-change memory based on the same idea. Given m cells with different levels, we can induce a permutation of length m from them and use each permutation to represent a message. Therefore, a group of m cells can store $\log_2(m!)$ bits of information. For example, suppose we have $m = 5$ cells and their charge levels are $(1.5, 5.2, 0.3, 4.9, 7.8)$. Notice here all the levels are analog instead of discrete, as in the traditional method. Then the highest cell is 5, and the second highest is 2, and so on. The induced permutation is $(5, 2, 4, 1, 3)$ and in total we can represent 120 messages using 5 cells.

To write a permutation, we can program the lowest cell first, and then the second lowest cell, and so on. Over-injection is no longer a problem because we can always inject more electrons into the higher cells so as to obtain the relative order. As a result, the writing can be completed in a much smaller number of cycles. When level drift occurs, the stored information will be intact as long as the induced permutation is unchanged. Besides, combined with some error-correction techniques rank modulation can detect and correct errors. Another advantage of rank modulation

is that we can correct the errors by injecting more electrons to corresponding cells and no block erasure is necessary. Moreover, since we can take analog values as cell levels, it is possible to fit in more levels in each cell given that different levels are distinguishable. Hence rank modulation may effectively store more information per cell compared to conventional schemes even though it has the constraint that all cells have distinct levels.

Given the charge levels of a group of flash cells, sorting is used to induce a permutation, which in turn represents data. Motivated by the lower sorting complexity of smaller cell groups, we propose *bounded rank modulation* in Chapter 9, where a sequence of permutations of given sizes is used to represent data. In particular, we require that the permutation size is smaller than the sequence size, and the maximum possible levels a cell can support is upper bounded by a constant. For example, suppose we have a total of 8 cells and the maximum possible levels is 6. Then we can for simplicity write charge levels as integers $\{1, 2, \ldots, 6\}$. Further suppose the permutation size is 4. The charge levels of these cells $(1, 2, 3, 4, 5, 6, 2, 3)$ can represent two permutations: $(4, 3, 2, 1)$ and $(2, 1, 4, 3)$ if we group the first four and the last four cells separately. But we can also induce three permutations: $(4, 3, 2, 1), (4, 3, 2, 1), (2, 1, 4, 3)$ if we group the first, the middle, and last four cells. In other words, we can use a sliding window of size four, such that every adjacent two windows overlap by two cells. An interesting observation is that if we allow overlap, then a sequence contains more groups of permutations but at the same time satisfies more constraints (cells in each permutation are distinct and overlapped ranks should be consistent). Therefore, we study the capacity, or the amount of information per cell, of bounded rank modulation under the condition that permutations can overlap by a certain amount.

In Chapter 10 we propose *partial rank modulation*, where only a subset of cells in a permutation represents information. More specifically, we only utilize a certain number of the top cells and leave the rest as redundancy. This scheme also reduces decoding complexity, and at the same time keeps independency of different permutations. For example, if we only compare the top 2 cells out of a group of 5, the cell levels $(1.5, 5.2, 0.3, 4.9, 7.8)$ will result in a partial permutation $(5, 2, |4, 1, 3)$. The ordering of the cells $4, 1, 3$ does not carry information. In total we can represent 20 messages. Hence partial rank modulation gains low complexity with the sacrifice of capacity. We will construct Gray codes on such partial permutations, which is a sequence of permutations, such that every partial permutation appears exactly once and the transition between two adjacent permutations satisfies certain constraints. One possible constraint is that the highest cell level is only increased by a small amount, so that we can write many times before we reach the limit and

erase a block. It is easy to construct a counter from a Gray code. Then viewing a group of physical cells as one logical cell, we can obtain a large number of levels in a single cell. Applying other flash codes where only a small number of levels is increased during each rewrite, we can store information in flash memory with low cost. A related notion in combinatorics is called universal cycle, and each partial permutation appears as a subsequence in this cycle exactly once. We will discuss this problem starting from the universal cycles, and then show that Gray codes and universal cycles are equivalent. Our construction in fact solves an open problem of universal cycles.

Errors can be caused by charge leakage, which is a long-term factor causing the retention problem. Writing and reading disturbances also result in the change of cell levels. An error-correction code enables reliable storage of data by introducing redundant cells. In Chapter 11 we discuss the model of errors in rank modulation, namely, an error occurs when the level of one cell changes and two adjacent elements in the permutation exchange. For instance, if the level of cell 5 in permutation $(5, 2, 4, 1, 3)$ drops between cells 2 and 4, we get a swap: $(2, 5, 4, 1, 3)$. We say there is an error in this permutation. Such an error model is reasonable if the change of cell levels is gradual instead of dramatic. We propose t-error-correction codes based on t-ary error-correction codes on Hamming distances. We are able to construct 1-error-correcting rank modulation codes with half the size of the sphere-packing bound.

There has been a number of recent works using the information-theoretic approach to develop new storage schemes for flash memories. Coding techniques are studied for the traditional absolute-value information representations. Related to Gray codes mentioned above, the first set of work is on coding schemes for rewriting data [BJB07] [FLM08] [JBB07] [JB08] [YVSW08] [YSVW12], where the main idea is to write new message on top of old message. The challenge is that one can only increase levels, and the decoding is independent of previous messages. This idea can be traced back to write-once-memory decades ago [RS82] designed for punch cards and optical disks. Additionally, codes for correcting limited-magnitude errors [CSBB10] considered the problem of having specific kinds of errors in multi-level cells due to the fact that small-magnitude errors are more probable than large ones.

There are also a lot of work on relative-value information representation schemes for flash memories. Besides rank modulation [JMSB09], its variations consider the case of comparing a small number of cells [EGLSB11] which is similar to bounded rank modulation but does not consider the constraint of finite levels in a cell, as well as allowing multiple cells to have identical levels [EGJB12]. Error-correction for rank modulation for adjacent transposition (two adjacent levels

get exchanged) was studied first in [JSB10,JSB08], recently [BM10] derived the capacity for the full range of minimum distance between codewords, and systematic codes were proposed in [ZJB12]. Translocation errors were studied in [FSM12] where an error means dropping the rank of one cell by a certain amount, and [TS10] studied limited-magnitude errors under the infinity norm for rank modulation.

Chapter 9

Bounded Rank Modulation

9.1 Introduction

Rank modulation compares the relative values of m cells to represent information. To induce a permutation from the group of cells, a serial sorting algorithm of complexity $O(m \log m)$ is needed [Knu98]. In some realizations of rank modulation, such as [KPT12], a comparison circuit is built and permutation is obtained in $m-1$ interactions sequentially. In each iteration this circuit finds the highest level, which is then disabled or disconnected from the circuit for the following iterations. In both cases, decoding complexity increases with the size of the permutation. Reducing the sorting complexity is important for the efficient hardware implementation as well as low information access latency of rank modulation. Therefore, we propose rank modulation with bounded permutation sizes.

Let $(c_1, c_2, \cdots, c_m)$ denote the charge levels of m cells, where each c_i (for $1 \leq i \leq m$) is an analog number and all charge levels are distinct: $\forall\ i \neq j,\ c_i \neq c_j$. Let $\mathcal{I}(c_1, c_2, \cdots, c_m) = (a_1, a_2, \cdots, a_m)$ be a function that induces from the charge levels a permutation, where for $i = 1, 2, \cdots, m$,

$$a_i = |\{j | c_j \leq c_i, j = 1, 2, \cdots, m\}|.$$

Notice here that we define a_i as the number of cells not higher than cell i and $1 \leq a_i \leq m$. As a result, the induced permutation of integers in $\{1, 2, \ldots, m\}$ has the same relative ranks as the cell levels:

$$(a_1, a_2, \cdots, a_m) = \mathcal{I}(c_1, c_2, \cdots, c_m) = \mathcal{I}(a_1, a_2, \cdots, a_m).$$

For example, if $m = 4$ and $(c_1, c_2, c_3, c_4) = (0.2, 0.3, 1.2, 0.5)$, then the induced permutation is $(a_1, a_2, a_3, a_4) = (1, 2, 4, 3)$.

In Chapter 8 we defined the induced permutation in a different way. We define $\mathcal{J}(c_1, c_2, \ldots, c_m) = (b_1, b_2, \ldots, b_m)$ as the alternative permutation where b_i indicates the i-th highest cell index. Namely, b_i satisfies

$$|\{j|c_j \geq c_{b_i}, j = 1, 2, \cdots, m\}| = i.$$

It is not difficult to see that these two permutation definitions are equivalent, since they both provides all the information about the relative ranks of the cells. In our discussions on rank modulation, we will use both of them. In this chapter, we will only use the function $\mathcal{I}$. In the next two chapters, we will use the function $\mathcal{J}$.

To study the capacity under this constraint, we propose a discrete model. Normalize the gap between the minimum and maximum charge levels of the memory to 1, and let δ denote the minimum charge difference to distinguish two levels. Then the largest possible size for a permutation is $D = \lfloor \frac{1}{\delta} \rfloor + 1$. However, in practice the permutation size should be smaller than D not only to reduce the sorting complexity, but also to make cell programming efficiently implementable. In this chapter, we let $m \leq D$ denote the given permutation size which is also the number of cells in a group. Each cell level is denoted by an integer in the set $\{1, 2, \cdots, D\}$. It should be noted that these D discrete numbers do not mean that in practice the charge levels are to be discrete instead of analog. They are used to derive the theoretical capacity under the considered constraints. When more constraints are introduced, the model can certainly be generalized.

As a result, *bounded rank modulation* takes the discrete levels (up to D) of a sequence of cells, and induce permutations of size m which represent the information message. We use a sliding window approach, that is, to generate one permutation from each window along the cell-level sequence.

Let us look at an example with 8 cells, a maximum of $D = 6$ levels, and permutations of size 4. As shown in the previous chapter, the charge levels of these cells $(1, 2, 3, 4, 5, 6, 2, 3)$ can represent two permutations: $(1, 2, 3, 4)$ and $(3, 4, 1, 2)$ if we group the first four and the last four cells separately. But we can also induce three permutations: $(1, 2, 3, 4), (1, 2, 3, 4), (3, 4, 1, 2)$ if we group the first, the middle, and last four cells. Even though the number of permutations is increased with 2 overlaps (i.e., 2 shared cells) between permutations, there are also more constraints on possible permutations as they are no longer independent. For example, the three permutations $(1, 2, 3, 4), (1, 2, 3, 4), (1, 2, 3, 4)$ cannot be realized since they correspond to levels $(1, 2, 3, 4, 5, 6, 7, 8)$ and exceed the maximum level D. For another example, the permutations $(1, 2, \underline{3, 4}), (\underline{2, 1}, 3, 4), (3, 4, 1, 2)$ cannot be valid since the underlined ranks correspond to the same overlapped cells but

contradict each other on the ranks.

We will in this chapter study the capacity of bounded rank modulation by considering possible permutation sequences without the above conflicts. We mainly use the technique from constraint coding and an important result is that by allowing cell groups to have overlaps, the capacity can be improved.

Except for the first rank modulation work by Jiang et al. [JMSB09], the local rank modulation [EGLSB11] is also very related to our model. That work focused on Gray code constructions. In that work, the sliding window and overlaps are also used to generate permutations. But it is further assumed that the cell-level sequence is cyclic, so the end part of the sequence is also compared with the beginning part. Moreover, no limitations on the maximum number of levels is assumed.

In this chapter, we study the bounded rank modulation model, and explore the corresponding capacity. We present computational techniques and bounds for capacity, provide encoding and decoding techniques that achieve any rate smaller than the capacity, and compare the capacities of different schemes.

9.2 Definitions

In this section, we define the basic concepts of bounded rank modulation. For convenience, for any two integers a, b such that $a \leq b$, we define $[a, b] = \{a, a+1, \cdots, b\}$.

Let m and D be integers such that $m \leq D$. A *block* is a set of m cells whose levels are from the set $[1, D]$ and are all distinct. Let $(c_1, c_2, \cdots, c_m)$ denote those m cell levels. Then by definition, $c_i \in [1, D]$ for $i \in [1, m]$ and $\forall\ i \neq j$, $c_i \neq c_j$. For convenience, we call $(c_1, c_2, \cdots, c_m)$ a *block*, too, and call $\mathcal{I}(c_1, c_2, \cdots, c_m)$ the induced *permutation*. ($\mathcal{I}$ is as defined in the previous section.) If a block B induces a permutation P, then B is called a *realization* of P. Note that a permutation may have multiple realizations. For example, if $m = 6$ and $P = (1, 4, 3, 2)$, then both $(1, 6, 4, 3)$ and $(2, 5, 4, 3)$ are realizations of P.

Let $(c_1, c_2, \cdots, c_n)$ be the levels of n cells. Let $v < m$ be an integer and for convenience, let $(n - v)/(m - v)$ be an integer as well. For $i = 1, 2, \cdots, \frac{n-v}{m-v}$, let B_i denote the block $(c_{(i-1)(m-v)+1}, c_{(i-1)(m-v)+2}, \cdots, c_{(i-1)(m-v)+m})$. Note that the last v cell levels of B_i are also the first v cell levels of B_{i+1}, so we say these two blocks overlap by v. We say $(c_1, c_2, \cdots, c_n)$ is a *cell-level sequence* that consists of blocks that overlap by v, which we may also denote by

$(B_1, B_2, \cdots, B_{(n-v)/(m-v)})$. For $i = 1, 2, \cdots, \frac{n-v}{m-v}$, let the m levels in B_i be all distinct. Then the sequence induces $(n-v)/(m-v)$ permutations $(P_1, P_2, \cdots, P_{(n-v)/(m-v)})$, where $P_i = \mathcal{I}(B_i)$ for $i = 1, 2, \cdots, \frac{n-v}{m-v}$. We call $(P_1, P_2, \cdots, P_{(n-v)/(m-v)})$ the induced *permutation sequence*, and call $(B_1, B_2, \cdots, B_{(n-v)/(m-v)})$ its *realization*. Again, a permutation sequence may have multiple realizations.

Definition 9.1 *(*BOUNDED RANK MODULATION $\mathcal{C}(n, m, D, v)$*) In a bounded rank modulation (BRM) code $\mathcal{C}(n, m, D, v)$, every codeword is a permutation sequence $(P_1, P_2, \cdots, P_{(n-v)/(m-v)})$ that has at least one realization. (The meaning of the parameters n, m, D, v is as presented above.) Let $|\mathcal{C}(n, m, D, v)|$ denote the number of codewords in code $\mathcal{C}$. Then, the* capacity *of the code is*

$$cap(\mathcal{C}) = \lim_{n\to\infty} \frac{\log |\mathcal{C}(n, m, D, v)|}{n}.$$

In general, allowing overlap between permutations can increase capacity. When there is no overlap (i.e., $v = 0$), the BRM code has capacity $\frac{\log m!}{m}$. When $v > 0$, the capacity may increase because every permutation consumes just $m - v$ cells on average.

9.3 BRM Code with One Overlap and Consecutive Levels

In this section, we study a special form of BRM code that allows efficient computation of its capacity. First, we present a computational method based on constrained systems.

Since $c_i \in [1, D]$ for $i \in [1, m]$, the BRM code is a *constrained system* over the alphabet S_m (the symmetric group on the set $[1, m]$). Define a *labeled graph* $G = (V, E, L)$ to be a directed graph with a state set V, an edge set $E \subseteq V \times V$ and an edge labeling $L : E \to S_m$. For $(u, v) \in E$, $L(u, v) = l$ is denoted by $u \xrightarrow{l} v$. G *represents* $\mathcal{C}$ if the set of all finite sequences obtained from reading the labels of paths in G equals the set of the codewords of $\mathcal{C}$. If the outgoing edges of each state are labeled distinctly, then G is *deterministic*. And G is *irreducible* if $\forall\ u, v \in V$, there is a path from u to v. Define $A_{|V|\times|V|}$ as the *adjacency matrix* of G, where A_{uv} equals the number of edges from u to v. In addition, suppose a deterministic graph G represents $\mathcal{C}(n, m, D, v)$ and $A_1, A_2, \cdots, A_k$ are the adjacency matrices of the irreducible components in G, then

$$cap(\mathcal{C}(n, m, D, v)) = \frac{\max_{1\le i\le k} \log \lambda(A_i)}{m - v} \tag{9.1}$$

where $\lambda(A)$ is largest positive eigenvalue of A [MRS01].

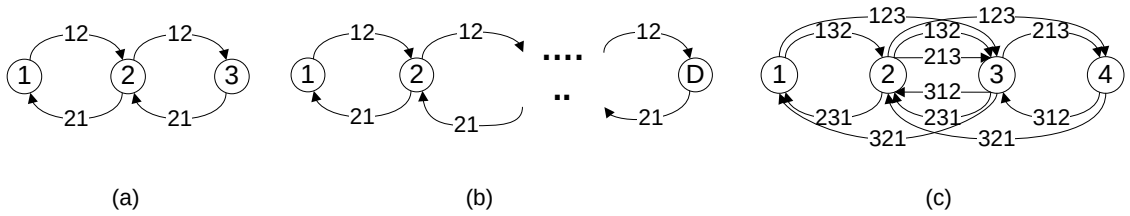

Figure 9.1: Labeled graphs for $\mathcal{C}_I$. (a) $\mathcal{C}_I(n,2,3,1)$; (b) $\mathcal{C}_I(n,2,D,1)$ and D is arbitrary; (c) $\mathcal{C}_I(n,3,4,1)$.

Example 9.2 *A BRM code $\mathcal{C}(n,2,3,1)$ can be represented by the graph G in Figure* 9.1 *(a). Each state represents the level of the current cell. $S_2 = \{12,21\}$, the states are $V = \{1,2,3\}$, and the edges are $E = \{(i,i+1)|i=1,2\} \cup \{(i,i-1)|i=2,3\}$. The labeling is defined by $L(i,i+1) =$ 12, $\forall\ i = 1,2$ and $L(i,i-1) = 21$, $\forall\ i = 2,3$. For example, the path along the states $1,2,3$, and 2 is a realization of the permutation sequence $(12,12,21)$. G is deterministic and irreducible. Hence, the adjacency matrix of G is*

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

*By (*9.1*), the capacity is* $\log(\lambda(A)) = 0.5$.

Notice in Example 9.2, the labeling L is essentially the ranks of the initial and terminal states of an edge. Also notice that every block $B_i = (c_i, c_{i+1})$ consists of two consecutive integers, i.e., $|c_i - c_{i+1}| = 1$. If we expand the idea of Example 9.2 to arbitrary $D \geq 2$ but keep $m = 2$, and $v = 1$, we will get the constrained system in Figure 9.1 (b). The adjacency matrix is

$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \dots & \dots & 1 & 0 \end{pmatrix}_{D\times D}$$

The capacity is $\log \lambda(A) = \log(2\cos(\frac{\pi}{D+1}))$ [MRS01].

We now formally define this type of constrained BRM code.

Definition 9.3 *(*BRM CODE WITH ONE OVERLAP AND CONSECUTIVE LEVELS $\mathcal{C}_I(n,m,D,1)$*) For the BRM code $\mathcal{C}_I(n,m,D,1)$, every codeword $(P_1,P_2,\cdots,P_{(n-1)/(m-1)})$ needs to satisfy the following additional constraint: the codeword has a realization $(B_1,B_2,\cdots,B_{(n-1)/(m-1)})$ such that for $i=1,2,\cdots,\frac{n-1}{m-1}$, the m cell levels in the block B_i form a set of m consecutive numbers. That is, if $B_i=(c'_1,c'_2,\cdots,c'_m)$, then $\{c'_1,c'_2,\cdots,c'_m\}=[\min_{j=1}^m c'_j,\max_{j=1}^m c'_j]$.*

In a labeled graph for $\mathcal{C}_I(n,m,D,1)$, each state corresponds to the charge level of an overlapped cell, so there are D states, $1,2,\cdots,D$. And each edge represents a permutation in a block $(c'_1,\cdots,c'_m)$. The first (or last) digit in an edge labeling corresponds to the initial (or terminal) state of the edge. Let $(a_1,\cdots,a_m)=\mathcal{I}(c'_1,\cdots,c'_m)$, then since each block has consecutive numbers, $\forall\ k,l\in[1,m]$,

$$c'_k-c'_l=a_k-a_l \tag{9.2}$$

For example, the labeled graph for $\mathcal{C}_I(n,3,4,1)$ is shown in Figure 9.1 (c).

The construction of the adjacency matrix for code $\mathcal{C}_I(n,m,D,1)$ is presented in the following theorem.

Theorem 9.4 *The adjacency matrix $A=(A_{ij})$ for $\mathcal{C}_I(n,m,D,1)$ has*

$$A_{ij}=(m-2)!\min\{m-|i-j|,i,j,D-i+1,D-j+1,D-m+1\} \tag{9.3}$$

if $1\le|i-j|\le m-1$, and $A_{ij}=0$ otherwise.

Proof: A_{ij} indicates the number of permutations with $c'_1=i,c'_m=j$. For fixed a_1 and a_m, there are $(m-2)!$ choices for $(a_2,\cdots,a_{m-1})$. Notice $i\to j$ only if $|a_1-a_m|=|c'_1-c'_m|\in[1,m-1]$. So $|\{(a_1,a_m)\}|\le m-|i-j|$, if $|i-j|\in[1,m-1]$. And $|\{(a_1,a_m)\}|=0$ otherwise. If $i\in[1,m]$, then by (9.2), $\min_{1\le k\le m}c'_k=c'_1-(a_1-1)=i-a_1+1\ge 1$, which implies $a_1\in[1,i]$, or $|\{a_1\}|=i$. Similarly, if $i\in[D-m+1,D]$, we will get $a_1\in[m-D+i,m]$, or $|\{a_1\}|=D-i+1$. For $i\in[D-m+1,m]$, $a_1\in[m-D+i,i]$, or $|\{a_1\}|=D-m+1$. And if $i\in[m,D-m+1]$, then $a_1\in[1,m]$, or $|\{a_1\}|=m$. Hence, $|\{a_1\}|=\min\{i,D-i+1,D-$

$m+1, m\}$. This argument also works for the terminal state j. Therefore, if $1 \leq |i-j| \leq m-1$,

$$\begin{aligned} A_{ij} &= (m-2)!|\{(a_1, a_m)\}| \\ &= (m-2)! \min\{m-|i-j|, |\{a_1\}|, |\{a_m\}|\} \\ &= (m-2)! \min\{m-|i-j|, i, j, D-i+1, \\ &\quad D-j+1, D-m+1\} \end{aligned}$$

And $A_{ij} = 0$ otherwise. □

The capacity of $\mathcal{C}_I$ is $cap(\mathcal{C}_I) = \frac{\log \lambda(A)}{m-1}$. Some values of $cap(\mathcal{C}_I)$ and the capacity of the non-overlap code $\mathcal{C}(m, m, D, 0)$ (for comparison) are shown in Figure 9.2.

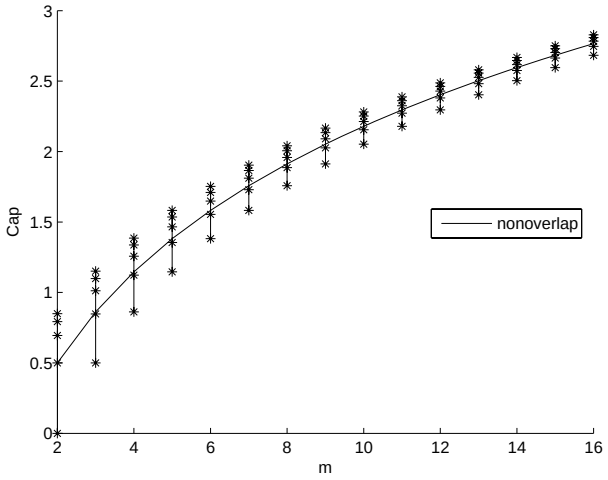

Figure 9.2: Capacity for $\mathcal{C}_I$ (stars) and for the non-overlap code (solid line). The stars in each vertical line correspond to the same permutation size m, and $D = m, m+1, \cdots, m+4$ from bottom to top.

