© 2020 Xiao Li

ON ERROR CORRECTING CODES FOR DISTRIBUTED STORAGE

BY

XIAO LI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mathematics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Bruce Reznick, Chair
Professor Iwan Duursma, Director of Research
Professor Olgica Milenkovic
Professor Alexander Yong

ABSTRACT

Two popular directions of error correcting codes for distributed storage are codes with additional recovery or regenerating properties.

First we have codes for additional recovery properties. Codewords in array format find applications in disk storage where columns are stored on different disks in combination with parity checks across disks that protect data against disk failures. The addition of global parities protects against sector failures on any of the disks while keeping storage overhead low. We construct sector-disk array codes that tolerate any combination of two disk failures and three sector failures with minimal overhead. This constructs for the first time codes with these parameters without relying on exhaustive search.

In the regenerating direction we have some modified layered codes in a two stage construction that gives regenerating codes with small field size. For more general parameters we define a Johnson graph code as a subspace of labelings of the vertices in a Johnson graph with the property that labelings are uniquely determined by their restriction to vertex neighborhoods specified by the parameters of the code. We give a construction and main properties for the codes. We show their role in the concatenation of layered codes to give regenerating codes for storage systems.

Focusing on the Minimum Storage regenerating (MSR) point with $d = n - 1$, we present graphical representations of codes with parameters $((n, k, d), (\alpha, \beta)) = ((qt, q(t-1), qt-1), (q^t, q^{t-1}))$ over small field size.

To my friends and family.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Error Correcting Codes for Distributed Storage

1.1.1 Some Basics of Error Correcting Codes

Error correcting codes are widely used in communication and data storage. By adding redundancy during encoding, an error correcting code allows for recovery of the original information when a certain number of errors occur. The length of an error correcting code is often fixed in storage applications. Such codes are also called block codes. Some good general references on the basics of error correcting codes are [1] and [2].

Definition 1. For a given finite field F of size q, an $[n, k]$ **block code** is an injective map

$$C : F^k \to F^n.$$

A **codeword** is a vector in the image of the map. The field size q is the alphabet size of the code. When C is a linear map, we say the code is a **linear code**.

All codes in the rest of the work are linear block codes so we will often drop the adjectives. For codes with a fixed n, codes with bigger k have more distinct codewords and carry more information. However, for error correction, having many distinct codewords is not enough. We need finer notions of being distinguishable.

Definition 2. The **minimum distance** of a set C of codewords of length n is defined as

$$d = \min_{\{x,y \in C : x \neq y\}} d(x, y)$$

where $d(x, y)$ is the Hamming distance between x and y.

A larger minimum distance allows to correct more errors. The form of error correction in this discussion would be in terms of tolerance of erasures. An **erasure** is a position in a codeword whose value is unknown. To say that a code can tolerate x number of erasures is to have any two of its codewords remain distinguishable after the same x positions are erased on each of the codewords.

For example, a code with minimum distance d can tolerate up to $d-1$ erasures. Any two codewords would remain distinct despite each missing $d-1$ entries. It is therefore desirable for an error correcting code to have many codewords that are far apart. One of the most well known results concerning the possible number of codewords at given minimal distance, length, and the alphabet size of an arbitrary block codes is the following [3].

Theorem 3 (Singleton Bound)**.** For block codes, we have the Singleton bound:

$$A_q(n,d) \leq q^{n-d+1},$$

where $A_q(n,d)$ is the maximum number of possible codewords with symbols in F of length n and minimum distance d.

For a linear code of length n and dimension k, often denoted an $[n,k]$ code, we have its minimal distance $d \leq n-k+1$ according to the Singleton bound. Codes achieving the Singleton bound are called maximum distance separable (MDS) codes.

One way to define a linear code is by giving its parity check matrix.

Definition 4. A **parity check matrix** for a linear block code C is a matrix H such that $\mathbf{v} \in \mathrm{Im}C$ if and only if $\mathbf{v}H^{\mathbf{T}} = \mathbf{0}$. A parity check is a parity check matrix consisting of one row.

An $[n,k]$ MDS code has parity check matrix of size $(n-k) \times n$ such that every $(n-k)\times(n-k)$ minor has full rank. One of the most widely used MDS codes, the Reed–Solomon code, has a parity check matrix of Vandermonde type. Reed-Solomon codes are important building blocks for many other codes. Generalized Reed-Solomon codes are obtained from Reed-Solomon codes by an invertible diagonal transformation.