It is clear that the capacity of the code $\mathcal{C}_I(n, m, D, 1)$ increases with D. And if $D \to \infty$, $cap(\mathcal{C}_I(n, m, D, 1)) \to \frac{\log m!}{m-1}$, which is larger than the capacity of a non-overlapping code $\mathcal{C}(n, m, D, 0)$. We now present a more general result.

Theorem 9.5 *For any $m \geq 2$ and $D \geq m+2$,*

$$cap(\mathcal{C}_I(n, m, D, 1)) > cap(\mathcal{C}(n, m, D, 0))$$

Proof: Notice $cap(\mathcal{C}(n, m, D, 0)) = \log m!/m$, $\forall\ D \geq m$. If we proved $cap(\mathcal{C}_I(n, m, m+2, 1)) > \log m!/m$, then this theorem is proved. When $m = 2, 3$, $cap(\mathcal{C}_I(n, 2, 4, 1)) = 0.6942 >$

$\log 2!/2 = 0.5$ and $cap(\mathcal{C}_I(n,3,5,1)) = 1.0120 > \log 3!/3 = 0.8617$. When $m \geq 4$, $D = m+2$, by (9.3), A is

$$(m-2)! \begin{pmatrix} 0 & 1 & 1 & 1 & \dots & 1 & 0 & 0 \\ 1 & 0 & 2 & 2 & \dots & 2 & 1 & 0 \\ 1 & 2 & 0 & 3 & \dots & 3 & 2 & 1 \\ 1 & 2 & 3 & 0 & \dots & 3 & 2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 2 & 3 & 3 & \dots & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 & \dots & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 & \dots & 1 & 1 & 0 \end{pmatrix}_{(m+2)\times(m+2)}$$

By (9.1), it is necessary to find $\lambda(A)$. Let $B = \frac{1}{(m-2)!}A$, I be the identity matrix and x be an indeterminate variable. $det(B - xI) = 0$ implies $(-x-3)^{m-3}(x^2+x-1)f(x) = 0$, where $f(x) = -x^3 + (3m-8)x^2 + (7m-10)x + 3m - 3$. Thus $\lambda(B)$ is the largest positive root of $f(x)$. Notice $\forall\ x > \lambda(B)$, $f(x) < 0$, but $f(3m-6) = 3(m^2+m-5) > 0$, $m \geq 4$. So $\lambda(B) > 3m-6$, and $\lambda(A) > (3m-6)(m-2)!$. Now we are left to show

$$\frac{\log \lambda(A)}{m-1} > \frac{\log(3(m-2)(m-2)!)}{m-1} \geq \frac{\log m!}{m}$$

which is equivalent to $\frac{3^m(m-2)^m(m-2)!}{m^{m-1}(m-1)^{m-1}} \geq 1$. Notice $m \geq 4$, $\left(1-\frac{1}{m}\right)^m \geq \frac{1}{e}$, and Stirling's Approximation, $m! \geq \sqrt{2\pi m}(m/e)^m$, thus

$$\begin{aligned} & \frac{3^m(m-2)^m(m-2)!}{m^{m-1}(m-1)^{m-1}} \\ = & \frac{3^m(m-1)(m-2)!}{m^{m-1}} \left(\frac{m-2}{m-1}\right)^{m-1} \frac{m-2}{m-1} \\ \geq & \frac{3^m(m-1)!}{m^{m-1}} \cdot \frac{1}{e} \cdot \frac{1}{2} \\ \geq & \frac{1}{2e} \frac{3^m\sqrt{2\pi(m-1)}}{e^{m-1}} \frac{(m-1)^{m-1}}{m^{m-1}} \\ \geq & \frac{1}{2e} \left(\frac{3}{e}\right)^m \sqrt{2\pi(m-1)} \geq 1 \end{aligned}$$

Thus the proof is completed. □

9.4 BRM Code with One Overlap

We now consider the general BRM code with one overlap, $\mathcal{C}(n, m, D, 1)$, which does not have the additional constraint of code $\mathcal{C}_I(n, m, D, 1)$.

In this case, the cell levels of a block, $\{c'_1, \cdots, c'_m\}$, can be any set Q such that $Q \subseteq [1, D]$ and $|Q| = m$. The labeled graph H generated is not deterministic in general. However, we are able to find a deterministic graph G that is equivalent to H (Lemma 2.1 in [MRS01]). Here is an example.

Example 9.6 *The labeled graph H of $\mathcal{C}(n, 2, 4, 1)$ is shown in Figure 9.3 (a). This is not deterministic since state 1 has 3 outgoing edges labeled 12. Let G be the deterministic representation of $\mathcal{C}$, then the states $V(G)$ are subsets of $V(H)$. And for $u, v \in V(G)$, $u \xrightarrow{l} v$ if $\forall\ j \in v$, $\exists\ i \in u$ and $i \xrightarrow{l} j$. So the resulting graph G is as shown in Figure 9.3 (b). States $\{2\}$, $\{3\}$, $\{1,3\}$, etc., have only outgoing edges, so their capacities are 0. Therefore the irreducible component of G maximizing $\lambda(A_i)$ is as in Figure 9.3 (c). By (9.1)we can then get $cap(\mathcal{C}(n, 2, 4, 1)) = \log \lambda(A_i) = 0.8791 > cap(\mathcal{C}_I(n, 2, 4, 1)) = 0.6942$.*

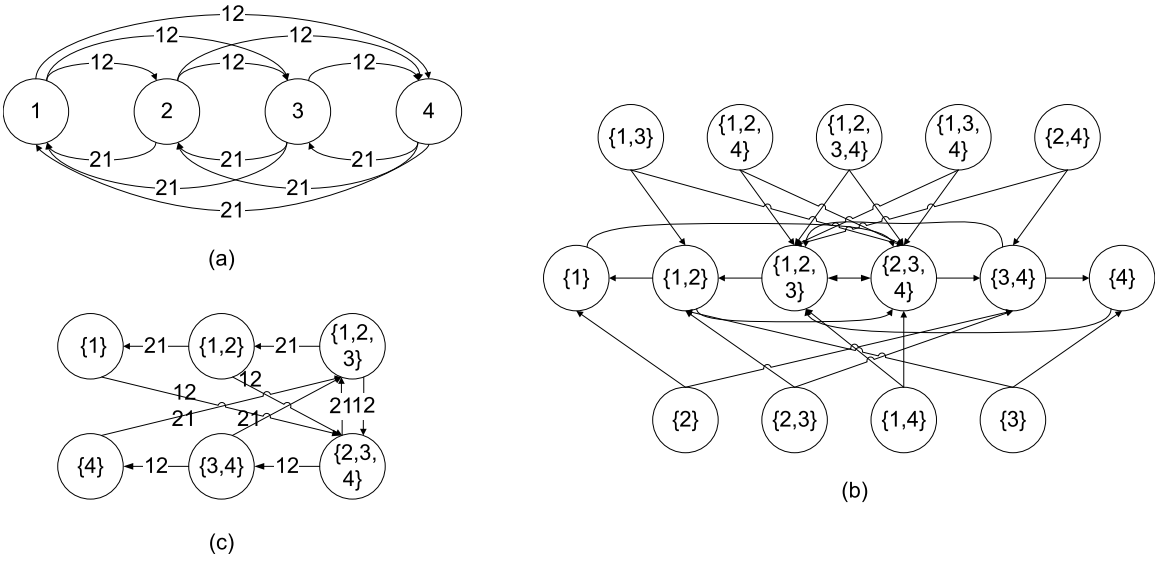

Figure 9.3: Labeled graphs for $\mathcal{C}(n, 2, 4, 1)$. (a) Labeled graph; (b) deterministic graph; (c) irreducible graph.

In general, suppose the deterministic graph G represents $\mathcal{C}(n, 2, D, 1)$, and A_i is the adjacency matrix for the irreducible component of G that has the largest eigenvalue. Then $\lambda(A_i)$ is the largest positive root of $-x^D + 2x^{D-1} - 1 = 0$. Comparing $cap(\mathcal{C}_I)$ and $cap(\mathcal{C})$, we have Figure 9.4. It can be seen that $cap(\mathcal{C})$ tends to 1 faster than $cap(\mathcal{C}_I)$, since it makes better use of the levels provided.

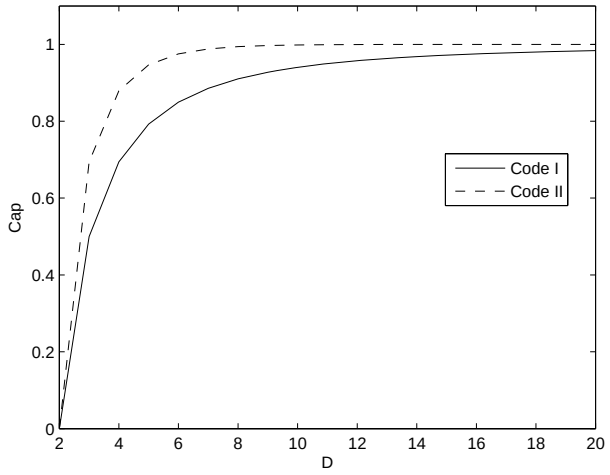

Figure 9.4: Capacity for $\mathcal{C}$. The solid and dashed lines show capacity for $\mathcal{C}_I$ and $\mathcal{C}$, respectively.

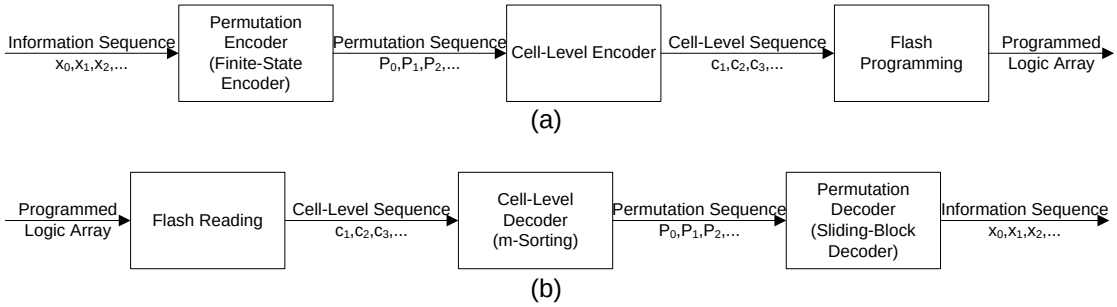

Figure 9.5: (a) Encoder and (b) decoder for BRM codes

The construction in the above example can be naturally extended to the case $m > 2$.

Encoders and decoders for BRM codes can be constructed as in Figure 9.5. In the encoding process, the input information sequence $(x_0, x_1, \dots)$ is first encoded into a permutation sequence, $(P_0, P_1, \dots)$, satisfying the maximum cell-level constraint, which is further mapped to a cell-level sequence, $(c_1, c_2, \dots)$. At last, the cell-level sequence is programmed into flash memory. And the decoder reverses this process by reading the cell levels, forming a permutation sequence, and at last retrieving the information.

- Permutation encoder/decoder

Let $\mathcal{C}$ be a constrained system with a representation G, and p,q be positive integers. The q-th power of $\mathcal{C}$, $\mathcal{C}^q$, is represented by the labeled graph G^q, with the same set of states as G and edges/labelings corresponding to paths/labelings of length q in G. A *finite-state encoder* with rate $p : q$ is a lossless labeled graph H such that $H \subseteq G^q$ and each state of H has out-degree 2^p. We

can then assign 2^p input tags (or p binary information bits) to the outgoing edges of each state. An encoder is (m,a)-*sliding-block decodable* if the i-th input tag in an input tag sequence is uniquely determined by the q-block labeling sequence $P_{i-m}^q, P_{i-m+1}^q, \ldots, P_i^q, \ldots, P_{i+a}^q$. The following theorem states that the capacity of any constrained system is always achievable [MRS01].

Theorem 9.7 *Let $\mathcal{C}(n, m, D, v)$ be a constrained system and $p/q < (m-v)cap(\mathcal{C}(n, m, D, v))$ for positive integers p and q. Then there exists a sliding-block decodable finite-state encoder with rate $p : q$ and permutation encoding rate $p/(q(m-v))$.*

Theorem 9.7 can be proved by explicit constructions of encoders, such as the state-splitting algorithm [MRS01]. And a sliding-block decoder is essentially a mapping from $(m + a + 1)$ q-block labelings to a p-block binary input tag. Notice in the decoding process, decoding delay and error propagation is controlled within $m + a + 1$ q-blocks, which depends on the construction of the encoder/decoder.

Example 9.8 *Continuing Example* 9.6*, take $p = 3$ and $q = 4$, then $p/q < cap(\mathcal{C}(n, 2, 4, 1)) = 0.8792$. The 4-th power of $\mathcal{C}(n, 2, 4, 1)$ has adjacency matrix*

$$A_i^4 = \begin{pmatrix} 4 & 2 & 1 & 1 & 2 & 3 \\ 3 & 2 & 1 & 1 & 1 & 3 \\ 2 & 1 & 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 & 1 & 2 \\ 3 & 1 & 1 & 1 & 2 & 3 \\ 3 & 2 & 1 & 1 & 2 & 4 \end{pmatrix}$$

After deleting States 3 *and* 4*, and some edges from the graph, we get a finite-state encoder in Figure* 9.4 *(a), which has out-degree $2^p = 8$ for each state, and each labeling block has size $q = 4$. The notation $u \overset{x^3/P^4}{\rightarrow} v$ means the* 4*-block labeling P^4 is assigned the binary input tag x^3. For convenience, we denote the permutation $(1, 2)$ by* 1*, and $(2, 1)$ by* 0 *in the labeling. After merging the States* 1 *and* 6*, it is further simplified as in Figure* 9.4 *(b).*

Let State 1 *be the initial state for the encoding. Divide the input binary information bits into blocks of* 3*, and for any $x_i^3 = (x_{3i}, x_{3i+1}, x_{3i+2})$, encode it as the corresponding* 4*-block labeling, $P_i^4 = (P_{4i}, P_{4i+1}, P_{4i+2}, P_{4i+3})$.*

Notice that in the encoding process, each 4*-block labeling corresponds to only one input tag, independent of starting state. Therefore, we can construct a $(0, 0)$-sliding-block decoder: for each*

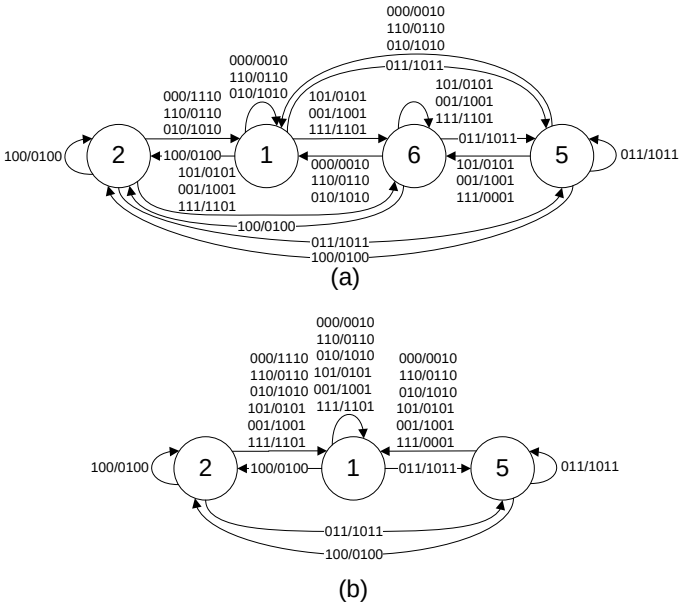

Figure 9.6: Rate $3:4$ finite-state encoder for $\mathcal{C}(n,2,4,1)$. (a) Labeled subgraph with out-degree 8, and (b) simplified encoder.

received permutation block P_i^4 decode it to the unique information block $x_i^3 = (x_{3i}, x_{3i+1}, x_{3i+2})$, which equals to $(P_{4i+1}, P_{4i+2}, P_{4i+3})$ if $P_{4i} \neq P_{4i+1}$ and equals to $(P_{4i+3}, P_{4i+3}, P_{4i+3})$, otherwise.

- Cell-level encoder/decoder

Before programming into flash memories, we must first encode the permutation sequence $(P_0, P_2, \dots)$ into a cell-level sequence $(c_1, c_2, \dots)$, such that it induces this permutation sequence and does not exceed the maximum level. The following construction provides such an encoding. Let $P_i = (a_1^i, a_2^i, \dots, a_m^i)$ and denote $c_{i(m-1)+j}$ by c_j^i, for $i \geq 0$, $j = 1, 2, \dots, m$. Then for BRM codes with one overlap, c_j^i is the cell level of a_j^i, for $i \geq 0$, $j = 1, 2, \dots, m$, and $c_1^i = c_m^{i-1}$, for $i \geq 1$.

Construction 9.9 *Let $(P_0, \dots, P_i, \dots)$ be the input permutation sequence of the cell-level encoder. Then the following assignment of $c_1^0, c_2^0, \dots, c_{m-1}^0, \dots, c_1^i, c_2^i, \dots, c_{m-1}^i, \dots$ defines a cell-level encoder.*

$$c_1^0 = \begin{cases} a_1^0, & \text{if } a_1^0 < a_m^0 \\ D - m + a_1^0, & \text{if } a_1^0 > a_m^0 \end{cases}$$

For $i \geq 1$,

$$c_1^i = \begin{cases} \max(a_1^i, c_1^{i-1} + a_m^{i-1} - a_1^{i-1}), & \text{if } a_1^{i-1} < a_m^{i-1}, a_1^i < a_m^i \\ \max(a_1^i, a_m^{i-1}), & \text{if } a_1^{i-1} > a_m^{i-1}, a_1^i < a_m^i \\ \min(D - m + a_1^i, D - m + a_m^{i-1}), & \text{if } a_1^{i-1} < a_m^{i-1}, a_1^i > a_m^i \\ \min(D - m + a_1^i, c_1^{i-1} + a_m^{i-1} - a_1^{i-1}), & \text{if } a_1^{i-1} > a_m^{i-1}, a_1^i > a_m^i \end{cases} \tag{9.4}$$

For $i \geq 0$ and $j = 2, \ldots, m-1$,

$$c_j^i = \begin{cases} c_1^i + a_j^i - a_1^i, & \text{if } a_1^i < a_m^i, a_j^i < a_m^i \text{ or } a_1^i > a_m^i, a_j^i > a_1^i \\ c_1^{i+1} + a_j^i - a_m^i, & \text{if } a_1^i < a_m^i, a_j^i > a_m^i \text{ or } a_1^i > a_m^i, a_j^i < a_1^i \end{cases} \tag{9.5}$$

When $m = 2$, using 1 and 0 to represent the permutations $(1,2)$ and $(2,1)$, respectively, the above construction is reduced to

$$c_1 = \begin{cases} 1, & \text{if } P_0 = 1 \\ 0, & \text{if } P_0 = 0 \end{cases}$$

and for $i \geq 1$,

$$c_{i+1} = \begin{cases} c_i + 1, & \text{if } P_{i-1} = 1, P_i = 1 \\ 1, & \text{if } P_{i-1} = 0, P_i = 1 \\ D, & \text{if } P_{i-1} = 1, P_i = 0 \\ c_i - 1, & \text{if } P_{i-1} = 0, P_i = 0 \end{cases}$$

One can check that if a permutation sequence satisfies the constraints for $\mathcal{C}(n, 2, D, 1)$, i.e., there are at most $D-1$ zeros (or ones) between any two successive ones (or zeros), then the generated cell-level sequence realizes this sequence and has cell levels in $[1, D]$.

We now show that Construction 9.9 generates a cell-level realization for each codeword in BRM code.

Theorem 9.10 *Let $(P_0, P_1, \ldots, P_n)$ be a codeword in $\mathcal{C}((m-1)n+1, m, D, 1)$ and $C = (c_1^0, c_2^0, \ldots, c_{m-1}^0, \ldots, c_1^{n-1}, c_2^{n-1}, \ldots, c_{m-1}^{n-1}, c_1^n)$ be the cell-level sequence generated in Construction 9.9. Then C is a realization of $P = (P_0, P_1, \ldots, P_{n-1})$. In particular, each cell level ranges between 1 and D.*

Proof: From (9.5) it is clear that $(c_1^i, c_2^i, \ldots, c_{m-1}^i, c_1^{i+1})$ has ranks $P_i = (a_1^i, a_2^i, \ldots, a_m^i)$. We are left to show that C ranges between 1 and D.

Assume $(d_1^0, d_2^0, \ldots, d_{m-1}^0, \ldots, d_1^n, d_2^n, \ldots, d_{m-1}^n, d_1^{n+1})$ is an arbitrary realization of P and $1 \leq d_j^i \leq D$ for $j = 1, \ldots, m-1$, $i = 0, 1, \ldots, n$. For c_1^i, $i = 0, 1, \ldots, n-1$, we will prove a stronger condition:

$$\begin{aligned} &a_1^i \leq c_1^i \leq d_1^i, a_m^{i-1} \leq c_1^i, \text{ if } a_1^i < a_m^i \\ &d_1^i \leq c_1^i \leq a_1^i + D - m, c_1^1 \leq a_m^{i-1} + D - m, \text{ if } a_1^i > a_m^i \end{aligned} \tag{9.6}$$

The inequalities containing a_m^{i-1} are not considered when $i = 0$. Notice the cell-level sequence $(d_1^i, \ldots, d_{m-1}^i, d_1^{i+1})$ induces $(a_1^i, \ldots, a_{m-1}^i, a_m^i)$. Hence, for $1 \leq j, k \leq m$ such that $a_j^i \geq a_k^i$, we have

$$a_j^i - a_k^i \leq d_j^i - d_k^i \tag{9.7}$$

Let $a_j^i = a_1^i, a_k^i = 1$, and $a_j^i = m, a_k^i = a_1^i$, and we get

$$a_1^i \leq d_1^i \leq a_1^i + D - m \tag{9.8}$$

Similarly, since $(d_1^{i-1}, \ldots, d_{m-1}^{i-1}, d_1^i)$ induces $(a_1^{i-1}, \ldots, a_{m-1}^{i-1}, a_m^{i-1})$, we have $a_m^{i-1} \leq d_1^i \leq a_m^{i-1} + D - m$. Suppose (9.6) holds, then $1 \leq a_1^i \leq c_1^i \leq a_1^i + D - m \leq D$ and $1 \leq a_m^{i-1} \leq c_1^i \leq a_m^{i-1} + D - m \leq D$. And by (9.5), either $1 \leq a_j^i = a_1^i + a_j^i - a_1^i \leq c_j^i = c_1^i + a_j^i - a_1^i \leq (a_1^i + D - m) + a_j^i - a_1^i = a_j^i + D - m \leq D$ or $1 \leq a_j^i \leq c_j^i = c_1^{i+1} + a_j^i - a_m^i \leq a_j^i + D - m \leq D$, which would complete the proof.

we will prove (9.6) by induction. For the base case($i = 0$), if $a_1^0 < a_m^0$, then $c_1^0 = a_1^0 \leq d_1^0$. And if $a_1^0 > a_m^0$, then $d_1^0 \leq a_1^0 + D - m = c_1^0$. Now suppose (9.6) holds for $i - 1 \geq 0$. If $a_1^{i-1} < a_m^{i-1}$ and $a_1^i < a_m^i$, then by induction,

$$a_1^{i-1} \leq c_1^{i-1} \leq d_1^{i-1}, \quad a_m^{i-2} \leq c_1^{i-1} \tag{9.9}$$

Now either $a_m^{i-1} \overset{(9.9)}{\leq} c_1^{i-1} + a_m^{i-1} - a_1^{i-1} \leq a_1^i = c_1^i \overset{(9.8)}{\leq} d_1^i$ or $a_1^i \leq c_1^{i-1} + a_m^{i-1} - a_1^{i-1} = c_1^i \overset{(9.9)}{\leq} d_1^{i-1} + a_m^{i-1} - a_1^{i-1} \overset{(9.7)}{\leq} d_1^i$ and $a_m^{i-1} \overset{(9.9)}{\leq} c_1^{i-1} - a_1^{i-1} + a_m^{i-1} = c_1^i$. Therefore, (9.6) holds for c_1^i in this case. If $a_1^{i-1} > a_m^{i-1}$ and $a_1^i < a_m^i$, then by (9.8) either $a_m^{i-1} \leq a_1^i = c_1^i \leq d_1^i$ or $a_1^i \leq a_m^{i-1} = c_1^i \leq d_1^i$. Therefore, (9.6) holds in this case, too. And we can prove by similar arguments that (9.6) is true for the other two cases, thus complete this proof. □

The cell-level decoder is an m-sorter that orders every m-cell-level tuple associated with the permutations. The complexity is $m \log m$.

- Flash programming/reading

To avoid over-programming, the programming of a cell-level sequence into flash memories is operated from the lowest to the highest rank. Such a programming method will lead to a delay of n in the worst case. However, since for BRM code with one overlap, only c_1^i, $i \geq 0$ are contained in two permutations, we only need to obtain correct ordering for $\{c_1^i, c_1^{i+1}\}$ and $\{c_1^i, c_2^i, \ldots, c_{m-1}^i, c_1^{i+1}\}$, for all $i \geq 0$. And ordering of other sets of cell levels are not used in the code. Construction 9.11 follows the above idea and has smaller delay than n, for n sufficiently long.

Construction 9.11 *we first define a writing operation for c_1^i, denoted by $Op(i)$, as follows.*

1. *If $i \geq 1$, compare c_1^i and c_1^{i-1}. If $c_1^i < c_1^{i-1}$, write into flash $\{c_j^{i-1} | j = 2, \ldots, m-1, c_j^{i-1} < c_1^i\}$ from low to high level. Otherwise, write $\{c_j^{i-1} | j = 2, \ldots, m-1, c_1^{i-1} < c_j^{i-1} < c_1^i\}$ from low to high level.*

2. *If $c_1^i < c_1^{i+1}$, write into flash $\{c_j^i | j = 2, \ldots, m-1, c_j^i < c_1^i\}$ from low to high level. If $c_1^i > c_1^{i+1}$, write $\{c_j^i | j = 2, \ldots, m-1, c_1^{i+1} < c_j^i < c_1^i\}$ from low to high level.*

3. *Write c_1^i so that it has a higher charge level than all the cells written in steps 1 and 2.*

4. *If $i \geq 1$, and $c_1^i > c_1^{i-1}$, then write $\{c_j^{i-1} | j = 2, \ldots, m-1, c_j^{i-1} > c_1^i\}$ from low to high level. And if $c_1^i > c_1^{i+1}$, write into flash $\{c_j^i | j = 2, \ldots, m-1, c_j^i > c_1^i\}$ from low to high level.*

In steps 1, 2, and 4, the only requirement is that the charge level of each programmed cell is higher than the previously written one.

Now starting from $i = 0$, if $c_1^i < c_1^{i+1}$, do $Op(i)$, and update i with $i+1$. Otherwise, find the smallest $e \geq 1$, such that $c_1^{i+e} < c_1^{i+e+1}$, do $Op(i+e), Op(i+e-1), \ldots, Op(i)$, and then update i with $i+e+1$.

The above procedure clearly realizes the given permutation sequence, and moreover has a worst-case delay $D(m-1)$. In the worst case, $c_1^i = D, c_1^{i+1} = D-1, \ldots, c_1^{i+D-1} = 1$, and $c_1^{i-1} < c_2^{i-1} < D$, for some $i \geq 1$, hence one has to receive $c_2^{i-1}, \ldots, c_{m-1}^{i-1}, c_1^i, \ldots, c_{m-1}^i, \ldots, c_1^{i+D-1}$ to write c_2^i.

For BRM code with permutations of size 2, the programming process reduces to operations on c_1^i only, for $i \geq 0$, with maximum delay D.

Flash reading can be simply realized by sequentially reading off cell charge levels from flash.

Notice that both the cell-level encoder and flash programming has rate 1, so the BRM code encoder has rate $p/(q(m-v))$ by Theorem 9.7, which can be arbitrarily close to the capacity.

We will now give an example of the complete encoding/decoding process for BRM code $\mathcal{C}(n,2,4,1)$. The encoder has rate 0.75 in this example.

Example 9.12 *Encoding: suppose the information sequence is* $(x_0 \dots x_{11} \dots) =$ $(001 \quad 100 \quad 000 \quad 010 \dots)$. *Then following Example* 9.8*, the encoded permutation sequence is* $(P_0 \dots P_{15}) = (1001 \quad 0100 \quad 1110 \quad 1010 \dots)$*, and the state transition in the finite-state encoder is state* $1 \to 1 \to 2 \to 1 \to 1$. *Using Construction* 9.9*, we get the encoded cell-level sequence* $(c_1 \dots c_{16} \dots) = (1\ 431\ 41\ 431\ 2\ 3\ 41\ 41\ 4 \dots)$. *And at last it can be programmed into flash in the order:* $(c_1, c_4, c_3, c_2, c_6, c_5, c_9, c_8, c_7, c_{10}, c_{11}, c_{13}, c_{12}, c_{15}, c_{14} \dots)$.

Decoding: assume we are to decode the flash cells programmed above. First read the cells sequentially and then compare c_{i+1} *and* c_{i+2}*, for* $i \geq 0$. *Decode* P_i *as* 1 *if* $c_{i+1} < c_{i+2}$*, as* 0 *if* $c_{i+1} > c_{i+2}$. *Thus we get* $(P_0 \dots P_{15}) = (1001 \quad 0100 \quad 1110 \quad 1010 \dots)$*, and by the decoding scheme in Example* 9.8*, we get* $(x_0 \dots x_{11} \dots) = (001 \quad 100 \quad 000 \quad 010 \dots)$.

9.5 Lower Bound for Capacity

In this section, we present a lower bound to the capacity of the BRM code. To derive this bound, we first present a new form of rank modulation called the *Star BRM*.

9.5.1 Star BRM

A Star BRM code uses $n+v$ cells. For convenience, let n be a multiple of $m-v$. v of these $n+v$ cells are called *anchors*, and we denote their cell levels by $(\ell_1, \ell_2, \cdots, \ell_v)$. The other n cells are called *storage cells*, and we denote their cell levels by $c_1, c_2, \cdots, c_n$. For $i = 1, 2, \cdots, v$, $\ell_i \in [1, D]$; for $i = 1, 2, \cdots, n$, $c_i \in [1, D]$. For $i = 1, 2, \cdots, \frac{n}{m-v}$, we define *block* B_i to be these m cell levels: $(\ell_1, \ell_2, \cdots, \ell_v, c_{(i-1)(m-v)+1}, c_{(i-1)(m-v)+2}, \cdots, c_{i(m-v)})$. We can see that these $\frac{n}{m-v}$ blocks share the same v cells, namely, the anchor cells. For $i = 1, 2, \cdots, \frac{n}{m-v}$, we require that the m cell levels in the block B_i are all different, and we use P_i to denote the permutation induced by B_i. B_i is a *realization* of P_i. Again, a permutation sequence $(P_1, P_2, \cdots, P_{n/(m-v)})$ may have multiple realizations.

Definition 9.13 *(*STAR BRM CODE $\mathcal{S}(n,m,D,v)$*) In a Star BRM code $\mathcal{S}(n,m,D,v)$, every codeword is a permutation sequence $(P_1, P_2, \cdots, P_{n/(m-v)})$ that has at least one realization. (The meaning of the parameters n,m,D,v is as presented above.) Let $|\mathcal{S}(n,m,D,v)|$ denote the number of codewords in code $\mathcal{S}$. Then, the* capacity *of the code is*

$$cap(\mathcal{S}) = \lim_{n\to\infty} \frac{\log|\mathcal{S}(n,m,D,v)|}{n+v}.$$

To derive the capacity of Star BRM, we first show how the anchor levels $(\ell_1, \ell_2, \cdots, \ell_v)$ affect the permutation sequences. Define $Z(\ell_1, \ell_2, \cdots, \ell_v)$ as the total number of permutations that can be induced by the cell levels $(\ell_1, \ell_2, \cdots, \ell_v, c'_1, c'_2, \cdots, c'_{m-v})$, where the m cell levels are all different and all belong to the set $[1, D]$. (Here $(\ell_1, \ell_2, \cdots, \ell_v)$ are fixed, and the $m-v$ cell levels $(c'_1, c'_2, \cdots, c'_{m-v})$ can vary and therefore can have $\frac{(D-v)!}{(D-m)!}$ combinations. Some of them induce the same permutation.) It can be observed that when we permute the v anchor levels $(\ell_1, \ell_2, \cdots, \ell_v)$, the value of $Z(\ell_1, \ell_2, \cdots, \ell_v)$ remains the same. For example, when $v = 3$ and $D = 6$, $Z(2,3,6) = Z(3,2,6) = Z(6,2,3)$. So without loss of generality (WLOG), we assume $\ell_1 < \ell_2 < \cdots < \ell_v$.

Given $(\ell_1, \ell_2, \cdots, \ell_v)$, let $\beta(\ell_1, \ell_2, \cdots, \ell_v)$ denote the number of solutions for the variables $x_1, x_2, \cdots, x_{v+1}$ that satisfy the following two conditions: (1) $\sum_{i=1}^{v+1} x_i = m - v$; (2) $x_1 \in [0, \ell_1 - 1]$, $x_i \in [0, \ell_i - \ell_{i-1} - 1]$ for $i \in [2, v]$, and $x_{v+1} \in [0, D - \ell_v]$.

Lemma 9.14 *Given $D \geq m > v$, we have $Z(\ell_1, \ell_2, \cdots, \ell_v) = (m-v)! \cdot \beta(\ell_1, \ell_2, \cdots, \ell_v)$.*

Proof: Given the anchor cell levels $(\ell_1, \ell_2, \cdots, \ell_v)$, a permutation induced by $(\ell_1, \ell_2, \cdots, \ell_v, c'_1, c'_2, \cdots, c'_{m-v})$ can be uniquely determined by the following two steps: (1) determine the relative order of the $m-v$ cell levels $(c'_1, c'_2, \cdots, c'_{m-v})$ (that is, which cell level is the highest, second highest, and so on $\cdots$ among them); (2) determine how many cell levels among $(c'_1, c'_2, \cdots, c'_{m-v})$ are below ℓ_1, or between ℓ_1 and ℓ_2, or between ℓ_2 and ℓ_3, $\cdots$, or above ℓ_v. Step 1 has $(m-v)!$ choices, and step 2 has $\beta(\ell_1, \ell_2, \cdots, \ell_v)$ choices. So the conclusion holds. □

Lemma 9.15 *$Z(\ell_1, \ell_2, \cdots, \ell_v)$ is maximized when the numbers in the following set differ by at most one: $\{\ell_1 - 1, D - \ell_v\} \cup \{\ell_i - \ell_{i-1} - 1 | i = 2, 3, \cdots, v\}$. (That is, every number in the above set is either $\lfloor \frac{D-v}{v+1} \rfloor$ or $\lceil \frac{D-v}{v+1} \rceil$.)*

Proof: By Lemma 9.14, maximizing $Z(\ell_1, \ell_2, \cdots, \ell_v)$ is equivalent to maximizing $\beta(\ell_1, \ell_2, \cdots, \ell_v)$. Define $\alpha_1 = \ell_1 - 1$, $\alpha_i = \ell_i - \ell_{i-1} - 1$ for $i \in [2, v]$, and $\alpha_{v+1} = D - \ell_v$.

Suppose there exists $i \neq j$ such that $\alpha_i \geq \alpha_j + 2$. WLOG, let $i < j$. Let $x_1, x_2, \cdots, x_{v+1}$ be variables satisfying these two conditions: (1) $\sum_{k=1}^{v+1} x_k = m - v$; (2) $x_k \in [0, \alpha_k]$ for $k \in [1, v+1]$. The number of such solutions is $\beta(\ell_1, \ell_2, \cdots, \ell_v)$. Now, let us fix the values of $x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{j-1}, x_{j+1}, \cdots, x_{v+1}$ (in a valid solution), and see how many different values x_i can take. (Note that the value of x_j is determined by x_i.)

Let $z = D - \sum_{k \in \{1,\cdots,i-1,i+1,\cdots,j-1,j+1,\cdots,v+1\}} x_k = x_i + x_j$. Let $\gamma(z)$ denote the number of values x_i can take. The constraints are $0 \leq x_i \leq \alpha_i, 0 \leq z - x_i \leq \alpha_j$. If $z \geq \alpha_i$, $\gamma(z) = \alpha_i + \alpha_j - z + 1$; if $\alpha_j \leq z < \alpha_i$, $\gamma(z) = \alpha_j + 1$; if $z < \alpha_j$, $\gamma(z) = z + 1$. So if we increase α_j by one and decrease α_i by one, $\gamma(z)$ will not decrease although the values $\alpha_1, \alpha_2, \cdots, \alpha_{v+1}$ will become more even. So given a sequence $(\ell_1, \ell_2, \cdots, \ell_v)$, we can change it that way into a sequence that satisfies the condition in the lemma, without decreasing $\beta(\ell_1, \ell_2, \cdots, \ell_v)$. It is easy to see that when $\alpha_1, \alpha_2, \cdots, \alpha_{v+1}$ differ by at most one, no matter what their order is, $\beta(\ell_1, \ell_2, \cdots, \ell_v)$ is the same (which is the maximum value of $\beta(\ell_1, \ell_2, \cdots, \ell_v)$). □

Let $(\ell_1^*, \ell_2^*, \cdots, \ell_v^*)$ be the v anchor levels that satisfy the condition in Lemma 9.15. It maximizes the value of $Z(\ell_1, \ell_2, \cdots, \ell_v)$. For convenience, we assume that $\ell_1^* < \ell_2^* < \cdots < \ell_v^*$. It is very simple to find these v values. For convenience, we use Z^* to denote $Z(\ell_1^*, \ell_2^*, \cdots, \ell_v^*)$, and use β^* to denote $\beta(\ell_1^*, \ell_2^*, \cdots, \ell_v^*)$.

The values of β^* and Z^* can be computed efficiently by the following dynamic programming algorithm of time complexity $O(D^2)$. Let $\alpha_1 = \ell_1^* - 1$, $\alpha_i = \ell_i^* - \ell_{i-1}^* - 1$ for $i \in [2, v]$, and $\alpha_{v+1} = D - \ell_v^*$. Let $w(i, j)$ denote the number of solutions for $x_1, x_2, \cdots, x_i$ such that $\sum_{k=1}^{i} x_k = j$ and $x_k \in [0, \alpha_k]$ for $k = 1, \cdots, i$. The algorithm is as follows: (1) $w(i, j) = \sum_{k=0}^{\alpha_i} w(i-1, j-k)$. Also, $w(i, j) = 0$ if $j < 0$, $w(1, j) = 1$ if $0 \leq j \leq \alpha_1$, and $w(1, j) = 0$ if $j > \alpha_1$; (2) $\beta^* = w(v+1, D-v)$, and $Z^* = (m-v)!\beta^*$.

The following theorem presents the capacity of the Star BRM.

Theorem 9.16 *The capacity of the Star BRM code $\mathcal{S}(n, m, D, v)$ is*

$$cap(\mathcal{S}) = \frac{\log Z^*}{m - v}.$$

Proof: We first show that $cap(\mathcal{S}) \leq \frac{\log Z^*}{m-v}$. There are $v!\binom{D}{v}$ ways to assign values to $(\ell_1, \ell_2, \cdots, \ell_v)$, which we denote by $W = \{w_1, w_2, \cdots, w_{v!\binom{D}{v}}\}$. We call $(\ell_1, \ell_2, \cdots, \ell_v, c_1, c_2, \cdots, c_n)$ the *cell-level sequence*. For $i = 1, 2, \cdots, v!\binom{D}{v}$, let $\gamma_{i,n}$ denote the maximum set of cell-level sequences satisfying two conditions: (1) They all assign w_i to

$(\ell_1, \ell_2, \cdots, \ell_v)$; (2) The permutations induced by them are all distinct.

By the definition of $Z(\ell_1, \ell_2, \cdots, \ell_v)$, every *block* can induce $Z(\ell_1, \ell_2, \cdots, \ell_v)$ permutations. Since there are $n/(m-v)$ blocks, we get $|\gamma_{i,n}| = (Z(w_i))^{\frac{n}{m-v}}$. By Lemma 9.15, $Z(w_i) \leq Z^*$. Since every codeword of $\mathcal{S}$ has at least one realization in some $\gamma_{i,n}$,

$$|\mathcal{S}(n,m,D,v)| \leq \sum_{i=1,2,\cdots,v!\binom{D}{v}} |\gamma_{i,n}| \leq v!\binom{D}{v}(Z^*)^{\frac{n}{m-v}}.$$

So

$$cap(\mathcal{S}) = \lim_{n\to\infty} \frac{\log|\mathcal{S}(n,m,D,v)|}{n+v} \leq \lim_{n\to\infty} \frac{\log(v!\binom{D}{v}(Z^*)^{\frac{n}{m-v}})}{n} = \frac{\log Z^*}{m-v}.$$

We now show that $cap(\mathcal{S}) \geq \frac{\log Z^*}{m-v}$. Say that $(\ell_1^*, \ell_2^*, \cdots, \ell_v^*) = w_{i'}$ for some i'. Every codeword of $\mathcal{S}$ has at most one configuration in $\gamma_{i',n}$, so $|\mathcal{S}(n,m,D,v)| \geq |\gamma_{i',n}|$. So

$$cap(\mathcal{S}) = \lim_{n\to\infty} \frac{\log|\mathcal{S}(n,m,D,v)|}{n+v} \geq \lim_{n\to\infty} \frac{\log|\gamma_{i',n}|}{n} = \lim_{n\to\infty} \frac{\log(Z^*)^{\frac{n}{m-v}}}{n} = \frac{\log Z^*}{m-v}.$$

So the theorem is proved. □

The above proof leads to the following corollary.

Corollary 9.17 *The Star BRM code $\mathcal{S}(n,m,D,v)$ achieves its capacity even if the v anchor cell levels are fixed as $(\ell_1^*, \ell_2^*, \cdots, \ell_v^*)$.*

The capacity of the Star BRM code $\mathcal{S}(n,m,D,v)$ is non-decreasing in D. However, when $D = (m-v+1)v + (m-v)$, the capacity reaches its maximum value. Further increasing D will not increase the capacity. That is because when $D \geq (m-v+1)v + (m-v)$, Z^* reaches its maximum value $m!/v!$.

9.5.2 Lower Bound for the Capacity of BRM

We now derive a lower bound for the capacity of the bounded rank modulation code $\mathcal{C}(n,m,D,v)$.

Theorem 9.18 *For the BRM code $\mathcal{C}(n,m,D,v)$, when $m \geq 2v$, its capacity*

$$cap(\mathcal{C}) \geq \frac{\log Z^* + \log v! + \log(m-2v)!}{2(m-v)}.$$

(As presented previously, Z^* is a value determined by the parameters m, D, v.)

Proof: Let $\mathcal{S}(n,m,D,v)$ be a Star BRM code with an additional constraint: every codeword of $\mathcal{S}$ has a realization in which the v anchor levels are $(\ell_1^*,\ell_2^*,\cdots,\ell_v^*)$. By Corollary 9.17, $\mathcal{S}$ achieves capacity. For convenience, assume $n/(m-v)$ is an integer.

Let $(P_1,P_2,\cdots,P_{n/(m-v)})$ be a codeword in $\mathcal{S}$, and let $(\ell_1^*,\ell_2^*,\cdots,\ell_v^*,c_1,c_2,\cdots,c_n) = (B_1,B_2,\cdots,B_{n/(m-v)})$ be its realization. For $i=1,2,\cdots,n/(m-v)$, corresponding to block B_i, we build two blocks B_i' and B_i'' of length m as follows. Say $B_i=(\ell_1^*,\ell_2^*,\cdots,\ell_v^*,c_1',c_2',\cdots,c_{m-v}')$. The first v cell levels of B_i' take values from the set $\{\ell_1^*,\ell_2^*,\cdots,\ell_v^*\}$ (we have $v!$ choices here), and the next $m-v$ cell levels of B_i' are the same as $(c_1',c_2',\cdots,c_{m-v}')$. The first v cell levels of B_i'' overlap the last v cell levels of B_i'. For the next $m-2v$ cell levels of B_i'', we first pick $m-2v \le D-2v$ values different from the first v and the last v cell levels of B_i', then let the $m-2v$ cell levels take only those $m-2v$ values (we have $(m-2v)!$ choices here). The final v cell levels of B_i'' take values again from the set $\{\ell_1^*,\ell_2^*,\cdots,\ell_v^*\}$. Then we construct a cell-level sequence $(B_1',B_1'',B_2',B_2'',\cdots,B_{n/(m-v)}',B_{n/(m-v)}'')$, where every two adjacent blocks overlap by v. Corresponding to every codeword $s\in\mathcal{S}$, there are at least $(v!(m-2v)!)^{\frac{n}{m-v}}$ such cell-level sequences, which we denote by Q_s. It is simple to see that no two cell-level sequences in Q_s induce the same permutation sequence. On the other side, when $s\neq s'$, every pair of cell-level sequences from Q_s and $Q_{s'}$, respectively, also induce different permutation sequences. (To see that, let us call the pair of cell-level sequences q and q'. Replace all their overlapping cell levels by $(\ell_1^*,\ell_2^*,\cdots,\ell_v^*)$, and get two new cell-level sequences p and p'. The codewords s and s' are subsequences of $\mathcal{I}(p)$ and $\mathcal{I}(p')$, respectively. Since $s\neq s'$, $\mathcal{I}(p)\neq\mathcal{I}(p')$. So $\mathcal{I}(q)\neq\mathcal{I}(q')$.) We can also see that every cell-level sequence constructed above induces a codeword in the code $\mathcal{C}(2n+v,m,D,v)$.

So corresponding to the $|\mathcal{S}(n,m,D,v)|$ codewords of the Star BRM code $\mathcal{S}(n,m,D,v)$, we can find at least $|\mathcal{S}(n,m,D,v)|(v!(m-2v)!)^{\frac{n}{m-v}}$ codewords of the BRM code $\mathcal{C}(2n+v,m,D,v)$. So the capacity of code $\mathcal{C}(n,m,D,v)$ is

$$\begin{aligned} cap(\mathcal{C}) &\ge \lim_{n\to\infty}\frac{\log|\mathcal{S}(n,m,D,v)|+(\log v!+\log(m-2v)!)\cdot\frac{n}{m-v}}{2n+v} \\ &= \lim_{n\to\infty}\frac{\log|\mathcal{S}(n,m,D,v)|}{2n}+\frac{\log v!+\log(m-2v)!}{2(m-v)} \\ &= \frac{cap(\mathcal{S})}{2}+\frac{\log v!+\log(m-2v)!}{2(m-v)} \\ &= \frac{\log Z^*+\log v!+\log(m-2v)!}{2(m-v)}. \end{aligned}$$

So the theorem is proved. □

Corollary 9.19 *Let $\mathcal{C}(n,m,D,v)$ be a BRM code, and let $\mathcal{S}(n,m,D,v)$ be a Star BRM code. Then, when $m \geq 2v$,*

$$cap(\mathcal{C}) \geq \frac{1}{2} \cdot cap(\mathcal{S}).$$

In particular, when $v > 1$ or $m > 2v$, $cap(\mathcal{C}) > \frac{1}{2} \cdot cap(\mathcal{S})$.

We now present a lower bound for the case $m < 2v$. Define $A_k^n = \binom{n}{k}k! = n!/(n-k)!$, which is the number of ways to arrange k elements in n positions. Suppose $m < 2v$ and $v = k(m-v)+s$, where $k \in \mathbb{N}^+$ and $1 \leq s \leq m-v$. Let $r = m-v-s$. (So $0 \leq r \leq m-v-1$.) Define a constant $M = A_s^{m-v}(A_{m-v}^{2(m-v)-s})^{k-1}(m-v)!$. We have the following lower bound for the BRM code when $m < 2v$.