Example 5. A [5,3] generalized Reed-Solomon code has the following parity

check matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{pmatrix},$$

where $\alpha_0, \alpha_1, \alpha_2, \alpha_3$, and α_4 are distinct field elements.

1.1.2 The Distributed Setting

In the distributed setting of data storage, codewords are often cut into smaller pieces and stored in different places, called nodes or disks. For example, we can have an $[N, K]$ code C which has codewords of length N and cut it into n pieces and stored on n nodes. Instead of one map $C : F^K \to F^N$, we have n maps $C_i : F^K \to F^{n_i}, 1 \leq i \leq n, \sum_i n_i = N$. We can concatenate n codewords from $C_1, C_2, \ldots, C_n$ to get a single vector of length N. Requiring such concatenated vectors of length N to generate a K dimensional subspace in F^N gives us a way to recover the original map C. Note that we can have the n_i being the same by setting some entries of the code to zero. This gives us an array code where the codewords can be viewed as arrays or matrices.

The most fundamental element that makes codes for distributed storage different is that we assume not all errors are equally likely to happen to the code C. In most applications, since the pieces are stored separately, it is more likely that whole pieces get erased instead of random erasures across multiple pieces. Codes are designed to tolerate this kind of errors so that we can recover the original message from fewer nodes.

Going further than tolerating erasures of content stored at some nodes, often called disk failures, we want our code to also tolerate some more random errors. This could simply be achieved by reducing the amount of information stored. The real challenge is to construct codes that can tolerate as many random errors as information theoretically possible without sacrificing the amount of data stored. The tightness condition on the number of additional parity checks being equal to the number of additional random erasures tolerable is from the definition of Maximal Recoverable Codes (MRC). Chapter 2 presents work to enhance access by allowing random errors on top of disk failures.

The other direction is to require codes to have repair properties. Codes are required to tolerate erasures of content stored at some nodes. In other words,

a subset of nodes should be able to provide their content and allow for the original data to be recovered. This is called the data collection requirement. In addition to data collection, we want our codes to have a repair property, where a failed node can be repaired using help provided from some other available nodes. Such repair properties come at a cost and there are two major families of such codes with different cost considerations. One is called locally recoverable codes, for which cost of repair is measured by the number of helper nodes contacted. Locally recoverable codes were introduced in [6]. The other is regenerating codes, for which cost of repair is measured by the amount of data required from the helper nodes. Regenerating codes were introduced in [9]. We see that the more information we store on each node, the more helper information we would need for regeneration. The trade-off between the two is the key constraint for regenerating codes. Regenerating codes are the topic for Chapters 3 and 4.

1.2 Enhancing Data Access: Sector Disk Codes

Suppose we have an array code C of length $r \times n$. The code words can be viewed as $r \times n$ arrays, where each of the n disks store r entries. We can require C to tolerate m disk failures, where a disk failure is having the r erasures on one of the n disks.

For a 3×5 array code, the left is not an example of disk failures, where the right is an example of two disk failures. We use $*$ to denote an erasure.

$*$				$*$
	$*$		$*$	
		$*$		

$*$	$*$			
$*$	$*$			
$*$	$*$			

Example 6. A 3×5 array code tolerating two disk failures has the following parity check matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & & & & & \\ \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & & & & & \\ & & & & & * & & & & \\ & & & & & & 1 & 1 & 1 & 1 & 1 \\ & & & & & & \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{pmatrix}.$$

There are five disks each having three symbols. The parity check matrix has three blocks. Each block has five columns corresponding to the five disks. Each block is a $[5, 3]$ MDS code so the code can tolerate two failed disks.

In addition, we can require C to tolerate an additional s random erasures. The following is an example of a disk failure and two additional random erasures.

*		*		
*				
*				*

Definition 7. An $r \times n$ array code is an (m, s) **sector disk (SD) code** if it tolerates m disk failures and an additional s random erasures.

Sector-disk codes were introduced in [4] and provide disk storage with protection against both device failures and sector failures without the large storage overhead of existing solutions, including RAID 6. Important results and detailed information about sector-disk codes and related families of codes can be found in [4], [5], [7]. Most examples of sector-disk codes were found though exhaustive search [4]. The minimal overhead requirement for a sector-disk code is a strong condition that has been hard to realize through theoretical constructions. The first constructions that tolerate two sector failures appear in [12], and a general construction for sector-disk codes tolerating two sector failures in [13], [14].