Theorem 9.20 *For the BRM code $\mathcal{C}(n,m,D,v)$, when $m < 2v$ and $D \geq m+r$, its capacity*

$$cap(\mathcal{C}) \geq \frac{\log(Z^* \cdot M \cdot r!)}{m+r}$$

Proof: Use the notations in Theorem 9.18. For each block B_i, $i = 1,2,\cdots,n/(m-v)$, we first construct $k+2$ blocks $B_i^{(1)}, B_i^{(2)}, \cdots, B_i^{(k+2)}$. And then build a cell-level sequence $q = (B_1^{(1)}, B_1^{(2)}, \cdots, B_1^{(k+2)}, \cdots, B_{n/(m-v)}^{(1)}, B_{n/(m-v)}^{(2)}, \cdots, B_{n/(m-v)}^{(k+2)})$ of length $n' = (k+2)n + v = \frac{n(m+r)}{m-v} + v$, such that every two adjacent blocks overlap by v. Define $B_0^{(k+2)} = (1,2,\cdots,m-v,\ell_1^*,\ell_l^*,\cdots,\ell_v^*)$. Then the first v cell levels of $B_i^{(1)}$ are the same as the last v cell levels in $B_{i-1}^{(k+2)}$, $\forall\ i = 1,\cdots,n/(m-v)$, and the first v cell levels of $B_i^{(j)}$ are exactly the last v cell levels of $B_i^{(j-1)}$, $\forall\ j = 2,\cdots,k+2$. So we are left to build the last $m-v$ cell levels of $B_i^{(j)}$. Assign $(c_1', c_2', \cdots, c_{m-v}')$ to the last $m-v$ cell levels of $B_i^{(1)}$. For $B_i^{(2)}$, the $(v+1)$-th through the $(v+r)$-th cell levels take $r \leq D-m$ numbers from $[1,D]$ that are different from B_i. We have $r!$ choices here. And the last s cell levels are assigned s values from $\{\ell_1^*,\ell_l^*,\cdots,\ell_v^*\}$ that are different from the last $(k-1)(m-v)+s$ cell levels of $B_{i-1}^{(k+2)}$. Thus identical levels in a block are avoided and we have A_s^{m-v} choices here. For the last $m-v$ cell levels of $B_i^{(j)}$, $j = 3,\ldots,k+1$, we pick $m-v$ values from $\{\ell_1^*,\ell_l^*,\cdots,\ell_v^*\}$ that are not in the last $(m-v)(j-3)+s$ cell levels of $B_i^{(j-1)}$ nor in the last $(k-j+1)(m-v)+s$ cell levels of $B_{i-1}^{(k+2)}$. There are $A_{m-v}^{2(m-v)-s}$ choices for each $j = 3,\cdots,k+1$. Finally, the last $m-v$ cell levels of $B_i^{(k+2)}$ are chosen from $\{\ell_1^*,\ell_l^*,\cdots,\ell_v^*\}$ such that they are different from the last $(m-v)(k-1)+s$ cell levels of $B_i^{(k+1)}$, which results in $(m-v)!$ choices.

Notice q is a valid cell-level sequence of $\mathcal{C}(n', m, D, v)$ as each cell level is no more than D and any block in q has m different levels. For each codeword s in $\mathcal{S}$, there are at least $M \cdot r!$ such cell-level sequences, denoted by Q_s. Also notice that similar to Theorem 9.18, neither two cell-level sequences in Q_s nor two cell-level sequences in distinct Q_s and $Q_{s'}$ induce the same permutation sequence for $\mathcal{C}(n', m, D, v)$. Moreover, the number of distinct permutation sequences in $\mathcal{C}(n', m, D, v)$ is at least $\sum_{s \in \mathcal{S}} |Q_s| \geq |\mathcal{S}(n, m, D, v)|(M \cdot r!)^{\frac{n}{m-v}}$. Therefore,

$$\begin{aligned} cap(\mathcal{C}) &\geq lim_{n\to\infty} \frac{\log |\mathcal{S}(n, m, D, v)|(M \cdot r!)^{\frac{n}{m-v}}}{n'} \\ &= lim_{n\to\infty} \frac{(m-v)\log |\mathcal{S}(n, m, D, v)|}{n(m+r)} + \frac{\log(M \cdot r!)}{m+r} \\ &= \frac{(m-v)cap(\mathcal{S})}{m+r} + \frac{\log(M \cdot r!)}{m+r} \\ &= \frac{\log(Z^* \cdot M \cdot r!)}{m+r}. \end{aligned}$$

Thus we have proved the theorem. □

Corollary 9.21 *Let $\mathcal{C}(n, m, D, v)$ be a BRM code, and $\mathcal{S}(n, m, D, v)$ be a Star BRM code. Then if $m < 2v$ and $D \geq m + r$,*

$$cap(\mathcal{C}) > \frac{(m-v)cap(\mathcal{S})}{m+r} = \frac{cap(\mathcal{S})}{k+2}$$

9.6 Concluding Remarks

The question of what overlap provides the highest capacity for a given permutation size and a given maximum level is partially discussed in this chapter. Denote this optimal overlap by $v^*(D)$. The following observations show the two extreme cases and the result of Theorem 9.5:

1. When $D = m$, $cap(\mathcal{C}(n, m, m, v)) = \frac{log(m-v)!}{m-v}$. Therefore, $cap(\mathcal{C}(n, m, m, 0)) > cap(\mathcal{C}(n, m, m, 1)) > \cdots > cap(\mathcal{C}(n, m, m, m-1))$. We can conclude that $v^*(m) = 0$.

2. When $D = \infty$, it is clear that $cap(\mathcal{C}(n, m, \infty, v)) = \frac{\log(m!/v!)}{m-v}$. Then $cap(\mathcal{C}(n, m, \infty, 0)) < cap(\mathcal{C}(n, m, \infty, 1)) < \cdots < cap(\mathcal{C}(n, m, \infty, m-1))$, which implies that $v^*(\infty) = m - 1$

3. When $D \geq m + 2$, $cap(\mathcal{C}(n, m, D, 1)) > cap(\mathcal{C}(n, m, D, 0))$. So the optimal overlap for $D \geq m + 2$ satisfies $v^*(D) \geq 1$.

The optimal overlap values for $m+1 \leq D < \infty$ are not thoroughly examined, which can be our future work direction. Besides, for any rate no more than the capacity, the exact forms of the permutation encoders and decoders in addition to their efficiency and complexities are still left to be worked on. In addition, a generalized BRM code can be viewed as a set of n cells, among which we choose subsets of size m and form permutations. All cell levels are no more than D and two subsets may overlap. Under this framework, how to make choices of the subsets so as to optimize the capacity is still an open problem.

In summary, this chapter used the tools of labeled graphs to find the capacities of BRM codes with one overlap. In particular, it showed that if two extra charge levels are given, one can use them by way of overlap and achieve higher capacity than non-overlap codes. In addition, star BRM code is introduced to obtain a lower bound for the capacity of BRM codes.

Chapter 10

Partial Rank Modulation

10.1 Introduction

Similar to the previous chapter, we study rank modulation with low sorting complexity. However, instead of having small permutations along a long sequence of cells, we propose using the k highest out of n cells to represent information in flash memory, and call it *partial rank modulation.* For example, let $n = 5, k = 2$, we write $(3,4,|1,2,5)$ to represent a permutation, where the left most value corresponds to the cell index with highest level. The $|$ sign separates the top cells on the left which represents information, and the bottom cells on the right which is redundant. Sometimes we may simply write the top k cells only, e.g., $(3,4)$. Partial rank modulation is a good candidate of coding scheme for flash memory because it preserves the advantages of rank modulation, and the sorting of k cells is less complex than n cells ($k \log k$ complexity for k cells, or $k-1$ complexity with parallel comparisons). Also, the unused $n-k$ cells provide redundancy and can be used to correct errors.

We focus on the problem of Gray codes for such partial permutations, or a sequence traversing all partial permutations, for all $k < n$. The transition between every two permutations in this chapter is to change only one cell to the highest in the group. For example, let $n = 3, k = 2$, we actually consider full permutations, and write them without the $|$ for simplicity. The cycle of permutations is a Gray code: $(1,2,3),(2,1,3),(3,2,1),(1,3,2),(3,1,2),(2,3,1)$. All of the full permutations are contained in the cycle, and the transitions change only one cell.

In [JMSB09], a Gray code for the full permutations was constructed ($k = n-1$). The Gray code can be used as a permutation counter that traverses $0,1,\ldots,n!-1$, and by combining the counter with other coding techniques (e.g., floating code [JBB07] [FLM08]), we are able to represent arbitrary information efficiently. As mentioned before, one can use a counter as a logical cell

with $n!$ levels, and rewrite with a small increase in the levels each time. As a result, no block erasure is necessary until the logical cell reaches its maximum, which is much less frequent compared to a physical cell.

A related topic in combinatorics is universal cycles, which is a sequence containing all partial permutations exactly once. The construction of such universal cycles is an open problem since the 90s [CDG92]. Section 10.5 shows that the Gray code is equivalent to the universal cycles, so we will only concentrate on constructions of universal cycles in most of this chapter.

For a set Σ with n elements, a *permutation* of size n is an n-tuple $P = (p_n, \ldots, p_2, p_1)$, where $p_i \in \Sigma$, and p_i are all distinct, $i = 1, 2, \ldots, n$. For simplicity, Σ is taken as the set $[n] = \{1, 2, \ldots, n\}$ in the chapter. The indices are ordered decreasingly for convenience of description. Every permutation defines a mapping from Σ to itself, i.e., the i-th element of Σ is mapped to p_i. All of the permutations of size n form the symmetric group $\mathcal{S}_n$ with function compositions as operations. In this chapter, we are going to use $\mathcal{S}_n$ to represent the set of all permutations of size n. A *k-partial permutation*, or *k-permutation*, out of n elements is a k-tuple $Q = (p_k, \ldots, p_2, p_1)$ with k distinct elements in Σ, for some $k \leq n-1$. We say Q is induced from P if the first k elements of P are the same as Q.

A *de Bruijn sequence* is a cyclic sequence of some alphabet Σ where each possible k-tuple of this alphabet appears as a consecutive subsequence exactly once [dBE46]. The length of a de Bruijn sequence is $|\Sigma|^k$. For example, if $|\Sigma| = \{0, 1\}$ and $k = 3$, then 00010111 is a de Bruijn sequence. More generally, let C be a set whose elements are represented by k-tuples of the alphabet Σ. Then a *universal cycle* on the set C is a cyclic sequence of length $|C|$ for which every element in C corresponds to exactly one subsequence [CDG92].

For *universal cycles on k-partial permutations*, Σ is a set of size n, and C contains all the k-partial permutations, $k < n$. For example, $(4, 3, 4, 2, 3, 2, 4, 1, 3, 1, 2, 1)$ is a universal cycle for 2-partial permutations out of $n = 4$. It contains all the 2-partial permutations in its subsequences: $(4, 3), (3, 4), (4, 2), \cdots, (2, 1), (1, 4)$. A natural question to ask is: is there a universal cycle on partial permutations for any parameters k, n? We will show in this chapter that the answer is yes, and give explicit constructions of such cycles.

Universal cycles were first proposed in [CDG92], where the set C was considered to be all possible tuples, order-isomorphic permutations, partitions, or subsets. The existence of universal cycles on k-partial permutations was proved in [Jac93], however, the proof was nonconstructive. An explicit construction of a cycle for $k = n-1$ was given in [RW10]. But for other values

of k, the construction is still an open problem. In this chapter, we will provide an algorithm to construct the cycle for all $k \leq n-1$. The algorithm is dependent only on the top k elements of each permutation, and has complexity $O(k)$.

The problem of order-isomorphic permutations is similar to k-partial permutations because its elements in C is also k-partial permutations. However, two permutations are isomorphic (and therefore considered duplicated) if their relative ordering are the same. Namely, suppose $Q = (p_k, \ldots, p_2, p_1)$ and $Q' = (p'_k, \ldots, p'_2, p'_1)$ are isomorphic, then $p_i < p_j$ if and only if $p'_i < p'_j$, for $1 \leq i \neq j \leq k$. For example, $(1,3,2)$ and $(2,5,3)$ are isomorphic permutations. In [Joh09], constructions of universal cycles on order-isomorphic permutations are given for $n = k+1$, and it proves that $n = k+1$ symbols are sufficient for such universal cycles.

Another related problem is the Hamiltonicity of graphs of k-permutations with $n-k$ transitions ($k < n$), where each k-permutation corresponds to a vertex and each $k+1$-permutation corresponds to an directed edge. Namely, it is a universal cycle with distinctive elements in any consecutive $k+1$ subsequences. In [SKKC03], Hamiltonicity was shown for $k = 2$. And in [Isa06], Hamilton cycles are proved for all n and $k \leq n-3$, and for $n \geq 4$, $k = n-2$, the graph is proved to be not Hamiltonian.

Recently, Gray codes for local rank modulation, another variation of rank modulation, was raised in [EGLSB11]. And in local rank modulation, k-permutations with overlaps are extracted from a sequence of $m > k$ cells.

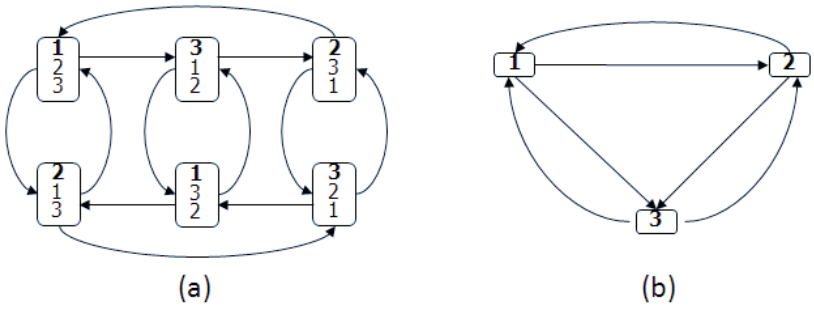

Figure 10.1: Directed graphs with $n = 3, k = 1$. Valid transitions are t_2, t_3. (a) The directed graph $\mathcal{G}$ of full permutations. A universal cycle contains a subset of the vertices. (b) The directed graph $\mathcal{H}$ of partial permutations. A universal cycle is a Hamiltonian cycle in this graph.

A universal cycle on partial permutations defines a Gray code, namely, a sequence of partial permutations, such that the transition between two adjacent permutations belongs to a set of valid transitions. Suppose there is a set of transitions (or a set of functions) defined on $\mathcal{S}_n$, then we can

draw a directed graph with all the permutations as vertices and all the transitions as edges. Denote this graph by $\mathcal{G}$. See Figure 10.1 (a) for an example. We would like to know if there exists a cycle in $\mathcal{G}$ whose vertices induce all k-partial permutations. In this chapter, we confine ourselves to the *push-to-the-top* transitions t_i, $i = k, k+1, \ldots, n$: for permutation $P = (p_1, p_2, \ldots, p_n)$, the i-th element becomes the first. That is, $t_i(P) = (p_i, p_1 \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$. Throughout the chapter, the left-hand side is considered as top, and the right side is bottom. Thus, in the graph $\mathcal{G}$, each permutation P has $n-k+1$ outgoing edges, $t_k(P), t_{k+1}(P), \ldots, t_n(P)$. If this cycle exits, we call it a *Gray code on k-partial permutations*. For example, $(1,2,3), (3,1,2), (2,3,1)$ is a Gray code for $k = 1$, $n = 3$. When $k = n-1$, the cycle consists of a Gray code on full permutations. The choice of push-to-the-top transitions relates the Gray codes to universal cycles on partial permutations.

In this chapter we will construct universal cycles, analyze the complexity of the proposed algorithm, and establish the relation between universal cycles and Gray codes.

10.2 Definitions and Notations

A *k-partial permutation*, or *k-permutation*, out of n element is a k-tuple $Q = (p_1, p_2, \ldots, p_k)$ with distinct k elements of the set $[n] = \{1, 2, \ldots, n\}$. The number of possible k-partial permutations is $\binom{n}{k}k!$. The partial permutation is said to be induced from the permutation $P = (p_1, p_2, \ldots, p_n)$, for any ordering of $p_{k+1}, p_{k+2}, \ldots, p_n$, and the permutation P is said to be a realization of Q. Note that there are multiple realizations for each partial permutation when $k < n-1$. Sometimes we will denote this permutation by $P = (p_1, p_2, \ldots, p_k | p_{k+1}, p_{k+2}, \ldots, p_n)$ to emphasize the first k elements. When n, k are known in the context, we usually refer to the *bottom elements* in P as the bottom $n-k$ elements, and the *top elements* as the top k elements.

An (n,k) *universal cycle of k-partial permutations* is a sequence $A = (a_1, a_2, \ldots, a_N)$, $N = \binom{n}{k}k!$, $a_i \in \{1, 2, \ldots, n\}$, such that each k-partial permutation out of n elements is represented by exactly one subsequence $(a_{i+1}, a_{i+2}, \ldots, a_{i+k})$ for some $1 \leq i \leq N$. In this section, the index additions are computed modulo N. For example, if $n = 4$, $k = 2$, the sequence $A = (4,3,4,2,3,2,4,1,3,1,2,1)$ has subsequences $(a_1, a_2) = (4,3), (a_2, a_3) = (3,4), \ldots, (a_{12}, a_1) = (1,4)$. It can be seen that every k-partial permutation is included in this sequence. Thus, it is a universal cycle for $n = 4$ and $k = 2$. The question is, does such a cycle exist for all n, k? If yes, how to find this cycle? We will see in the chapter that the answer is yes, and there are indeed a large number of such cycles. Moreover, we will give constructions of universal cycles.

Another way to view a universal cycle is to break it up to a sequence of k-permutations, $(a_1, a_2, \ldots, a_k)$, $(a_2, a_3, \ldots, a_{k+1}), \ldots, (a_N, a_1, \ldots, a_{k-1})$. For purpose of description, we write each permutation backwards. So a universal cycle is equivalent to a cycle of k-permutations $(a_k, \ldots, a_2, a_1)$, $(a_{k+1}, \ldots, a_3, a_2), \ldots, (a_{k-1}, \ldots, a_1, a_N)$ such that each partial permutation appear exactly once. Hence in this chapter we focus on constructing such a sequence of partial permutations.

For any k-permutation in the universal cycle, we can see that there are only $n-k+1$ possible permutations following it. We define the transition from one full permutation to the next as follows. For $k \leq i \leq n$, define mapping t_i: $\mathcal{S}_n \rightarrow \mathcal{S}_n$, where $t_i(p_1, p_2, \ldots, p_n) = (p_i, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$. t_i is called a push-to-the-top transition. By abuse of notation, t_i can be defined on k-permutations, where we expand $Q = (p_1, p_2, \ldots, p_k)$ to a full permutation $P = (p_1, p_2, \ldots, p_n)$ with $p_n > p_{n-1} > \cdots > p_{k+1}$. And define $t_i(p_1, p_2, \ldots, p_k) = (p_i, p_1, p_2, \ldots, p_{k-1})$ if $k \leq i \leq n$.

Therefore, a universal cycle is equivalent to the following graph interpretation. Consider a directed graph $\mathcal{H}$, with vertices corresponding to k-permutations and edges corresponding to t_i defined on k-permutations. Again, assume that $k \leq i \leq n$. Figure 10.1 (b) shows an example of such graphs. Then a Hamiltonian cycle in this graph is a universal cycle.

Typically, we assume $2 \leq k < n$ because when $k = 1$, construction of universal cycles is trivial (we can do t_n n times).

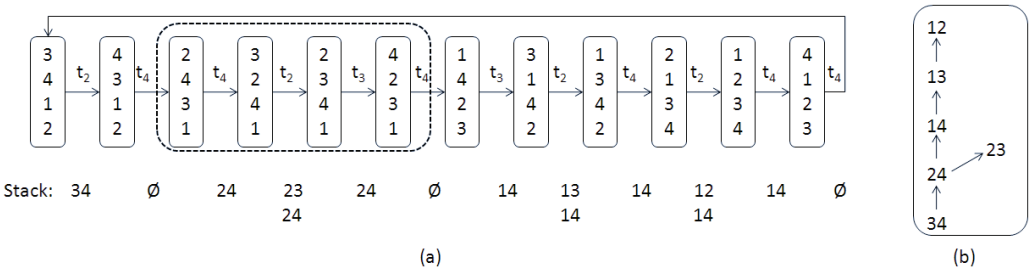

Figure 10.2: (a) A cycle for 2-partial permutations out of 4. The corresponding stack storing equivalence classes are shown below the cycle. (b) The insertion tree generating the cycle in (a).

Example 10.1 *Figure* 10.2*(a) shows a* $(4,2)$ *Gray code. Only the top two digits in each permutation are of concern. And the push-to-the-top transitions are indicated above each edge.*

We next define a partition on partial permutations, which will help us to construct the universal cycle. Define the partition by the relation $\sim$, where $P_1 \sim P_2$ if the first k elements of P_2 is a cyclic shift (to the left) of those of P_1. For example, if $k = 3$, $n = 6$, $E = \{(1,3,6), (6,1,3), (3,6,1)\}$

is an equivalence class. $\sim$ is obviously an equivalence relation. Let E be an equivalence class, if $P = (p_1, p_2, \ldots, p_k) \in E$, and $p_k = \max\{p_1, p_2, \ldots, p_k\}$, then we choose P as the representative of the equivalence class E (unless mentioned otherwise). We will denote $E = (p_1, p_2, \ldots, p_k)$. In the previous example, $(1,3,6)$ is the representative and we will write $E = (1,3,6)$. Each equivalence class has k members and there are $\binom{n}{k}(k-1)!$ equivalence classes.

10.3 Construction of Universal Cycles

As mentioned in Section 10.1, no explicit construction of universal cycles is known for $k < n-1$, and we will give such a construction for all $k \leq n-1$.

The universal cycles are constructed in several steps. We first construct a tree that contains all the equivalence classes, and then use this tree to generate all the partial permutations. Also, the construction of this tree is reduced to: (1) generating subtrees containing all the combinations of k elements out of n, and (2) finding a sequence of permutations on $k-1$ elements to connect the subtrees.

In the following, we will represent a universal cycle by its subsequences: $Q_1, Q_2, \ldots, Q_N$, where each Q_i is a k-permutation, and $N = \binom{n}{k}k!$. The transitions between the k-permutations are push-to-the-top operations t_i, $k \leq i \leq n$, or equivalently, push the i-th element p_i.

Notice that starting from any partial permutation, if we do $(k-1)$ times the transition t_k, we can traverse an equivalence class. If we start from some partial permutation in equivalence class E_1 and apply operations

$$\underbrace{t_k, \ldots, t_k}_{a \text{ times}}, t_i, \underbrace{t_k, \ldots, t_k}_{k-1 \text{ times}}, t_{k+1}, \underbrace{t_k, \ldots, t_k}_{k-2-a \text{ times}} \tag{10.1}$$

for $0 \leq a \leq k-2$ and $i > k$, we can still traverse all members of E_1. And if the partial permutation after t_i belongs to E_2, the above operations also traverse E_2. In another word, inserting E_2 does not affect the completion of E_1. We call the above operation *insertion*. See Figure 10.3 for an example. In addition, if $a = k-1$ in the above operation sequence, we omit the last operations t_{k+1} and $(k-2-a)$ times of t_k. And we traverse E_1, followed by traversing E_2. We consider this case as insertion, too. The permutations circled by dashed line in Figure 10.2(a) also illustrates this notion, where $E_1 = (2,4)$ is inserted by $E_2 = (2,3)$.

We will draw insertions in a directed tree. The root of the tree is the unique node with in-going degree 0, and an edge starts from the parent and ends at the child. The nodes of the tree are

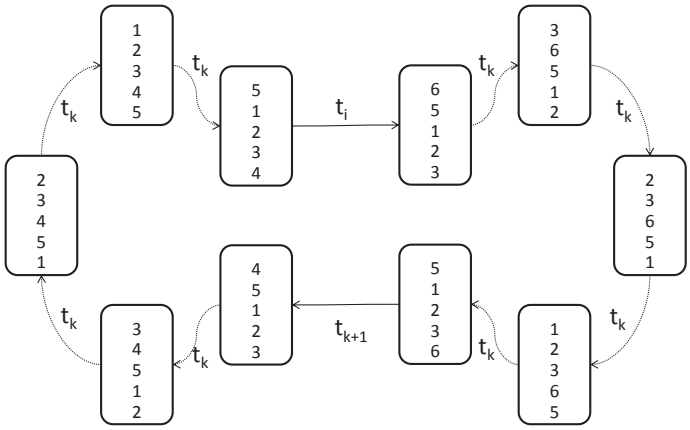

Figure 10.3: Insertion. Each box represents a k-permutation. The equivalence class $E_2 = (5,1,2,3,6)$ (the 5 permutations on the right) is inserted to $E_1 = (1,2,3,4,5)$ (the 5 permutations on the left). Here $k = 5$, $n \geq 6$, and t_i pushes 6 to the top.

equivalence classes. An edge means the child is inserted in the parent. Since transition t_i in 10.2 only changes one digit in the partial permutation, the parent and the child differ in only one digit for their representatives (after cyclic shift). We can insert at most k children to a parent, each changing one of its k digits. In a word, define an *insertion tree* to be a tree with equivalence classes as nodes, with k children at most for each parent, each changing one different digit of their parent. We may have multiple layers of insertions. The cycle in Figure 10.2(a) is generated by the insertion tree in Figure 10.2(b).

Lemma 10.2 *Any insertion tree of k elements out of n that contains each equivalence class once and only once will generate an (n,k) universal cycle. Such an insertion tree is called a generating tree.*

Proof: Claim: an insertion tree will induce a cycle of all members of the equivalence classes of the tree (possibly with duplicates if the tree has duplicated nodes).

Use induction on the depth. Base case is one single node, which is true by doing k times t_k. For a tree with depth of D, the subtrees of the root have depth of at most $D-1$ and, by induction, generate cycles of all members of their equivalence classes. Let the root be $(p_1, p_2, \ldots, p_k)$ (p_k is not necessarily the largest). For one subtree, suppose its root is $(p_1, p_2, \ldots, p_{i-1}, p_i', p_{i+1}, \ldots, p_k)$, then we can insert the subtree to the root and construct a cycle of k-permutations as follows: $(p_1, p_2, \ldots, p_k) \xrightarrow{t_k} \cdots \xrightarrow{t_k} (p_{i+1}, \ldots, p_k, p_1, \ldots, p_i) \xrightarrow{\text{push } p_i'} [(p_i', p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1}) \rightarrow \cdots \rightarrow (p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1}, p_i')] \xrightarrow{\text{push } p_i} (p_i, p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1}) \xrightarrow{t_k} \cdots \xrightarrow{t_k} (p_2, \ldots, p_k, p_1)$. The subsequence in the square brackets corresponds to the subtree. If $i = 1$, the part after

the square brackets is omitted, and the last permutation is $(p_2, \ldots, p_k, p_1')$. If $i \neq 1$, the last permutation is $(p_2, \ldots, p_k, p_1)$. In both cases, this sequence goes through the equivalence classes in the subtree and the root completely and it is indeed a cycle. Similarly, we can insert other subtrees into this cycle in the same manner. As each child changes a different digit of the root, these subtrees will not affect each other. So the claim is true.

Now as a generating tree contains each equivalence class exactly once, we have a cycle with every partial permutation exactly once. □

The above theorem is similar to the cycle merging technique in [Joh09], which was used to generate order-isomorphic permutations.

By the proof, if we start from some node m of the insertion tree, we can construct a cycle that traverse the subtree rooted at m in the following manner: (1) pass some of the k-partial permutations belonging to m first, and (2) traverse one of the subtrees that is rooted at a child of m, and repeat (1) and (2) until the k-partial permutations belonging to m is traversed.

This procedure works like a stack. That is, when we go to the current node m, put it in the stack. And then pass some of its corresponding partial permutations. If every element in the equivalence class of m is passed through, remove m from the stack. Next, go to a child of m, and treat the child as the current node. If the current node has been traversed, and has no children, then visit a node from the stack. One cycle ends when the stack is empty, and we can start again by pushing the first element of the first permutation to the top. For example, in Figure 10.2(a), the stack after each partial permutation is written in the lower dashed box.

Now the problem of universal cycles for partial permutations is reduced to constructing generating trees.

Before going through constructions, let us think about whether such generating trees exist. Consider an insertion tree of k out of n in Figure 10.4. In this figure, the first digit in the representative of an equivalence class does not have to be the largest. Each rectangular represents a subtree with all equivalence classes that have one digit fixed as some integer, and possibly duplicates occur in each rectangular. An X in the graph represents a non-fixed digit. Given a tree of all equivalence classes of $(k-1)$ out of n, we can add the fixed integer at the fixed position in each node, and connection rules for insertion trees are not violated. Thus we obtain a subtree in the rectangular. By induction on k for fixed n, these subtrees exist. The equivalence classes $(1, 2, \ldots, k-1, k+1), (1, 2, \ldots, k-1, k+2), \ldots, (1, 2, \ldots, k-1, n)$ are nodes in the subtree $1XX\ldots X$, and each of these nodes leads to a new subtree.

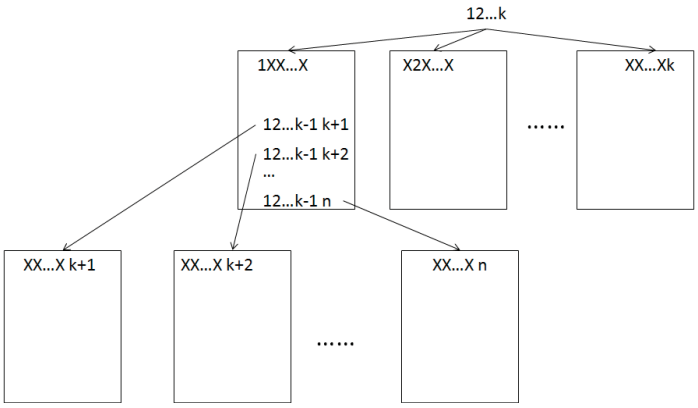

Figure 10.4: An insertion tree of k out of n, which contains all equivalence classes but also allows duplicates.

Obviously the tree in Figure 10.4 contains every equivalence class, but there are some duplicates. Starting from bottom to up, if there are two identical class $E_1 = E_2$ in this tree, we can remove E_2 and move all its decedents under E_1 (treat them as the descendants of E_1). If thus E_1 has 2 children that change the same digit of E_1, then these two children differ in only one digit. Thus we can move one child and its decedents under the other. Repeat this procedure until all descendants of E_1 have valid children. Afterwards, remove the next duplicate, and so on. Therefore, generating trees exist, and as we can imagine, there are many such trees. A proof of existence of universal cycles using graph theory can be found in [Jac93].

We will give a special class of generating trees based on combinations, where the elements are ordered increasingly. Let $(c_1, c_2, \ldots, c_n)$ and $(d_1, d_2, \ldots, d_n)$ be vectors with distinct values. Then they are *order-isomorphic* if for all $1 \leq i < j \leq n$, $c_i < c_j$ if and only if $d_i < d_j$. Suppose we have an insertion tree T_1 that contains each combination of k out of n once, and the nodes in T_1 are all ordered increasingly. Namely, each node $(p_1, p_2, \ldots, p_k)$ in T_1 satisfies $p_1 < p_2 < \cdots < p_k$. Suppose the root of the tree is $(n-k+1, n-k+2, \ldots, n)$. Then we will construct trees $T_2, T_3, \ldots$ such that they have similar structure as T_1, but the nodes do not have increasing order. Moreover, these trees will be connected by permutations $\sigma_1, \sigma_2, \sigma_3, \ldots$ on $[k-1]$, where σ_1 is fixed to be the identity permutation. Obviously, all nodes in T_1 are order-isomorphic to $(\sigma_1, k) = (1, \ldots, k-1, k)$. For an integer x and a permutation σ on $[k-1]$, write $\sigma + x = (\sigma(1) + x, \sigma(2) + x, \ldots, \sigma(k-1) + x)$. If we can transit from a node in T_1 to the node $(\sigma_2 + (n-k), n)$ and does not violate the rule of an insertion tree (a child changes one digit from the parent and each child changes a different digit), then we can use this as the root for a new tree T_2. And by the same structure as T_1, except

that we change each $(p_1, p_2, \ldots, p_k)$ in T_1 to $(p_{\sigma_2(1)}, p_{\sigma_2(2)}, \ldots, p_{\sigma_2(k-1)}, p_k)$ in T_2, we get another insertion tree T_2 and each node in T_2 is order-isomorphic to (σ_2, k). After that, we transit from T_2 to T_3 with permutation σ_3, and each node in T_1 is replaced by $(p_{\sigma_3(1)}, p_{\sigma_3(2)}, \ldots, p_{\sigma_3(k-1)}, p_k)$, and so on.

For example, in Figure 10.5 ($n = 6, k = 3$), the subtree on the left (in dashed lines) is tree T_1, and we transit from T_1 to a node $(5, 4, 6)$. Hence $\sigma_1 = (1, 2)$ and $\sigma_2 = (2, 1)$. The node in T_1, for instance, $(p_1, p_2, p_3) = (2, 4, 6)$, is replaced with $(p_{\sigma_2(1)}, p_{\sigma_2(2)}, p_3) = (p_2, p_1, p_3) = (4, 2, 6)$. Each node in T_2 is isomorphic to $(\sigma_2, k) = (2, 1, 3)$.

We have the following theorem.

Theorem 10.3 *If there is an insertion tree T_1 that contains every combination of k out of n once, and a sequence of valid transitions from previous trees (as described above), $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$, traversing all permutations on $[k-1]$, then $T_1 \to T_2 \to \cdots \to T_{(k-1)!}$ is a generating tree of k-partial permutations.*

Proof: We first show that this three is an insertion tree. Since the transition within T_1 and from one subtree to the next is assumed to be valid, we need to show each child in T_i changes only one digit of the parent and changes a different digit, for $i \geq 2$. However, the nodes in T_i and T_1 are identical except the ordering of their elements are permutated by σ_i, so T_i is also a valid insertion tree.

Next we show that the tree contains every equivalence class exactly once. Recall that we represent the equivalence classes with $(p_1, \ldots, p_k)$ where $p_k = \max\{p_1, \ldots, p_k\}$. For any $p_1 < p_2 < \cdots < p_k$, and any equivalence class $(p_{\sigma(1)}, p_{\sigma(2)}, \ldots, p_{\sigma(k-1)}, p_k)$, there exists exactly one $1 \leq i \leq (k-1)!$, such that $\sigma = \sigma_i$ and this equivalence class appears exactly once in the tree T_i. □

Now the problem of finding universal cycles for partial permutations reduces to finding T_1 and $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$.

Construction 10.4 *(Construction of T_1)*
The root $(n-k+1, n-k+2, \ldots, n)$ has one child $(n-k, n-k+2, \ldots, n)$, and the node $(1, 2, \ldots, k)$ has a parent $(1, 2, \ldots, k-1, k+1)$ and no children. Starting from the root, and following the connection rules below for all the other nodes, we will get a T_1.

1. *A node $(p_1, p_2, \ldots, p_{k-n+t-1}, t, t+1, \ldots, n)$, for $1 \leq p_{k-n+t-1} < t-1$ and $n-k+2 \leq t \leq n$, is connected to the at most 3 nodes:*

Parent $(p_1, p_2, \ldots, p_{k-n+t-1} + 1, t, t+1, \ldots, n)$

Child$_1$ $(p_1, p_2, \ldots, p_{k-n+t-1} - 1, t, t+1, \ldots, n)$ *if* $p_{k-n+t-1} - 1 > p_{k-n+t-2} \geq 1$

Child$_2$ $(p_1, p_2, \ldots, p_{k-n+t-1}, t-1, t+1, \ldots, n)$

2. *Otherwise, a node* $(p_1, p_2, \ldots, p_k)$ *is connected to at most two nodes:*

Parent $(p_1, p_2, \ldots, p_k + 1)$

Child $(p_1, p_2, \ldots, p_k - 1)$ *if* $p_k - 1 > p_{k-1}$

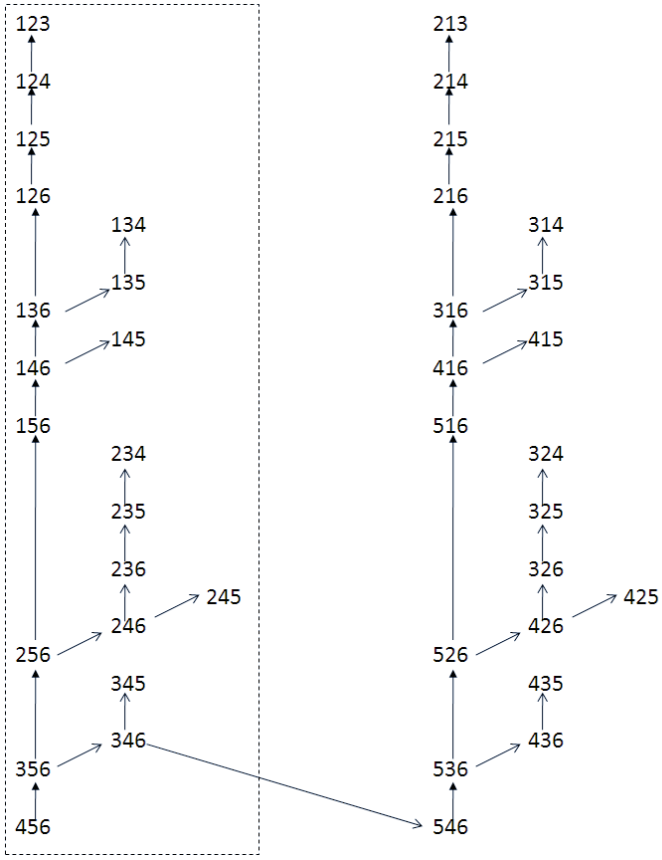

Figure 10.5: A generating tree of 6 out of 3 based on Construction 10.4 and 10.6. The σ_1, σ_2 are $12 \xrightarrow{\alpha_2} 21$

Figure 10.2(b) is one example of Construction 10.4. Another example of 6 out of 3 is shown in Figure 10.5. T_1 is the left part of the tree in dashed line. T_1 is similar to listing combinations lexicographically.

Theorem 10.5 *Construction* 10.4 *forms a tree* T_1 *that contains every combination of k out of n.*

Proof: The construction already includes every combination. So we need to prove that T_1 is a connected graph with no cycles.

Starting from any node in T_1, and tracing back the parent, is equivalent to increasing the k-th digit to the maximum, and then increasing the $(k-1)$-th digit to the maximum, and so on. All of these backward paths will end at $(n-k+1, n-k+2, \ldots, n)$. Therefore, graph T_1 is connected. Moreover, as a node is always smaller than its parent in lexicographical order, there is no cycle in T_1. Hence T_1 is a tree. □

It is easy to observe that T_1 can be viewed as a subtree of $k-1$ out of $n-1$ combined with a subtree of k out of $n-1$. Namely, it is formed by a tree with the digit 1 and a tree without 1. And each subtree can be further decomposed into two. Therefore, T_1 can be constructed recursively.

Another possible T_1 is a single line, where each parent has only one child. There are several different constructions in the literature. For example, the homogeneous scheme and the near-perfect scheme can be found in [Knu05]. Here we list two examples for $n = 6, k = 3$ taken from [Knu05]. A homogeneous example: $123, 124, 134, 234, 235, 135, 125, 145, 245, 345, 346, 146, 246, 236, 136, 126, 156, 256, 356, 456$. A near-perfect example: $456, 256, 156, 356, 346, 246, 146, 126, 136, 236, 234, 134, 124, 123, 125, 135, 235, 245, 145, 345$. Here a node is the parent of its neighbor on the right.

For construction of $\sigma_1, \ldots, \sigma_{(k-1)!}$, we will assume $k \geq 3$, because otherwise only T_1 itself is the generating tree. We first find which transitions are allowed from subtree T_t to T_{t+1}. Consider the node in T_t,

$$a = (p_1, p_2, \ldots, p_{i-1}, n-k, p_{i+1}, \ldots, p_{k-1}, n)$$

$$\text{where } \{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{k-1}\} = \{n-k+1, n-k+2, \ldots, n-1\} \setminus \{y\} \qquad (10.2)$$

for some $n-k+2 \leq y \leq n-1$. This node exists since $n \geq k+1$ and $k \geq 3$. We can now define a child of a as $b = (p_1, p_2, \ldots, p_{i-1}, y, p_{i+1}, \ldots, p_{k-1}, n)$. Let $x = y - n + k$ and b be the root of T_{t+1}. Since (σ_t, k) is isomorphic to a and (σ_{t+1}, k) is isomorphic to b, we can see that $\sigma_{t+1} = \alpha_x \circ \sigma_t$, where $\alpha_x = (x, 1, 2, \ldots, x-1, x+1, \ldots, k-1)$. Here $\circ$ is the composition of permutations and is computed from right to left.

From Construction 10.4, we know that node a has only one child c in T_t (we need to order elements in a increasingly first), which changes $y+1$ of a to y. And b changes $n-k$ of a to y. Thus, children b and c are valid for the parent a, since they change different digits of a. Hence, α_x is a valid transition from tree T_t to T_{t+1}.

Permuted by α_x, the lowest position in σ_t becomes the x-th lowest position in σ_{t+1}. In other words, from the dual (inverse permutation) of σ_{t+1} to the dual of σ_t, we push the x-th element to the top.

Construction 10.6 *(Construction of $\sigma_1, \ldots, \sigma_{(k-1)!}$)*
Suppose $\mathcal{C}$ is a cycle of full permutations of $[k-1]$ with push-to-the-top transitions. First take the dual of each permutation in $\mathcal{C}$. Then replace each push-to-the-top operation t_x with α_x and reverse the cycle direction. And thus we will get a cycle of $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$, generated by operations α_x, $1 < x \leq k-1$. Starting from the identity permutation σ_1, and ignoring the last transition, we get a path of $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$.

$\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$ can be easily generated once we have recursively generated $(k-2)$-partial permutations out of $k-1$ elements (or full permutations on $k-1$ elements) using the constructions above. We can also use the cycle of full permutations in [JMSB09] [RW10]. For example, if $k = 4$, then Construction 10.4 will form a cycle $\mathcal{C}$ on 3 elements using the operation sequence $t_2, t_3, t_3, t_2, t_3, t_3$. And the push-to-the-bottom sequence is $231 \xrightarrow{t_2} 321 \xrightarrow{t_3} 132 \xrightarrow{t_3} 213 \xrightarrow{t_2} 123 \xrightarrow{t_3} 312 \xrightarrow{t_3} 231$. The corresponding σ_i's are $312 \xleftarrow{\alpha_2} 321 \xleftarrow{\alpha_3} 132 \xleftarrow{\alpha_3} 213 \xleftarrow{\alpha_2} 123 (\xleftarrow{\alpha_3}) 231 \xleftarrow{\alpha_3} 312$. Deleting the arrow in brackets, we get a path.

Notice that if we construct σ_i's using a $(k-1, k-2)$ universal cycle, then the push-to-the-top transitions are only t_x, with $x = k-1$ or $x = k-2$. Therefore, $y = x + n - k$ in (10.2) is either $y = n-1$ or $y = n-2$. Thus, if we want to recursively construct our (n,k) universal cycle, there are only two possible nodes in each subtree that may lead to a new subtree.

The following example shows how to decide the next subtree, given a permutation. Assume $n = 7, k = 4$, and the current k-partial permutation is $(5,4,6,1)$. This node belongs to the equivalence class $(1,5,4,6)$, and we know σ_t is $(1,3,2)$ for the current subtree. The dual of $(1,3,2)$ is itself. To find the next subtree, we need to find the permutation before $(1,3,2)$ in a cycle of permutations of size 3. By the previous example, it should be $(3,2,1)$, whose dual is itself again. Thus, the next subtree should have $\sigma_{t+1} = (3,2,1)$. The root of the next subtree is $(6,5,4,7)$.

Having constructed T_1 and $\sigma_1, \ldots, \sigma_{(k-1)!}$, we are able to draw a generating tree. Figure 10.5 shows a generating tree of 3 out of 6. The sequence σ_1, σ_2 is $12 \xrightarrow{\alpha_2} 21$, and the transition node from T_1 to T_2 is $(3,4,6)$.

Knowing a generating tree and the current partial permutation, how shall we decide the operation for the next step? Define w_m to be the *position of the newly changed digit* from the parent to the

node m, and the position is counted from top to bottom. More specifically, let $m = (p_1, \ldots, p_k)$, then its parent is $(p_1, \ldots, p_{w_m-1}, p'_{w_m}, p_{w_m+1}, \ldots, p_k)$, where $p_{w_m} \neq p'_{w_m}$. Its parent defers from m only at position w_m. For the root, w_{root} can be any integer between 1 and k. But we will assign $w_{root} = k$. For example, in Figure 10.5, $(1,2,4)$ is the parent of the node $m = (1,2,3)$, and the 3rd digit is changed from 4 to 3, so $w_m = 3$, $p_{w_m} = p_3 = 3$.

Let the current partial permutation be $(p_j, \ldots, p_k, p_1, \ldots, p_{j-1})$ and the current equivalence class be $m = (p_1, p_2, \ldots, p_k)$. The following algorithm is designed based on Lemma 10.2 and finds the next partial permutation for an arbitrary generating tree:

Algorithm 10.7 *(Transition from any partial permutation)*

- *If m has no children,*
  - *if $j \equiv w_m + 1$ mod k, then do step A;*
  - *else, do t_k.*
- *Else (if m has children),*
  - *if $j \equiv w_y + 1$ mod k for some child $y = (p_1^{(y)}, p_2^{(y)}, \ldots, p_k^{(y)})$, push $p_{w_y}^{(y)}$ to the top;*
  - *else if $j \equiv w_m + 1$ mod k, then do step A;*
  - *else do t_k.*

Step A: find the parent of m, say z. If $w_z = w_m$, find the parent of z, repeat until $w_l \neq w_m$ for some ancestor $l = (p_1^{(l)}, p_2^{(l)}, \ldots, p_k^{(l)})$. Then, push $p_{w_m}^{(l)}$ to the top. If no such ancestor exists, we are at the end of a cycle, so push the k-th element of the root to the top.

Thus at any moment with any partial permutation, we first find out which equivalence class it is in, and then we have a mechanical way to decide the next step. This algorithm does not need the ordering of the entire permutation, but only the first k. At each step we either push the k-th highest element to the top, or push some particular element (e.g., $p_{w_y}^{(y)}$) to the top. The ranking of this particular element does not matter.

If we use a recursive construction for α_x in Construction 10.6 and for T_1 in Construction 10.4, Algorithm 10.7 can be simplified. In the following, Algorithm 10.8 or $G(n,k)$ decides the transition in a universal cycle, and 10.9 or $H(n,k)$ decides the transition in a reverse universal cycle. Namely, given a current permutation, $G(n,k)$ decides the next transition and $H(n,k)$ decides the previous transition.

Given a permutation, take the top k elements $(q_1, q_2, \ldots, q_k)$ and suppose $q_l = \max_{i=1}^{k} q_i$. Order the top k elements increasingly as $m = (p_1, p_2, \ldots, p_k)$. Let $Q = (q_{l+1}, q_{l+2}, \ldots, \ldots, q_k, q_1, \ldots, q_{l-1})$. Let σ be isomorphic to Q and $\sigma \in \mathcal{S}_{k-1}$ be a permutation on $[k-1]$. Let π be the dual of σ.