Theorem 8 (Blaum, Plank, Schwartz, Yaakobi, '14)**.** For certain fields F, the following parity check matrix defines a code that tolerates $m = 2$ disk failures and $s = 2$ additional erasures.

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & & & & & & & \\
\alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & & & & & & & \\
 & & & & & * & & & & & & \\
 & & & & & & 1 & 1 & 1 & 1 & 1 & \\
 & & & & & & \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \\
1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \dots & 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \\
1 & \alpha^{14} & \alpha^{13} & \alpha^{12} & \alpha^{11} & \dots & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha &
\end{pmatrix}.$$

A construction that tolerates three sector failures was left as an open problem. In Chapter 2 we provide a construction and include as examples cyclic array codes of size 17×5 and 65×7 over fields of bit size 8 and 12.

Similar to previous works, we construct a parity check matrix of dimension $(rm+s)\times rn$, and show that all the necessary minors have the MDS property. We take advantage of the symmetry of the parity check matrix of the array code and change perspective to view the checks as evaluation of polynomial checks. This is continued in Chapter 2.

1.3 Repairing Failed Nodes: Regenerating Codes

Suppose that we have n nodes, called disks. We can require that any k out of the n disks are enough for data collection, i.e. content of any k out of n disks are sufficient for recovering the entire original message. This is sometimes called the k out of n property. In addition, we can require that a failed disk is recoverable with the help of d out of the $n-1$ remaining nodes. This brings the following definition [9].

Definition 9. A code with n disks each storing α symbols each is an **exact regenerating code** of parameters (n, k, d, α, β) if it satisfies the following conditions:

1. Data collection: Any k out of the n disks are sufficient to reconstruct the stored information.

2. Repair: A failed disk needs β symbols each from any d of the remaining $n-1$ disks for exact repair.

The total amount of information stored is denoted by M.

Here "exact" means that the precise content is recovered, as opposed to some modified symbols that would allow the system to maintain its properties. The latter is called functional repair and is significantly easier to achieve.

One of the earliest examples of exact regenerating codes are the layered codes [16].

Example 10. For $(n, k, d) = (4, 3, 3), \alpha = 3, \beta = 2$, we have an exact repair code given by the following relations:

x_1	x_2	x_3
y_1	y_2	y_3
z_1	z_2	z_3
t_1	t_2	t_3

$$x_1 = y_2 + z_3$$
$$x_2 = y_3 + t_1$$
$$x_3 = z_1 + t_2$$
$$y_1 = z_2 + t_3,$$

where the x, y, z and t variables are stored in the four disks, respectively. It is easy to verify that the code has the required data collection and repair properties.

We can rearrange the array as follows:

	123	124	134	234
D_1	x_1	x_2	x_3	
D_2	y_2	y_3		y_1
D_3	z_3		z_1	z_2
D_4		t_1	t_2	t_3

.

The checks would be a parity check relation on each column. This latter formulation is more convenient for the general definition of a layered code.

Definition 11. A **layered code** of parameters $(n, n-1, n-1)$ and $2 \leq v \leq n$ is a code of length $v\binom{n}{v}$ with relations defined by parity checks according to layers. A layer L is a subset of $\{1, 2, \ldots, n\}$ of size v. In an $n \times \binom{n}{v}$ array we can label the rows by $1 \leq i \leq n$ and the columns by the layers. A layer L contains a symbol at the i^{th} row if and only if $i \in L$. For each layer L we have a parity check relation that the symbols in the column sum to zero.

Example 12. For $(n, k, d) = (5, 4, 4), v = 3$, we get a layered code with parameters $(\alpha, \beta, M) = (6, 3, 20)$, constructed as follows:

	123	124	125	134	135	145	234	235	245	345
D_1	A_1	B_1	C_1	E_1	X_1	Y_1				
D_2	A_2	B_2	C_2				Z_2	U_2	V_2	
D_3	A_3			E_3	X_3		Z_3	U_3		W_3
D_4		B_4		E_4		Y_4	Z_4		V_4	W_4
D_5			C_5		X_5	Y_5		U_5	V_5	W_5

.