Algorithm 10.8 *(Universal cycle $G(n,k)$)*

1 *(Transit to the next subtree) If m satisfies (10.2), with $y = n-1$ or $y = n-2$, and $q_k = n-k$, find π and compute the transition t_x before π using $H(k-1,k-2)$.*

   - *If $x = y-n+k$ and $(y, q_1, \ldots, q_{k-1}) \neq (y, \ldots, n, n-k+1, \ldots, y-1)$, push y to the top.*
   - *Else, go to step 3.*

2 *(Transit to the previous subtree) Else if $m = (n-k+1, n-k+2, \ldots, n)$, $Q \neq (n-k+1, \ldots, n-1)$, and $q_k = n-1$ or $n-2$, find π and compute the transition t_x after π using $G(k-1,k-2)$. Let $y = x+n-k$.*

   - *If $q_k = y$, push $n-k$ to the top.*
   - *Else, go to step 3.*

3 *(Transit within a subtree) If $m = (p_1, p_2, \ldots, p_{k-n+t-1}, t, t+1, \ldots, n)$, for $1 \leq p_{k-n+t-1} < t-1$ and $n-k+2 \leq t \leq n$,*

   - *If $q_k = t$, push $t-1$ ($Child_2$).*
   - *Else if $q_k = p_{k-n+t-1}$ and $p_{k-n+t-1} - 1 > P_{k-n+t-2} \geq 1$, push $p_{k-n+t-1} - 1$ ($Child_1$).*
   - *Else if $q_k = p_{k-n+t-1}$, push $t-1$ (Ancestor).*
   - *Else, do t_k.*

   Else,

   - *If $q_k = p_k$ and $p_k - 1 > p_{k-1}$, push $p_k - 1$ (Child).*
   - *Else if $q_k = p_k$, push n (Ancestor).*
   - *Else, do t_k.*

Step 3 corresponds to the two cases in Construction 10.4. $H(n,k)$ finds the previous permutation and is very similar to $G(n,k)$ and has the same complexity as $G(n,k)$. Let the current permutation be $(q_1, q_2, \ldots, q_n)$. Also, we need to compute $G(k-1, k-2)$ in step 2. We can either use Algorithm 10.8 or the following simplified algorithm. The following algorithms realizes $H(n, n-1)$ (and $G(n, n-1)$) and finds its previous (and next) transition, which should be either t_n or t_{n-1}. Define Q and π in the same way as in Algorithm 10.8.

Algorithm 10.9 *(Reverse universal cycle H(n,n-1))*

1 *(Transit to the previous subtree) If* q_1*=1, and* $q_n = n-1$ *or* $n-2$*, find* π *and compute the previous* t_x *of* π *using* $H(n-2, n-3)$.

   - *If* $x = q_n - 1$ *and* $(q_2, \ldots, q_n) \neq (q_n + 1, \ldots, n, 2, \ldots, q_n)$ *output* t_n.
   - *Else, go to step 3.*

2 *(Transit to the next subtree) Else if* $q_n = 1$, $Q \neq (2, \ldots, n)$, *and* $q_1 = n-1$ *or* $n-2$*, find* π *and compute the next* t_x *of* π *by* $G(n-2, n-3)$.

   - *If* $q_1 = x + 1$, *output* t_n.
   - *Else, go to step 3.*

3 *(Transit with in a subtree)*

   - *If* $q_1 - q_n = 1$ *or* -1, *output* t_n.
   - *Else, output* t_{n-1}.

Algorithm 10.10 *(Universal cycle G(n,n-1))*

1 *(Transit to the next subtree)*
   If q_{n-1}*=1, and* $q_n = n-1$ *or* $n-2$*, find* π *and compute the previous* t_x *of* π *using* $H(n-2, n-3)$.

   - *If* $x = q_n - 1$ *and* $(q_n, q_1, \ldots, q_{n-2}) \neq (q_n, \ldots, n, 2, \ldots, q_n - 1)$ *output* t_n.
   - *Else, go to step 3.*

2 *(Transit to the previous subtree)*
   Else if $q_n = 1$ *and* $Q \neq (2, \ldots, n)$*, find* π *and compute the next* t_x *of* π *by* $G(n-2, n-3)$.

- *If* $q_{n-1} = x + 1$, *output* t_n.
- *Else, go to step 3.*

3 (Transit with in a subtree)

- *If* $q_{n-1} - q_n = 1$ *or* -1, *output* t_n.
- *Else, output* t_{n-1}.

Figure 10.2(a) is an example of the above algorithms with $n = 4$, $k = 2$.

10.4 Complexity Analysis

In this section, we analyze the computational complexity of Algorithm 10.8 10.9 and 10.10 and compare them with some previous constructions of subtree T_1 and cycle of full permutations $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$.

Even though the algorithm $G(n,k)$ is recursive, we will show the number of recursions is low on average. To compute $G(n,k)$, in the worst case, we need to call $H(k-(2i-1),k-2i)$, for all $1 \leq i \leq \lfloor k/2 \rfloor$. There are 4 partial permutations in each subtree (and in total $4(k-1)!$ partial permutations) that we need to compute α_x using $H(k-1,k-2)$ or $G(k-1,k-2)$. Thus, the $(k-1,k-2)$ cycle is computed 4 times, leading to a total of $4(k-1)!$ recursive iterations.

To compute the $(k-1,k-2)$ cycle, $H(k-3,k-4)$ and $G(k-3,k-4)$ are called $4^2(k-3)!$ times. Similarly, the iteration of the $(k-(2i-1),k-2i)$ cycle is computed $4^i(k-2i+1)!$ times, for $1 \leq i \leq \lfloor k/2 \rfloor$. Thus, the total number of calls of G and H is

$$\sum_{i=1}^{\lfloor k/2 \rfloor} 4^i(k-2i+1)! < \sum_{i=1}^{\lfloor k/2 \rfloor} (k-i)(k-i)! \leq \sum_{i=1}^{k} (k-i)(k-i)! = k! - 1$$

And the average number of calls of function G or H for all the permutations is less than

$$1 + \frac{k!}{\binom{n}{k} k!} \to 1$$

when $n \to \infty$. Therefore, we only need to run G or H once on average.

Hence, the average computational complexity of $G(n,k)$ is only dependent on the non-recursive operations. In particular, recognizing the form of m (such as if $m = (p_1, p_2, \ldots, p_{k-n+t-1}, t, t+1, \ldots, n)$, for some $1 \leq p_{k-n+t-1} < t-1$ and $n-k+2 \leq t \leq n$) does not require the exact ordering of m, if we have $O(k)$ auxiliary space $(u_{n-k+1}, u_{n-k+2}, \ldots, u_n)$. For all $i = 1, 2, \ldots, k$, if

$q_k \geq n-k+1$, set $u_{q_k} = 1$. If $u_n = 1$, then m is in the form $(p_1, p_2, \ldots, p_{k-n+t-1}, t, t+1, \ldots, n)$. And $t-1$ is the largest index such that $u_t = 0$. In this way, $O(k)$ time and $O(k)$ space are needed.

To compute the dual permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{k-1})$, we need to find the maximum $q_l = \max_{i=1}^{k} q_i$, which is $O(k)$ time. Then for $i = 1, 2, \ldots, k-l$, if $q_{i+l} = q$, $\pi_{q-1} = i$; for $i = k-l+1, \ldots, k-1$, if $q_{i-k+l} = q$, $\pi_{q-1} = i$. These operations take $O(k)$ time and $O(k)$ space.

The complexity of the rest is $O(1)$. Therefore, we can conclude that $G(n,k)$ has computational complexity $O(k)$ in both time and space.

It should be noted that if $k = n-1$, in the algorithm for $H(n, n-1)$ or $G(n, n-1)$, we only need to check if $q_1 - q_n = \pm 1$ in step 3. Therefore, the average complexity is $O(1)$ in time and space.

We do not store the value y in step 1 and 2 for each subtree in Algorithm 10.8 because in flash memory we prefer no auxiliary storage and use storage only for the data (or permutation in our case). If we were allowed to store y, then the number of recursions can be further reduced.

For comparison, let us consider homogeneous and near-perfect algorithms [Knu05] for generating all (n,k) combinations and replace Construction 10.4. Both homogeneous and near-perfect algorithms require sorting the k elements in a partial permutation first. If we insist on linear computation time, then this will take $O(n)$ space and $O(k)$ time. And if we do not allow auxiliary storage as mentioned in the previous paragraph, these two algorithms are both recursive and take $O(k)$ time. The overall complexity is $O(k)$ time and $O(n)$ space.

In addition, we can compare Algorithm 10.9 with the algorithms in [JMSB09] and [RW10] for generating the σ_i's. All algorithms require $O(1)$ time on average. However, an $(n, n-1)$ permutation cycle in [JMSB09] has transitions t_i, $i = 2, 3, \ldots, n$. And therefore we will need to check $2(k-2)$ partial permutations in step 1 and 2 in each subtree in Algorithm 10.8, instead of 4 as in Algorithm 10.9.

The comparison is summarized in Figure 10.6. One advantage of the algorithms in [Knu05] [JMSB09] [RW10] is that one can find an explicit mapping between integers and combinations (or permutations) according to the cycle, even though it is still not clear how to map integers to partial permutations.

subtree alg.	space	time
Cnstr. 10.4	O(k)	O(k)
homogeneous	O(n)	O(k)
near-perfect	O(n)	O(k)

σ_i alg.	space	time	recursions
Alg. 10.9	O(1)	O(1)	4
[JMSB09]	O(1)	O(1)	2(k-2)
[RW10]	O(1)	O(1)	4

Figure 10.6: Comparison of algorithms for (n,k) universal cycle. Space and time are average computational complexity. Recursions shows the number of partial permutations in each subtree that needs recursive computation.

10.5 Equivalence of Universal Cycles and Gray Codes

As mentioned before, universal cycles are related to Gray codes, which have applications in codes for flash memories, therefore, we will study Gray codes in this section.

Consider a directed graph $\mathcal{G}$, whose vertices are permutations of length n. There is a directed edge if there is a push-to-the-top transition t_i from one vertex to another, for $k \leq i \leq n$. So each vertex in $\mathcal{G}$ is restricted to $n-k+1$ outgoing edges. An (n,k) *Gray code on partial permutations* is a cycle in $\mathcal{G}$ whose vertices induce every k-partial permutation once and only once. Note that this cycle contains only a subset of the vertices in $\mathcal{G}$. A Gray code contains $N = \binom{n}{k}k!$ permutations, and we denote them by $\mathbf{P} = (P_1, P_2, \ldots, P_N)$. It should be noted that each P_i is a full permutation.

In Gray code we only allow t_i, for $k \leq i \leq n$. The constraint that $i \geq k$ is imposed for two reasons: (a) these are the valid transitions in universal cycles, and (b) in flash memory, to minimize the effect of leakage. The first reason will be revealed in this section. Notice that if $2 \leq i < k$, then after the operation t_i, there will be a charge gap between the p_{i-1}-th and the p_{i+1}-th cell. As information is stored in the highest k cells, we want to keep these k cells close in charge levels. When large deflation happens, lower charge levels may decrease to none at all, and gap among the first k cells may cause an error.

Recall the graphic description of universal cycles in 10.2 as a Hamiltonian cycle in graph $\mathcal{H}$. $\mathcal{H}$ and $\mathcal{G}$ differ in the fact that in $\mathcal{G}$, we require the bottom $n-k$ elements of a permutation to be consistent along the cycle, while in $\mathcal{H}$ the bottom $n-k$ elements can be in arbitrary order. For example, if $n=4$, $k=2$, in $\mathcal{G}$, $(1,3|2,4) \rightarrow (4,1|3,2)$ is realized by pushing value 4 to the top. But in $\mathcal{H}$, $(1,3) \rightarrow (4,1)$ can be induced by the permutations $(1,3|2,4) \rightarrow (4,1|2,3)$, which is not a valid transition in $\mathcal{G}$.

In this section, we will show that the universal cycle is equivalent to the Gray code on partial permutations using transitions t_i, $k \leq i \leq n$. For example, it is easy to see that the Gray code in Figure 10.2 induces the universal cycle $A = (4,3,4,2,3,2,4,1,3,1,2,1)$. However, it is not clear

whether a universal cycle corresponds to exactly one Gray code, because we may change the bottom $n-k$ elements in each permutation and get another universal cycle, or the bottom $n-k$ elements are not consistent so there is no Gray code. We will show in the section that the Gray code is indeed unique.

Define a *path* of elements as a sequence of elements $(a_1, a_2, \ldots, a_N)$ where subsequences are taken without wrapping around, e.g., $(a_N, a_1, \ldots, a_{k-1})$ is not considered as a subsequence. Similarly, a *path* of permutations is a sequence of elements $(P_1, P_2, \ldots, P_N)$ where the transition from P_N to P_1 is not considered. Moreover, the transitions between permutations are always assumed to be push-to-the-top operation, and only the bottom $n-k$ elements are pushed. On the other hand, a *cycle* of elements or permutations is a sequence with wrapping around.

Lemma 10.11 *Let $(P_1, P_2, \ldots, P_N)$ be an (n,k) Gray code, then there is a unique (n,k) universal cycle corresponding to it.*

Proof: Given the Gray code, suppose $P_1 = (a_k, a_{k-1}, a_1 | b_1, b_2, \ldots, b_{n-k})$ and let $b_0 = a_1$ be the k-th element of P_1. Assume $P_2 = t_{j+k}(P_1)$ for some $0 \leq j \leq n-k$. Then $P_2 = (b_j, a_k, a_{k-1}, \ldots, a_2 | b_0, \ldots, b_{j-1}, b_{j+1}, \ldots, b_{n-k})$ if $j > 0$ and $P_2 = (b_j, a_k, a_{k-1}, \ldots, a_2 | b_1, b_2, \ldots, b_{n-k})$ if $j = 0$. Define $a_{k+1} = b_j$ as the top element of P_2. Then $(a_1, a_2, \ldots, a_k, a_{k+1})$ contains two k-permutations (without wrapping around). In the same manner, let a_{k+i} be the top element of P_{i+1}, $i = 1, 2, \ldots, N-k$. Then $(a_1, a_2, \ldots, a_N)$ is a path generating k-permutations.

Moreover, P_1 is obtained from P_{N-k+2} by applying the push-to-the-top transitions $k-1$ times, and only the bottom $n-k+1$ elements are pushed. Thus a_1 must be the top element of P_{N-k+2}. Similarly, a_i must be the top element of $P_{N-k+1+i}$, for $i = 1, 2, \ldots, k-1$. Hence $(a_1, a_2, \ldots, a_N)$ is a cycle generates k-permutations. It contains all k-permutations exactly once because a Gray code contains every k-permutation exactly once. Thus we get a universal cycle from the Gray code. □

Generalizing the observation in the above proof, we can easily check that a_{i+k-l} in the universal cycle is the l-th element in P_{i+k-l}.

On the other hand, we show that a universal cycle can be mapped to a unique Gray code. The proof first shows that there are multiple permutation paths mapped to a universal cycle, and then finds the only path that is also a cycle.

Lemma 10.12 *Given an (n,k) universal cycle $A = (a_1, a_2, \ldots, a_N)$, there is a unique corresponding Gray code $\mathbf{P} = (P_1, P_2, \ldots, P_N)$.*

Proof: We first claim that given a universal cycle A, there is a corresponding path of permutations $\mathbf{P} = (P_1, P_2, \ldots, P_N)$. For a given $1 \leq i \leq N$, let $b_1, b_2, \ldots, b_{n-k} \in [n]$ be distinct integers different from $\{a_i, \ldots, a_{i+k-1}\}$ in arbitrary order. Therefore, $P_i = (a_{i+k-1}, a_{i+k-2}, \ldots, a_i | b_1, b_2, \ldots, b_{n-k})$ is a permutation. And let $b_0 = a_i$. Since $(a_{i+1}, a_{i+2}, \ldots, a_{i+k})$ does not include duplicated elements, $a_{i+k} = b_j$ for some $0 \leq j \leq n-k$. If $j > 0$, let $P_{i+1} = (a_{i+k}, a_{i+k-1}, \ldots, a_{i+1} | b_0, \ldots, b_{j-1}, b_{j-2}, \ldots, b_{n-k})$. If $j = 0$, let $P_{i+1} = (a_{i+k}, a_{i+k-1}, \ldots, a_{i+1} | b_1, b_2, \ldots, b_{n-k})$. From P_i to P_{i+1}, we push $b_j = a_{i+k}$ to the top, which is a push-to-the-top transition and only the lower $n-k+1$ elements can be pushed. Hence the claim holds.

From above, we can see: (a) For $1 \leq l \leq k$, the l-th element in P_i is a_{i+k-l}. In particular, the k-th elements of $\mathbf{P}$ consist of the universal cycle A. (b) Given universal cycle A, the permutation path $\mathbf{P}$ is uniquely defined by the bottom elements of P_1.

Since A is a cycle, we can assume the top elements of P_1 are $(p_1, p_2, \ldots, p_k)$, and the top elements of P_N are $(p_2, p_3, \ldots, p_k, p'_k)$. If $\mathbf{P}$ is a Gray code, then from P_N to P_1 we need to push p_1 to the top.

Now we claim that there is a unique P_1 such that $\mathbf{P}$ is a Gray code. For $1 \leq i \leq N$, define q_i to be the $(k+1)$-th element of P_i. For $2 \leq i \leq N$, from P_{i-1} to P_i, if we push the k-th element to the top (do t_k), then $q_i = q_{i-1}$; otherwise, q_i is the k-th element of P_{i-1}. Since we cannot do t_k for all $1 \leq i \leq N$ and get all the k-permutations, let $i_1, i_2, \ldots$ be the indices such that we do t_j $(j > k)$ on $P_{i_1-1}, P_{i_2-1}, \ldots$ Therefore, q_{i_l} is the k-th element of Q_{i_l-1} and $q_1 = \cdots = q_{i_1}, q_{i_1+1} = \cdots = q_{i_2}, \ldots$ This implies that $(q_1, q_2, \ldots, q_N)$ are independent of the bottom elements of $\mathbf{P}$, given the k-th elements of $\mathbf{Q}$, or equivalently, given the universal cycle A.

Let $P_N = (p_2, p_3, \ldots, p_k, p'_k | r_1, r_2, \ldots, r_{n-k})$ and $R = (r_1, r_2, \ldots, r_{n-k})$ be its bottom elements. Scanning $q_N, q_{N-1}, \ldots, q_1$ one by one, if an integer already appears before, or if it belongs to $\{p_2, p_3, \ldots, p_k, p'_k\}$, then delete it. The remaining sequence is exactly R. This is because any r_i must appear in the top elements in at least one of $P_1, \ldots, P_{N-1}$, and at the transitions previous to P_N, we introduce R to the bottom elements. Moreover, given the universal cycle A, $(q_1, q_2, \ldots, q_N)$ and R are independent of the bottom elements of $\mathbf{P}$. Therefore, no matter what the bottom elements in P_1 are, P_N will be identical. Pushing p_1 to the top in P_N, we get the unique P_1 such that $\mathbf{P}$ is a Gray code. □

Combining Lemma 10.11 and 10.12, we have the following theorem.

Theorem 10.13 *There is a bijection between (n,k) universal cycles and Gray codes, where the*

concatenation of the k-th element of each permutation in the Gray code consists of the universal cycle.

By the above discussions, we know that traversing the k-partial permutations using push-to-the-top transitions is identical to generating a universal cycle. So we can apply Algorithms 10.8 and 10.9 to construct Gray codes. However, we need to define the initial full permutation in the sequence, so that it actually forms a cycle. For example, we can see that the initial permutation should be $(4,5,6|1,2,3)$ in Figure 10.5 for the specific ordering of the lower elements. In general, we have:

Theorem 10.14 *The first partial permutation should be* $(n-k+1,\ldots,n|1,\ldots,n-k)$ *so that Algorithm* 10.8 *forms a Gray code.*

Proof: Let $Q_0, Q_1, \ldots, Q_{N-1}$ be the k-partial permutations defined by Algorithm 10.8. By proof of Lemma 10.12, the first permutation should be $(n-k+1,\ldots,n|r_1,\ldots,r_k)$, where $r_1,\ldots,r_k$ are the "newly appeared" elements of $q_0, q_{N-1}, q_{N-2}, \ldots, q_1$ excluding $\{n-k+1,\ldots,n\}$. If the transition from P_{i-1} to P_i is not t_k, then q_i is the k-th element of P_{i-1}.

Suppose y is the value defined in step 1 in Algorithm 10.8, $n-k+1 \le y \le n-1$. Consider the node in first subtree T_1, $m = (n-k,\ldots,y-1,y+1,\ldots,n)$. It has parent $(n-k,\ldots,y-2,y,y+1,\ldots,n)$, Child_2 $(n-k,\ldots,y-1,y,y+2,\ldots,n)$, and Child_3 $(y,n-k+1,\ldots,y-1,y+1,\ldots,n)$. We know that Child_3 leads to the other subtrees $T_2,\ldots,T_{(k-1)!}$. It can be seen that in the Gray code, Child_3 and the other subtrees appear before Child_2 and the rest of T_1.

Similarly, for any node $(p_1, p_2, \ldots, p_{k-n+t-1}, t, t+1, \ldots, n)$, its Child_2 appears before Child_1. (In Figure 10.5, for any node in T_1, the Gray code always visit its right branch before its top branch.)

Combining these two facts, the last $n-k$ partial permutations in the Gray code with transitions $t_i, i > k$, are $(n-k+2,\ldots,n,1), (n-k+3,\ldots,n,1,2), \ldots, (1,2,\ldots,k)$. And $(r_1,r_2,\ldots,r_{n-k})=(1,2,\ldots,k)$. □

10.6 Conclusions

In this chapter, we solved the open problem of explicit constructions for universal cycles on k-partial permutations, a sequence of length $\binom{n}{k}k!$ with each k-partial permutation as its subsequence exactly once. This problem is shown to be equivalent to construction of (n,k) Gray code on partial

permutations. And we proposed the application of partial permutations for flash memory, which allows less decoding cost while sacrificing some information capacity.

The construction can be broken down into two parts: a subtree with all $\binom{n}{k}$ combinations, and a $(k, k-1)$ universal cycle. Each transition can be determined solely by the current partial permutation. The algorithm in this chapter uses $O(k)$ time and space on average, and no extra storage space when no transition is made.

There are several open problems in this topic. For instance, is it possible to design recursions of an $(n+1, k+1)$ Gray code based on an (n, k) Gray code (or some other form of recursion)? If $n - k = 1$, the answer is yes and [RW10] provides a nice construction. Besides, we do not have efficient ways to map partial permutations to integers according to the constructed cycle, i.e., we do not know the position of a permutation in the cycle. These will be our future work directions.

Chapter 11

Error-Correcting Codes for Rank Modulation

11.1 Introduction

In flash memory, errors will occur due to charge leakage, write disturbance, and read disturbance. In this chapter, we assume that small change happens more often as an error. In rank modulation, a small increase or decrease of a cell level will count as an error if its rank is interchanged with an adjacent rank.

If we write a permutation $(p_1, p_2, \ldots, p_n)$ where p_i represents the cell index with rank i, then another permutation is distance 1 from it if two adjacent indices p_i, p_{i+1} are exchanged, for some $1 \leq i \leq n-1$. For example, in Figure 11.1 an edge connects two permutations of distance one. If an error occurs in $(1,2,3)$, then the corrupted permutation is $(2,1,3)$ or $(1,3,2)$. Take any two permutations with distance 3 we have a 1-error-correction code. One can take $(1,2,3), (3,2,1)$ as two legal codewords, for instance. If a single error occurs, one can always tell which codeword is closer to the corrupted word.

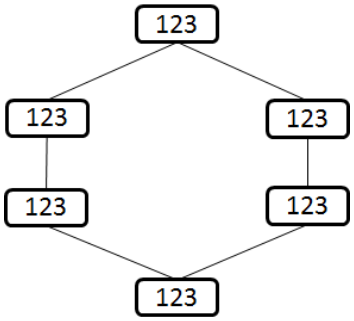

Figure 11.1: Permutations of length 3. Permutations of distance 1 are connected by an edge.