We have 10 groups of 3 symbols each (labeled by the same alphabet), and each group sums to zero to give ten checks. Each disk stores six symbols as shown above. We see that any four disks will allow complete recovery of the entire file. Also any $d = 4$ disk providing $\beta = 3$ symbols each will allow for recovery of a failed disk.

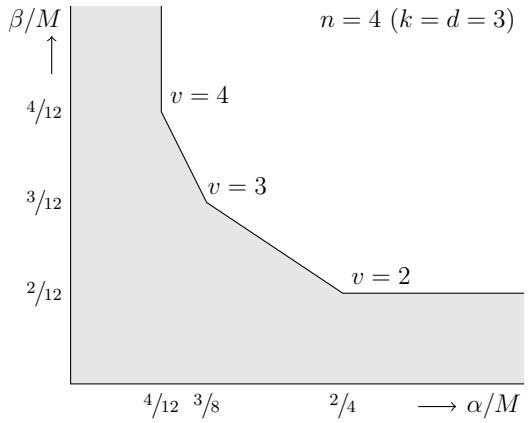


Figure 1.1: Storage cost α/M vs Disk repair cost β/M for $n = 4$ disks using local checks of length $v = 2, 3, 4$.

One central problem in the study of exact regenerating codes is the trade-off between storage overhead versus regeneration cost. Storage overhead, or storage cost, can be measured by α/M and repair cost can be measured by β/M. We want both to be low for applications. There are two extreme points for the trade-off, called the minimum storage regenerating (MSR) point, and the minimum bandwidth regenerating (MBR) point.

Definition 13. An exact repair regenerating code is **minimum storage regenerating** (MSR) if it has $M = k\alpha$ and $\beta = \alpha/((d+1)-k)$. An exact repair regenerating code is **minimum bandwidth regenerating** (MBR) if it has $\alpha = d\beta$ and $M = \frac{k(2d+1-k)}{2}\beta$.

As α and β both scale with the file size M, we can visualize the trade-off by plotting β/M against α/M. A key feature of layered codes is their optimality with respect to this trade-off, see Figure 1.1. In particular they achieve the MSR point for $v = n$, and the MBR point for $v = 2$.

One shortcoming of layered codes is that they do not provide great access. As k is restricted to be $n - 1$ for layered codes, all but one disk have to be contacted for downloading data. In order to have better access, we want to have a lower k so that we can download the data from fewer disks. This led to many code constructions. Some of the constructions require further dividing the original message symbols and this process is called sub-packetization. A high sub-packetization level is not desirable for applications.

One way to achieve a lower k is to modify layered codes. This leads to the

Improved layered codes [17]. The codes work for $k \leq d$ but suffer from higher storage overhead. Similar and more intuitive constructions appear in the earlier part of Chapter 3. The codes constructed there have the advantage of using binary alphabets or other fields of small size. The later part of Chapter 3 presents a key result in the construction and properties of Johnson graph codes.

Definition 14. A **Johnson graph code** $JGC(n, v, k, r)$ on the vertices of the Johnson graph $J(n, v)$ is a code of length $N = \binom{n}{v}$ and dimension $K = |B_r(A)|$ such that, for any k-subset A of $\{0, 1, \ldots, n-1\}$, $B_r(A)$ forms an information set.

Here $B_r(A)$ is the distance r-neighborhood defined by the set A. This comes up naturally as we access a k-subset of the n disks. One of the applications of Johnson graph codes is to give a simpler description of Improved layered codes. Details are in the last section of Chapter 3.

The product matrix constructions [22] work for parameters satisfying ($n = d + 1, k, d \geq 2k - 2$). They have low sub-packetization levels but only exist for select parameters.

Determinant codes, Cascade codes [20], [21] work for general (n, k, d), but suffer from higher sub-packetization levels. The last part of Chapter 3 gives a friendly description for cascade codes with $(n, k, n-1)$ as an application of Johnson graph codes.

The explicit construction of MDS array codes [23] and the related Coupled-layer codes constructions [18] work over small field size and have sub-packetization levels lower than the Cascade codes at the MSR point. Chapter 4 explores code constructions using graphical representations as factor graphs and recover the parameters of these leading constructions at the MSR point.