A ball of radius R centered at a word x is the collection of words with distance no more than R from x. If $d(x,y)$ represents the distance between two words x,y, and let Ω be the space of all words, a ball is

$$\mathcal{B}_R(x) = \{y \in \Omega : d(x,y) \leq R\}.$$

Define an addition $+$ operation on two words. For binary vectors, it can be viewed as bit-by-bit XOR and for permutations it is composition. Since the metrics we discuss here is translation invariant, or $d(x,y) = d(x+a,y+a)$, the ball size only depends on the radius, not on the center. Therefore, $\mathcal{B}_R(x) = \mathcal{B}_R$ for any center x. For an arbitrary block code, the sphere-packing bound is an upper bound on the number of codewords that is obtained by packing balls in a certain metric into the space of all possible words [MS77]. Let $\mathcal{C}$ be a t-error-correction code then

$$|\mathcal{C}| \leq \frac{|\Omega|}{|\mathcal{B}_t|}.$$

The reason to have t as the radius is that we need to be able to distinguish between erroneous versions of codewords after t errors. For example, in Figure 11.1 the ball size of radius 1 is $\mathcal{B}_1 = 3$ and the space size is $\Omega = 6$. Therefore, the sphere-packing bound is $|\mathcal{C}| \leq \frac{6}{3} = 2$ and we can actually achieve this bound. We say a code satisfying equality of the bound is a *perfect code*. For $t = 1$ error-correction permutation codes, the sphere-packing bound is [JSB08]:

$$|\mathcal{C}| \leq \frac{n!}{n} = (n-1)!.$$

In this chapter, we propose a class of t-error-correction codes (ECC) that reduces the permutations to ECC of alphabet size $t+1$ and Hamming distance. In particular, for rank modulation of size $n = 2^r$, we have a 1-error-correction code of size $(n-1)!/2$, that is half the sphere-packing bound.

The error model for rank modulation in flash memory was first proposed in [JSB08] from where we borrowed some results and methods. In [BM10] capacity for the full range of minimum distance $\mathcal{O}(n^{1+\epsilon})$, $0 \leq \epsilon \leq 1$, was derived. This result mainly focused on the case with a large number of errors. For constant t errors, it provides an almost explicit construction which matches the sphere-packing bound except for a constant depending on t, which is similar to the parameters of our work. The only drawback is that it is not entirely explicit. In [ZJB12] systematic error-correction codes were proposed where some subset of the cells contain only and completely the information message.

We will start with some definitions, and then show constructions of 1-error-correcting codes on rank modulation. After that, we generalize it to correcting t errors.

11.2 Definitions

In this chapter, we consider error-correcting codes for rank modulation of n elements, $\{1, 2, \ldots, n\}$. An error is defined as a transposition of adjacent elements, i.e., when $(p_1, \ldots, p_i, p_{i+1}, \ldots, p_n)$ is changed into $(p_1, \ldots, p_{i+1}, p_i, \ldots, p_n)$ for some $1 \leq i \leq n-1$. The distance between two permutations is defined as the smallest number of adjacent transpositions such that one is changed into the other. This distance measure is in fact Kendall's τ-distance [KG90]. In flash memory cells, an error corresponds to a fluctuation in the charge level of some cell, such that it interchanged rank with a cell with close charge level.

For example, suppose there are $n = 5$ cells and their analog charge values are $(1.5, 5.2, 0.3, 4.9, 7.8)$, and the induced permutation is $(5, 2, 4, 1, 3)$. Suppose the fifth cell experiences some charge leakage and drops its value from 7.8 to 5.1. As a result, the corrupted permutation becomes $(2, 5, 4, 1, 3)$ and we say there is 1 error in this permutation.

The main idea in the construction in this chapter is to reduce a permutation to a $n-1$ digit $t+1$-ary vector and t correct errors for this vector.

Define *coordinates* $(v_n, v_{n-1}, \ldots, v_2)$ of a permutation $(p_1, \ldots, p_i, p_{i+1}, \ldots, p_n)$, where v_i equals the number of inversions (i, j) such that j is on the right of i in the permutation and $1 \leq j < i$. It is easy to see that $0 \leq v_i \leq i-1$ and [JSB10] showed the mapping between coordinates and permutations is a bijection. Also define the *coordinate sum* as $s = \sum_{i=2}^{n} v_i$. Define the *coordinate parities* of a permutation as $\mathbf{u} = (u_n, u_{n-1}, \ldots, u_2) = (v_n, v_{n-1}, \ldots, v_2) \bmod 2$. In general, define the *congruent coordinates* as $\mathbf{u} = (u_n, u_{n-1}, \ldots, u_2) = (v_n, v_{n-1}, \ldots, v_2) \bmod q$ for some integer q.

For example, if the permutation is $(4, 2, 3, 1)$ then the coordinates are $(3, 1, 1)$ and its parities are $\mathbf{u} = (1, 1, 1)$.

Consider two q-ary vectors $\mathbf{a} = (a_1, a_2, \ldots, a_n), \mathbf{b} = (b_1, b_2, \ldots, b_n)$ with $a_i, b_i \in [0, q-1]$ for all i. Then the Hamming distance between the two vectors is defined as the number of indices where they differ, namely,

$$\{i \in [n] : a_i \neq b_i\}.$$

The Lee distance of two vectors $\mathbf{a}, \mathbf{b}$ is defined as

$$\sum_{i=1}^{n} \min\{|a_i - b_i|, q - |a_i - b_i|\}.$$

It indicates the difference with wrap around in the magnitude of two vectors. Since Kendall's τ distance is not very easy to work with directly, we will transform the problem of error correction to either Hamming distance or Lee distance in the coordinates.

Suppose we fix a distance measure. The minimum distance of a code is the minimum distance of all pairs of codewords. A code corrects t errors if the minimum distance is at least $2t+1$ [MS77]. We denote the parameters of a code by (n, k, d) where n is the codeword length, k is the number of information digits, or systematic digits, and d is the minimum distance. The number of redundant digits, or parity digits is $r = n - k$. Moreover, we mainly consider systematic codes, where the information are stored in k digits and can be obtained easily.

11.3 Correcting One Error

We start with an observation on the coordinates in case of an error.

Lemma 11.1 *Suppose p_i and p_{i+1} are interchanged in the permutation $(p_1, \ldots, p_i, p_{i+1}, \ldots, p_n)$, then all the coordinates stay the same except that either $v_{p_{i+1}}$ increases by 1 or v_{p_i} decreases by 1. Therefore, the sum s changes by 1 if an error occurs.*

Proof: First notice interchanging p_i and p_{i+1} will not affect the number of inversions for v_j, $j \neq i, i+1$. If $p_i < p_{i+1}$, then $v_{p_{i+1}}$ increases by 1 because we have one more inversion (p_{i+1}, p_i) for $v_{p_{i+1}}$ and v_{p_i} stays the same. Similarly, if $p_i > p_{i+1}$, then v_{p_i} decreases by 1 and $v_{p_{i+1}}$ stays the same. □

By the above lemma, if an error occurs and p_i and p_{i+1} are interchanged, then the coordinate parity bit u_i flips when the coordinate sum s decreases by 1; u_{i+1} flips when s increases by 1. Now suppose we have a way to correct 1 error for $\mathbf{u}$, and we only allow codewords as permutations whose coordinate sum satisfies $s|4$ or $(s-1)|4$. If there is an error, s is changed to $s' = s+1$ or $s' = s-1$. We can first identify the flipped position i from the 1-error-correction code. Then if $(s'-1)|4$ (or $(s'-2)|4$), then we know $s' = s+1$, $s|4$ (or $(s-1)|4$, resp.), and p_{i-1} and p_i are interchanged. Similarly, if $s'|4$ or $(s'-3)|4$, we know $s' = s-1$ and p_i and p_{i+1} are interchanged. Thus, we have the following theorem.

Theorem 11.2 *If a rank modulation code satisfy (i) coordinate parities are binary codewords of some 1-error-correcting code in Hamming distance, and (ii) coordinate sum $s|4$ or $(s-1)|4$, then this rank modulation code is a 1-error-correcting code in Kendall's τ-distance.*

For example, if permutation $P = (4,2,3,1)$ is changed to $P' = (2,4,3,1)$ then the coordinates change from $(3,1,1) \equiv (1,1,1) \mod 2$ to $(2,1,1) \equiv (0,1,1) \mod 2$. The coordinate sum changes from $s = 5$ to $s' = 4$. Suppose we use the binary repetition code $(0,0,0),(1,1,1)$ for the coordinate parities, which apparently corrects one binary error. By receiving the permutation P', it is easy to see that the first coordinate is erroneous and we need to either flip $2,4$ or $4,3$. Since $s' = 4$ we can tell that $s' = s - 1$ and the first two elements are interchanged.

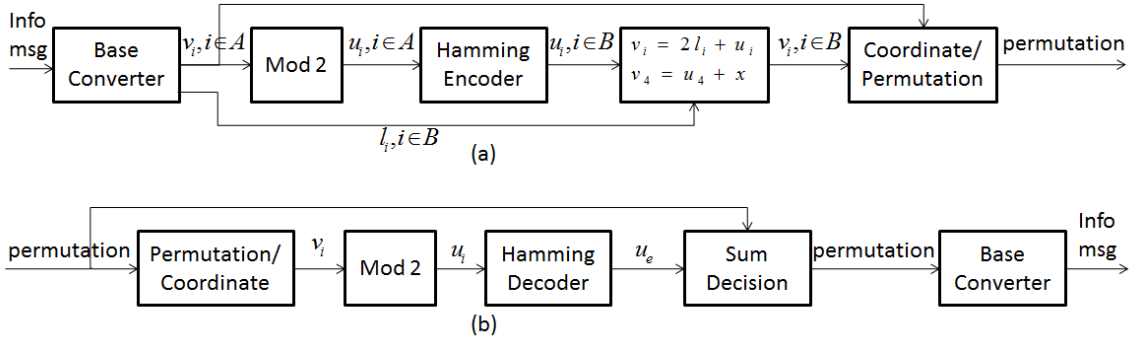

Figure 11.2: (a) Encoder and (b) decoder of a 1-error-correcting code using Hamming code and Theorem 11.2

To ensure the second property on the coordinate sum and make full use the cell levels, we have the following construction.

Construction 11.3 *We use $(2^r - 1, 2^r - r - 1)$ systematic Hamming code, $r \geq 2$, as the 1-error-correcting binary code in Theorem 11.2 (i). Let the permutation length be $n = 2^r$. And we take bits $B = \{2r, 2(r-1), \ldots, 2\}$ of u as parity check bits, and the other bits $A = \{2, \ldots, n\} \setminus B$ as information bits. Besides we use v_4 to control the sum s. Figure 11.2 shows the encoder and decoder of such a 1-error-correcting code. Define*

$$b_i = \begin{cases} i, & i \in A \\ i/2, & i \neq 4 \in B \\ 1, & i = 4 \end{cases}$$

In the encoding, each information message is first transformed into a vector $(l_n, l_{n-1}, \ldots, l_2)$ by base transformation, where $0 \leq l_i < b_i$. Let $v_i = l_i$ for $i \in A$. And then use $\{u_i\}_{i \in A}$ as information

bits of Hamming code and output $\{u_i\}_{i\in B}$*. After that, compute* $v_i = 2l_i + u_i$*, for* $i \neq 4 \in B$ *and* $v_4 = u_4 + x$ *such that* $x \in \{0,2\}$ *and sum* s *satisfies Theorem* 11.2 *(ii). At last, a permutation is generated based on the coordinates* v_i*,* $i = 2,\ldots,n$*.*

In the decoding, a permutation $(p_1,\ldots,p_n)$ *is first transformed into coordinates, and the coordinate parities* u_i*,* $i = 2,\ldots,n$ *are computed. Next Hamming decoding is carried out and error position* u_e *is located. Then, if* u_e *is not void,* p_e *is interchanged with* p_{e-1} *if* $(s-1)|4$ *or* $(s-2)|4$*, and in interchanged with* p_{e+1} *otherwise. Finally, information is extracted from the permutation.*

It is easy to see when $n = 2^r$, $\prod_{i=2}^{n} b_i = n!/2/2^r = (n-1)!/2$, which is the capacity of this rank modulation code. The sphere-packing bound of 1-error-correcting rank modulation code is $(n-1)!$ [JSB10]. Thus, we have the following corollary.

Corollary 11.4 *The 1-error-correcting rank modulation code using Hamming code and Theorem* 11.2 *is half the size of the sphere-packing bound.*

Even though the 1 error-correcting construction in [JSB10] has higher size than $(n-1)!/2$, the code proposed here has the advantage of being systematic on the coordinates and simple to encode or decode. In this code, most information is stored in the $k = 2^r - r - 1$ coordinates in A, and these coordinates correspond to fixed cells positions. So we can extract most information from comparing these cells with the others. The code in [ZJB12] is systematic on the cells but has only $(n-2)!$ codewords.

Example 11.5 *Suppose the size of the permutation is* $n = 8$ *and* $r = 3$*. Then each permutation has its corresponding coordinates* $(v_8, v_7, \ldots, v_2)$*. In Construction* 11.3*,* (v_8, v_7, v_5, v_3) *and* $\lfloor (v_6, v_2)/2 \rfloor$ *can be any vector as long as* $0 \leq v_i \leq i-1$*, and information is stored in these two vectors. However,* (u_6, u_4, u_2) *can only be a fixed vector according to* (u_8, u_7, u_6, u_3)*. In addition,* $\lfloor v_4/2 \rfloor$ *is chosen such that* $\sum_{i=2}^{8} v_i$ *is a multiple of 4 or a multiple of 4 plus 1. The size of the code is* $(8\times 7\times 5\times 3)\times(3\times 1\times 1) = 7!/2$*.*

11.4 Correcting t Errors

It is not difficult to see that we can extend the results in the previous section and correct t errors. In general, we have the following analogous result.

Theorem 11.6 *If a rank modulation code satisfy the following two conditions, then it is a t-error-correcting code in Kendall's τ-distance.*

(i) $(v_n, v_{n-1}, \ldots, v_2)$ mod $(t+1)$ has minimum Hamming distance $2t+1$.

(ii) For each $s = \sum_{i=2}^{n} v_i$, $0 \le s \le t$ mod $2(t+1)$.

Proof: Suppose the erroneous coordinates are $\mathbf{v}' = (v'_n, v'_{n-1}, \ldots, v'_2)$, and the original coordinates are $\mathbf{v} = (v_n, v_{n-1}, \ldots, v_2)$. Let error be $\mathbf{e} = (e_n, e_{n-1}, \ldots, e_2) = (v_n, v_{n-1}, \ldots, v_2) - (v'_n, v'_{n-1}, \ldots, v'_2)$. By Lemma 11.1 the Hamming distance between $\mathbf{v}' \mod (t+1)$ and $\mathbf{v} \mod (t+1)$ is no more than t, and

$$\sum_{i=2}^{n} |e_i| \le t. \tag{11.1}$$

Therefore,

$$|\sum_{i=2}^{n} e_i| \le \sum_{i=2}^{n} |e_i| \le t \tag{11.2}$$

If a rank modulation code satisfy condition (i), then we are able to determine $(e_n, e_{n-1}, \ldots, e_2)$ mod $(t+1)$ based on $(v'_n, v'_{n-1}, \ldots, v'_2) \mod (t+1)$. Now suppose there is another vector $\mathbf{f} = (f_n, f_{n-1}, \ldots, f_2)$ such that $\mathbf{e} \equiv \mathbf{f} \bmod (t+1)$. Then $\sum_{i=2}^{n} e_i \equiv \sum_{i=2}^{n} f_i \bmod (t+1)$. But by (11.2), $|\sum_{i=2}^{n} f_i| \le t$. Hence, $|\sum_{i=2}^{n} e_i - \sum_{i=2}^{n} f_i| \le 2t < 2(t+1)$. So, $\sum_{i=2}^{n} e_i - \sum_{i=2}^{n} f_i = \pm(t+1)$. Suppose $0 \le \sum_{i=2}^{n} e_i \le t$, then $-t \le \sum_{i=2}^{n} f_i < 0$. And no other error vector is equivalent to them mod $(t+1)$.

Let $s' = \sum_{i=2}^{n} v'_i$ be the coordinate sum of the erroneous permutation. By the above argument, any error pattern $\mathbf{e}$ mod $(t+1)$ will result in an original permutation with either $s^{(1)} = s' + (\sum_{i=2}^{n} e_i \mod (t+1))$ or $s^{(2)} = s' + (\sum_{i=2}^{n} e_i \mod (t+1)) - (t+1)$. Since $s^{(1)} - s^{(2)} = t+1$, only one of $s^{(1)}$ and $s^{(2)}$ falls in the category in condition (ii). Thus we can identify error correctly. □

We define the *code rate* of an error-correction code $\mathcal{C}$ as

$$\log |\mathcal{C}| / \log(n!),$$

since there are $n!$ permutations in total. Here $|\mathcal{C}|$ is the size of the codebook, or the number of different codewords. We have the following result for the constructed codes.

Construction 11.7 *Suppose we have a systematic $(n-1, k, d)$ $(t+1)$-ary code with $d \ge 2t+1$, $n \ge rt$, and $r = n-1-k$. We can use digits $B = \{t, 2t, \ldots, rt\}$ as parity, and $A = \{2, \ldots, n\} \backslash B$ as information digits. More particularly, use v_i mod $t+1$ as parity for all $i \in B$, $\lfloor \frac{v_{2t}}{t+1} \rfloor$ as sum adjustment, $\lfloor \frac{v_i}{t+1} \rfloor$ for all $i \ne 2t, i \in B$ and the other coordinates as information. The total number*

of different codewords is $\frac{n!}{2(t+1)^r}$ *and we obtain the code rate*

$$1 - \frac{\log(2(t+1)^r)}{\log(n!)}.$$

By Theorem 11.6 *we have a* t*-error-correction rank modulation code.*

Consider the sphere-packing bound of a linear $(t+1)$-ary t-error-correction code with length $n-1$,

$$(t+1)^k \leq \frac{(t+1)^{n-1}}{|\mathcal{B}(t)|},$$

where $\mathcal{B}(t)$ is the ball of radius t and $|\mathcal{B}(t)| = \sum_{i=0}^{t} \binom{n-1}{i} t^k$. Let $k*$ be the maximum value satisfying the above bound, and $r* = n-1-k*$ be the lower bound of the redundancy. Then the above construction satisfies

$$|\mathcal{C}| \leq \frac{n!}{2(t+1)^{r*}}. \tag{11.3}$$

Next we give an example of 2-error-correcting rank modulation code using $(n-1=11, k=6, d=5)$ ternary Golay code and Theorem 11.6. k is the information digits and d is the Hamming distance. If we use $(v_{12}, v_{10}, v_9, v_6, v_3)$ mod 3 as parity, $\lfloor v_6/3 \rfloor$ as sum adjustment, $\lfloor (v_{12}, v_{10}, v_9, v_6, v_3)/3 \rfloor$ and the other coordinates as information, we will get a codebook of size 887040. The sphere-packing bound is 6220800, and our code is about $1/7$ of the bound.

Similarly, we can use any linear ternary or quaternary code and set parity digits as those close to multiple of 3 or 4, and obtain rank modulation code against 2 or 3 errors. We compute codebook size for n as large as 51, and we use the table of ternary and quaternary code in [KP92]. Values of the code rates are plotted in Figure 11.3 and 11.4. In these graphs, the x-axis is the permutation length and the y-axis is the code rate. The solid line shows the sphere-packing bound of permutations, that is, codebook of size $\frac{2n!}{(n+2)(n-1)}$ for 2 errors and $\frac{6n!}{(n+1)(n^2+2n-6)}$ for 3 errors [JSB10]. And the dashed line is obtained from the $(t+1)$-ary bound in (11.3). In the ternary case, this bound is around $\frac{n!}{4n(n-2)+6}$ and $\frac{n!}{9n^3-45n^2+78n-40}$ for ternary and quaternary codes, respectively. These two bounds asymptotically only differ by constant factors 8 and 54, respectively. And if we consider the code rate this difference will vanish. The dash-dot line shows the constructed code rate based on actual $(t+1)$-ary codes. It can be observed that the code rate depends mostly on the rate of the ternary or quaternary code.

One can easily see that for constant t, the sphere-packing bound of the permutations and that of the $(t+1)$-ary codes both give upper bound of codebook size in the order of $O(\frac{n!}{n^t})$. They have

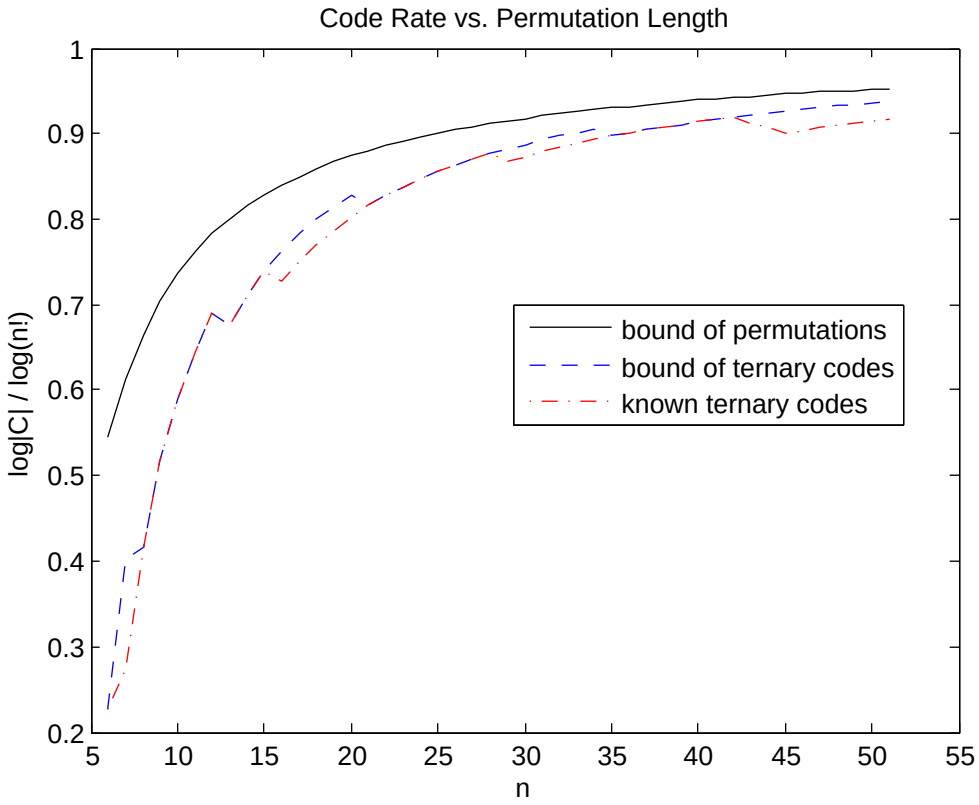

Figure 11.3: Code rates of 2-error-correction rank modulation codes based on ternary codes.

a difference by a constant factor depending on the value of t. Therefore the proposed construction has the potential of achieving asymptotically optimal code rate.

The construction above have the disadvantage of dealing with large alphabet size, which might increase the encoding/decoding complexity as well as difficulty to find efficient codes. Hence in the following we consider using binary codes.

Theorem 11.8 *Let $T \geq t$ be some integer. If a rank modulation code satisfy the following two conditions, then it is a t-error-correcting code in Kendall's τ-distance.*
(i) $(u_n, u_{n-1}, \ldots, u_2) =: (v_n, v_{n-1}, \ldots, v_2)$ *mod* $(T+1)$ *has minimum Lee distance* $2t+1$.
(ii) For each $s = \sum_{i=2}^{n} v_i$, $0 \leq s \leq t$ *mod* $2(T+1)$.

Proof: Observe that by Lemma 11.1 we know there will be at most t increases or decreases in the coordinates. Therefore the number of errors for the coordinates mod $T+1$ is no more than t in Lee distance. Similar to the proof of Theorem 11.6 we are able to determine the error vector mod $T+1$ by condition (i). Then there are at most two possible coordinate sums, which differ by $T+1$ and can be distinguished by condition (ii). □

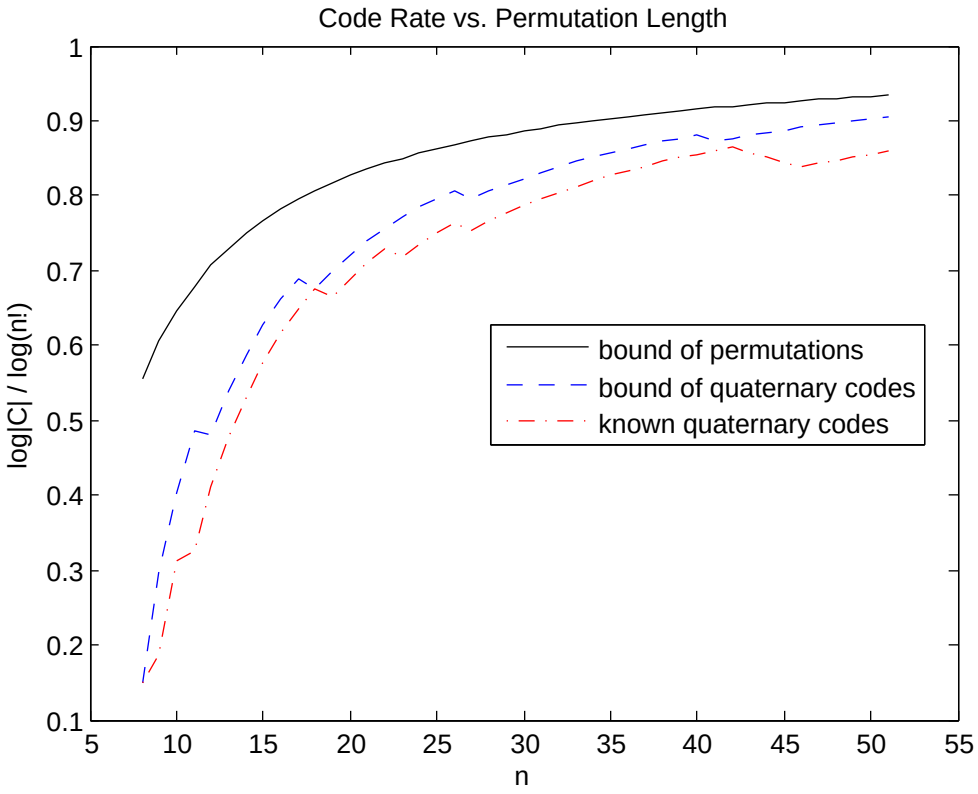

Figure 11.4: Code rates of 3-error-correction rank modulation codes based on quaternary codes.

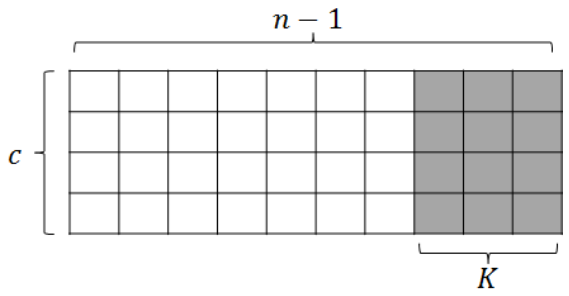

Figure 11.5: The underlined binary code and the congruent coordinates. Each entry is a bit and each column is an integer in $\mathbb{Z}_{T+1}$. The shaded bits are parity check bits, and the rest are information bits.

Let $c = \lceil \log(t+1) \rceil$, $T = 2^c - 1 \geq t$ and suppose we have a systematic (N, K, D) binary ECC with $N = (n-1)c$ and $D \geq 2t+1$. Let $R = (N-K)/c$. For simplicity, assume $n \geq (T+1)R$ and R is an integer. See Figure 11.5 for an illustration. The main idea is to use the binary code to correct t errors in Lee distance. Suppose the binary vector is $(w_1, w_2, \ldots, w_N)$. We can transform each column in this figure to an integer in $\mathbb{Z}_{T+1}$ by a binary Gray code:

$$u_{i+1} = \alpha(w_{c(i-1)+1}, w_{c(i-1)+2}, \ldots, w_{ci}), \tag{11.4}$$

where α bijectively maps a binary vector of length c to an integer in $\mathbb{Z}_{T+1}$ and the binary vector changes one bit if the integer increases or decreases by 1. Then we know the Hamming distance error in $(w_1, w_2, \ldots, w_N)$ is no more than the Lee distance error in $(u_2, u_3, \ldots, u_n)$, which is no more than t. Now we are ready to construct rank modulation codes based on binary error-correction codes.

Construction 11.9 *The encoder and decoder are shown in Figure* 11.6. *Use digits* $B = \{t, 2t, \ldots, rt\}$ *as parity, and* $A = \{2, \ldots, n\} \backslash B$ *as information digits. Define*

$$b_i = \begin{cases} i, & i \in A \\ i/(T+1), & i \neq 2(T+1) \in B \\ 1, & i = 2(T+1) \end{cases}$$

To encode, convert the message into a vector $(l_n, l_{n-2}, \ldots, l_2)$ *with* $l_i \in [0, b_i - 1], i \in [2, n]$. *Let* $v_i = l_i$ *for all* $i \in A$. *Take the congruent coordinate* $u_i, i \in A$ *and convert it to a binary vector* $(w_1, w_2, \ldots, w_K)$ *according to* (11.4). *From the binary ECC obtain parity check bits*

$(w_{K+1}, \ldots, w_N)$. *Use* (11.4) *to obtain the congruent coordinates* $u_i, i \in B$. *Then compute the coordinates in set* B *by*

$$v_i = \begin{cases} (T+1)l_i + u_i, & i \neq 2(T+1) \in B \\ x + u_i, & i = 2(T+1) \end{cases}$$

Here $x \in \{0, T+1\}$ *such that condition (ii) in Theorem* 11.8 *is satisfied. At last convert from coordinates to permutations.*

The decoder first find the coordinates of the permutation v_i, *and then the congruent coordinates* $u_i = v_i \mod (T+1)$. *Convert from* u_i *to binary Gray code* $w_i, i \in [N]$. *Using the binary Hamming decoder, the binary errors can be found. Converting them back to integers in* $\mathbb{Z}_{T+1}$ *using Gray mapping, we get congruent error* $\mathbf{u}_e = (e_n, e_{n-1}, \ldots, e_2) \mod (T+1)$. *The coordinate sum* s *helps to decide whether the error is* $\mathbf{u}_e$ *or* $\mathbf{u}_e - (T+1)$. *Adding the error back to the coordinates, we get the correct values for* v_i, *and can compute*

$$l_i = \begin{cases} v_i, & i \in A \\ \lfloor v_i/(T+1) \rfloor, & i \neq 2(T+1) \in B \\ 0, & i = 2(T+1) \end{cases}$$

The base converter at last determines the stored message.

Theorem 11.10 *Construction* 11.9 *has code rate* $1 - \log(2(T+1)^R)/\log(n!)$. *And corrects* t *errors in Kendall's* τ *distance.*

Proof: It is easy to see that the above construction has codebook size

$$\frac{n!}{2(T+1)^R}.$$

The Gray mapping guarantees that the distance in the binary Hamming code is no more than the Lee distance in $\mathbb{Z}_{T+1}$. And the coordinate $v_{2(T+1)}$ adjusts the coordinate sum properly. Thus the two conditions in Theorem 11.8 are satisfied and the code corrects t errors in Kendall's τ distance.

□

To compare the codebook size with the previous construction, assume $T = t$. Then by sphere-packing bound, $(T+1)^R = 2^{cR} = 2^{N-K} \geq \mathcal{B}(t) = \sum_{i=0}^{t} \binom{N}{i}$ is the ball size of in the binary

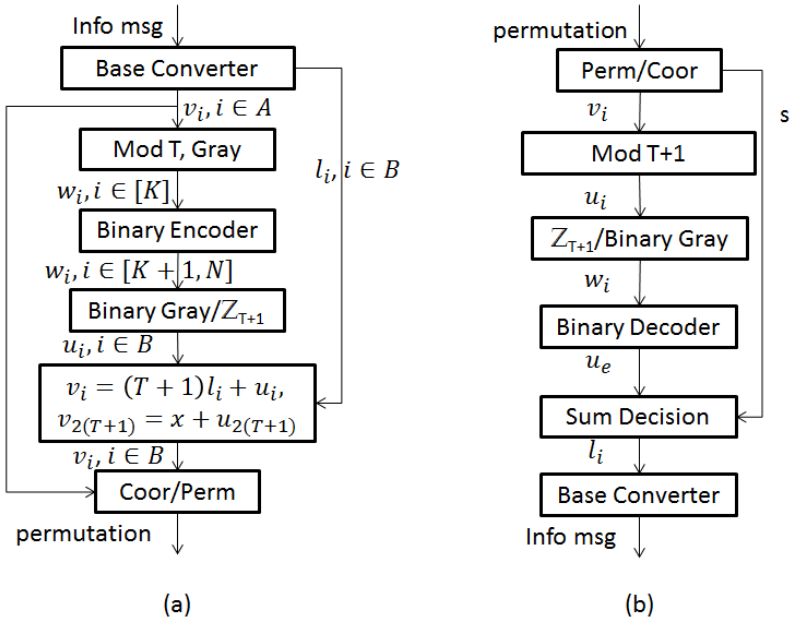

Figure 11.6: Encoder and decoder of a t-error-correction code based on binary codes.

Hamming metric. Since $N = c(n-1) = (n-1)\log(t+1)$, we have the ball size around $\binom{n\log(t+1)}{t} \approx \frac{(n\log(t+1))^t}{t!}$. On the other hand, Construction 11.7 has redundancy size $(t+1)^r \geq \sum_{i=0}^{t}\binom{n}{i}t^i$, which has ball size around $\binom{n}{t}t^t \approx \frac{(nt)^t}{t!}$. We can see that Construction 11.9 has better code rate when t is large.

For example, if $t = 3$ one can take the $(127, 106, 7)$ BCH code (see, e.g., [MS77]) and lengthen it to a $(128, 106, 7)$ code. Then $c = 2, T = 3, R = (N-K)/c = 11$ and $n = 64 > (T+1)R = 44$. The code rate is $1 - \log(2 * 4^1 1)/\log(64!) = 0.922$.

11.5 Conclusions

We have shown in this chapter a technique of embedding the Kendall's τ metric space into the q-ary Hamming or Lee space. We were able to construct t-error-correcting rank modulation codes with asymptotically optimal code rates, when t is a constant. There are other techniques such as embedding into binary Hamming space, into Lee space of different alphabet size, and so on (see, for example, [JSB10] [BM10]). Interesting open questions includes efficient code constructions given the number of errors, and error correction for multiset permutations where several cells can have identical rank.

Chapter 12

Concluding Remarks

In this thesis we mainly studied coding techniques for information storage. We have discussed erasure coding for distributed storage, and introduced the rebuilding access and bandwidth as measurement for repair cost. Different code constructions were studied and we were able to achieve optimal rebuilding in our constructions. We have also studied variations and error correction for rank modulation in flash memory. We addressed problems such as capacity, Gray code, and adjacent transposition errors. It can be seen that using information theory and other mathematical tools, we can solve some of the challenges and limitations in storage.

There are still a lot of open problems on storage. We only list a few possible directions as examples. For storage devices, such as flash memory, phase-change memory, memristor, and even the mature technology of hard disks, there are still lots of physical limitations not deeply studied. For example, phase-change memory uses amorphous and crystalline states of chalcogenide glass as cell levels, and the programming requires high current. To reduce the high power consumption and maintain small error rate is a very interesting topic from the material science and the information theory point of view. For another example, to increase the density of hard disks, the size of the magnetic region written or read each time is becoming smaller and smaller. Coding schemes as well as new disk technologies will contribute to avoiding noise and disturbances.

In data centers, the management of data and the system are of great importance besides erasure correction. To properly arrange cold and hot data, which has different access frequency, will significantly increase the lifetime and the response speed of the system. In addition, to store data according to file title or keywords, which is commonly used in today's data storage, is not necessarily the most efficient method. Storing data based on content, on the other hand, might make the information retrieval faster and even save space because of duplications. At last, storage nodes consume different amount of power while operated or idling. To better understand, predict, and design

node workloads will save energy on the storage as well as the cooling components. As data centers consume a noticeable amount of the power in the US, it is worthwhile to consider such problems.

References

[Alo99] N. Alon. Combinatorial Nullstellensatz. *Combinatorics Probability and Computing*, 8(1–2):7–29, 1999.

[BBBM95] M. Blaum, J. Brady, J. Bruck, and J. Menon. An efficient scheme for tolerating double disk failures in RAID architectures. *Computers, IEEE Transactions on*, 44(2):192–202, 1995.

[BBV96] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *Information Theory, IEEE Transactions on*, 42(2):529–542, 1996.

[BG07] J. E. Brewer and M. Gill. *Nonvolatile memory technologies with emphasis on flash*. Wiley-IEEE, 2007.

[BJB07] V. Bohossian, A. Jiang, and J. Bruck. Buffer codes for asymmetric multi-level memory. In *Information Theory Proceedings (ISIT), 2007 IEEE International Symposium on*, 2007.

[BM10] A. Barg and A. Mazumdar. Codes in permutations and error correction for rank modulation. *Information Theory, IEEE Transactions on*, 56(7):3158–3165, 2010.

[BSH05] A. Bandyopadhyay, G. Serrano, and P. Hasler. Programming analog computational memory elements to 0.2over 3.5 decades using a predictive method. In *Circuits and Systems (ISCAS), 2005 IEEE International Symposium on*, volume 3, pages 2148–2151, May 2005.

[CDG92] F. Chung, P. Diaconis, and R. Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110(1-3):43–59, 1992.

[CDH09] D. Cullina, A. G. Dimakis, and T. Ho. Searching for minimum storage regenerating codes. In *Allerton Conference on Control, Computing, and Communication*, 2009.

[CEG+04] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. *Proc. of the 3rd USENIX Symposium on File and Storage Technologies (FAST '04)*, pages 1–14, 2004.

[CHL11] V. R. Cadambe, C. Huang, and J. Li. Permutation code: optimal exact-repair of a single failed node in MDS code based distributed storage systems. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, 2011.

[CHLM11] V. R. Cadambe, C. Huang, J. Li, and S. Mehrotra. Polynomial length MDS codes with optimal repair in distributed storage systems. In *Proceedings of 45th Asilomar Conference on Signals Systems and Computing*, 2011.

[Cis12] Cisco. The zettabyte era. *Visual Networking Index (VNI)*, May 2012.

[CJ11] V. Cadambe and S. Jafar. Tensor product based subspace interference alignment for network coding applications. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, 2011.

[CJM10] V. R. Cadambe, S. A. Jafar, and H. Maleki. Minimum repair bandwidth for exact regeneration in distributed storage. In *Wireless Network Coding Conference (WiNC), 2010 IEEE*, 2010.

[CSBB10] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck. Codes for asymmetric limited-magnitude errors with application to multilevel flash memories. *Information Theory, IEEE Transactions on*, 56(4):1582–1595, April 2010.

[dBE46] N. G. de Bruijn and P. Erdos. A combinatorial problem. *Koninklijke Netherlands: Academe Van Wetenschappen*, 49:758–764, 1946.

[DGW+10] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*, 56(9):4539–4551, 2010.

[DRWS11] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, 2011.

[EGJB12] E. En Gad, A. Jiang, and J. Bruck. Trade-offs between instantaneous and total capacity in multi-cell flash memories. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, July 2012.

[EGLSB11] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck. Constant-weight gray codes for local rank modulation. *Information Theory, IEEE Transactions on*, 57(11):7431–7442, November 2011.

[FLM08] H. Finucane, Z. Liu, and M. Mitzenmacher. Designing floating codes for expected performance. In *Allerton Conference on Control, Computing, and Communication*, 2008.

[FSM12] F. Farnoud, V. Skachek, and O. Milenkovic. Rank modulation for translocation error correction. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, July 2012.

[GHSY12] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. *Information Theory, IEEE Transactions on*, 58(11):6925–6934, November 2012.

[Hal35] P. Hall. On representatives of subset. *Journal of the London Mathematical Society*, 10(1):26–30, 1935.

[HX08] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *Computers, IEEE Transactions on*, 57(7):889–901, 2008.

[Isa06] G. Isaak. Hamiltonicity of digraphs for universal cycles of permutations. *European Journal of Combinatorics*, 27:801–805, 2006.

[Jac93] B. W. Jackson. Universal cycles of k-subsets and k-permutations. *Discrete mathematics*, 117(1–3):141–150, 1993.

[JB08] A. Jiang and J. Bruck. Joint coding for flash memory storage. *Information Theory Proceedings (ISIT), 2008 IEEE International Symposium on*, 2008.

[JBB07] A. Jiang, V. Bohossian, and J. Bruck. Floating codes for joint information storage in write asymmetric memories. In *Information Theory Proceedings (ISIT), 2007 IEEE International Symposium on*, 2007.

[JMSB09] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *Information Theory, IEEE Transactions on*, 55(6):2659–2673, 2009.

[Joh09] J. Johnson. Universal cycles for permutations. *Discrete Mathematics*, 309(17):5264–5270, 2009.

[JSB08] A. Jiang, M. Schwartz, and J. Bruck. Error-correcting codes for rank modulation. *Information Theory Proceedings (ISIT), 2008 IEEE International Symposium on*, 2008.

[JSB10] A. Jiang, M. Schwartz, and J. Bruck. Correcting charge-constrained errors in the rank-modulation scheme. *Information Theory, IEEE Transactions on*, 56(5):2112–2120, 2010.

[KG90] M. Kendall and J. Gibbons. *Rank correlation methods (Fifth Edition)*. Oxford University Press, NY, 1990.

[Knu98] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley, 1998.

[Knu05] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3*. Addison-Wesley, 2005.

[KP92] F. R. Kschischang and S. Pasupathy. Some ternary and quaternary codes and associated sphere-packings. *Information Theory, IEEE Transactions on*, 38(2), 1992.

[KPT12] M. Kim, J. K. Park, and C. Twigg. Rank modulation hardware for flash memories. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, August 2012.

[MRS01] B. H. Marcus, R. M. Roth, and P. H. Siegel. *An introduction to coding for constrained systems, 5th Edition*. 2001. http://www.math.ubc.ca/ marcus/Handbook/index.html.

[MS77] F. MacWilliams and N. Sloane. *The theory of error-correcting codes*. North Holland Publishing Co., 1977.

[OD11] F. Oggier and A. Datta. Self-repairing homomorphic codes for distributed storage systems. In *INFOCOM, 2011 Proceedings IEEE*, April 2011.

[Ore11] D. Oren. Solid-state drives reshape the mobile-computing paradigm. *Mobile Dev and Design*, January 2011. SanDisk.

[PDC11a] D. S. Papailiopoulos, A. Dimakis, and V. R. Cadambe. Repair optimal erasure codes through hadamard designs. In *Allerton Conference on Control, Computing, and Communication*, 2011.

[PDC11b] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe. Repair optimal erasure codes through Hadamard designs. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, 2011.

[PPM+11] N. Papandreou, H. Pozidis, T. Mittelholzer, G. Close, M. Breitwisch, C. Lam, and E. Eleftheriou. Drift-tolerant multilevel phase-change memory. In *Memory Workshop (IMW), 2011 3rd IEEE International*, May 2011.

[RS60] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial & Applied Mathematics*, 8(2):300–304, 1960.

[RS82] R. Rivest and A. Shamir. How to reuse a write-once memory. *Information and control*, 55(1):1–19, 1982.

[RSK11] K. V. Rashmi, N. B. Shah, and P. V. Kumar. Enabling node repair in any erasure code for distributed storage. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, 2011.

[RSKR09] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Explicit construction of optimal exact regenerating codes for distributed storage. In *Allerton Conference on Control, Computing, and Communication*, 2009.

[RW10] F. Ruskey and A. Williams. An explicit universal cycle for the (n-1)-permutations of an n-set. *ACM Transactions on Algorithms (TALG)*, 6(3):45, 2010.

[SAP+13] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. *Proceedings of the VLDB Endowment*, 2013. To appear.

[SKKC03] A. G. Starling, J. B. Klerlein, J. Kier, and E. C. Carr. Cycles in the digraph P(n; k): an algorithm,. *Congressus Numerantium*, 162:129–137, 2003.

[SR10a] C. Suh and K. Ramchandran. Exact-repair MDS codes for distributed storage using interference alignment. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, 2010.

[SR10b] C. Suh and K. Ramchandran. On the existence of optimal exact-repair MDS codes for distributed storage. arXiv:1004.4663, 2010.

[SRKR10] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran. Explicit codes minimizing repair bandwidth for distributed storage. In *IEEE Information Theory Workshop (ITW)*, 2010.

[SRKV13] N. Silberstein, A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath. Optimal locally repairable codes via rank-metric codes. *CoRR*, 2013. http://arxiv.org/abs/1301.6331.

[SRVKR12] N. Shah, K. Rashmi, P. Vijay Kumar, and K. Ramchandran. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff. *Information Theory, IEEE Transactions on*, 58(3):1837–1852, 2012.

[TPD13] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis. Optimal locally repairable codes and connections to matroid theory. *CoRR*, 2013. http://arxiv.org/abs/1301.7693.

[TS10] I. Tamo and M. Schwartz. Correcting limited-magnitude errors in the rank-modulation scheme. *Information Theory, IEEE Transactions on*, 56(6):2551–2560, June 2010.

[TWB11] I. Tamo, Z. Wang, and J. Bruck. MDS array codes with optimal rebuilding. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, 2011.

[TWB12] I. Tamo, Z. Wang, and J. Bruck. Access vs. bandwidth in codes for storage. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, 2012.

[TWB13] I. Tamo, Z. Wang, and J. Bruck. Zigzag codes: MDS array codes with optimal rebuilding. *Information Theory, IEEE Transactions on*, 59(3):1597–1616, 2013.

[WD09] Y. Wu and A. G. Dimakis. Reducing repair traffic for erasure coding-based storage via interference alignment. In *Information Theory Proceedings (ISIT), 2009 IEEE International Symposium on*, 2009.

[WDR07] Y. Wu, A. G. Dimakis, and K. Ramchandran. Deterministic regenerating codes for distributed storage. In *Allerton Conference on Control, Computing, and Communication*, 2007.

[Woj12] M. Wojtasiak. There's 1500 free petabytes of cloud storage out there. October 2012. (Seagate) http://storageeffect.media.seagate.com/2012/10/storage-effect/theres-1500-free-petabytes-of-cloud-storage-out-there/.

[WTB11] Z. Wang, I. Tamo, and J. Bruck. On codes for optimal rebuilding access. In *Allerton Conference on Control, Computing, and Communication*, 2011.

[Wu09] Y. Wu. Existence and construction of capacity-achieving network codes for distributed storage. In *Information Theory Proceedings (ISIT), 2009 IEEE International Symposium on*, 2009.

[XB99] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *Information Theory, IEEE Transactions on*, 45(1):272–275, 1999.

[XBBW99] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner. Low-density MDS codes and factors of complete graphs. *Information Theory, IEEE Transactions on*, 45(6):1817–1826, 1999.

[XXLC10] L. Xiang, Y. Xu, J. C. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. *ACM SIGMETRICS Performance Evaluation Review*, 38(1):119–130, 2010.

[YSVW12] E. Yaakobi, P. Siegel, A. Vardy, and J. Wolf. Multiple error-correcting WOM-Codes. *Information Theory, IEEE Transactions on*, 58(4):2220–2230, April 2012.

[YVSW08] E. Yaakobi, A. Vardy, P. H. Siegel, and J. Wolf. Multidimensional flash codes. *Allerton Conference on Control, Computing, and Communication*, 2008.

[ZJB12] H. Zhou, A. Jiang, and J. Bruck. Systematic error-correcting codes for rank modulation. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, July 2012.