CHAPTER 2

CODES FOR RECOVERY: SECTOR-DISK CODES TOLERATING THREE ERASURES

2.1 Array codes from pairs of MDS codes

In this section we present two special types of array codes. The first array code appears in the context of secret sharing when two secret sharing schemes are combined into a composite secret sharing scheme [15]. The second array code defines a disjoint sector-disk code [14]. The codes are easy to define and have attractive properties but in general do not meet the full requirements of sector-disk codes (and thus not of partially MDS codes). Sector-disk codes will be discussed in the next sections.

Both array codes are of size $r \times n$ with rows $c_1, c_2, \ldots, c_r \in C$ for a fixed MDS code C of length n. For a code C with redundancy m, each individual row can tolerate m erasures. Without further parities the array code can recover from rm erasures provided they are equally divided over the rows. For the array codes to be defined below we ask that the column sum, respectively the row sum, satisfies an additional s parity checks. For each of the two cases we describe the sets of $rm + s$ erasures that the array code can tolerate.

Definition 15. Given integers n, m, r and s, such that $m \leq n$ and $s \leq r$, let C and C' be two MDS codes of type $[n, n-m]$ and $[r, r-s]$, respectively, such that the subcode $C_0 \subset C$ of words that are orthogonal to the all-one vector is MDS of type $[n, n-m-1]$. Define the $r \times n$ array code $C' \circ C$ as the set of those arrays for which each of the r rows belongs to C and the sum of the n columns belongs to C'.

An $[r, r-s]$ code C' is an MDS code if and only if every subset of $r-s$ out of r codeword symbols determines the codeword uniquely. The composite code $C' \circ C$ satisfies the following composite threshold.

Lemma 16. The column sum of a codeword in $C' \circ C$ can be determined

from exactly those subsets of codeword symbols that contain at least $n - m$ out of n symbols for at least $r - s$ out of r rows in the codeword array.

Proof. Straightforward application of the MDS thresholds for the codes C and C' to the code $C' \circ C$. □

Proposition 17. The array code with respect to $C' \circ C$ has redundancy $rm + s$. It tolerates m erasures in each row and s further erasures in distinct rows.

Proof. It is clear that $C' \circ C$ has redundancy at most $rm + s$. To prove that the redundancy is at least $rm + s$ it suffices to prove that the code can repair at least one erasure pattern of size $rm + s$. Thus it suffices to prove the second claim of the proposition. To repair m erasures in each row and s further erasures in distinct rows, first repair the $r - s$ rows with at most m erasures using that each row is a codeword in C. The $r - s$ repaired rows give $r - s$ symbols for the column sum which can then be completed uniquely to a codeword in C'. The completed column sum gives an additional parity with respect to the all-one vector for the remaining s rows. With this extra parity the remaining rows can be repaired as codewords in C_0 from $n - m - 1$ codeword symbols. □

Example 18. For codes C and C' of type $[5, 3]$ and type $[3, 1]$, respectively, the composite code $C' \circ C$ can recover the column sum for the 3×5 array below from the symbols c' and then the full array from the additional symbols c. Erasures tolerated by the code are marked as $\times$.

$$\begin{pmatrix} \times & \times & \times & c & c \\ \times & \times & \times & c & c \\ \times & \times & c' & c' & c' \end{pmatrix}$$

Remark 19. The condition for the subcode $C_0 \subset C$ in Definition 15 guarantees that the checks for $C' \circ C$ are independent and that the code has redundancy $rm + s$. The choice for the all-one word in the definition is for convenience and is not essential. The definition can be used with a different linear combination of columns as long as the coefficients form a word at distance $n - m$ from the dual code $C^{\perp}$. Such words may not exist for certain MDS codes (e.g. the hexacode) but do exist for all Reed-Solomon codes.

Definition 20. Given integers n, m, r and s, such that $m + s \leq n$, let $C_0 \subset C$ be a pair of nested MDS codes of type $[n, n-m-s]$ and $[n, n-m]$, respectively. Define the $r \times n$ array code with respect to $C_0 \subset C$ as the set of those arrays for which each of the r rows belongs to C and the sum of the r rows belongs to C_0.

Proposition 21. The array code with respect to $C_0 \subset C$ has redundancy $rm + s$. It tolerates m erasures in each of the r rows. Moreover, in case of m column erasures, it tolerates s further erasures in distinct columns.

Proof. The redundancy is clear. Also, the code C tolerates m erasures, which protects each of the rows against m erasures. For the case of m column erasures, the row sum is protected by C_0 against $m + s$ erasures. Thus, we can recover first the row sum, then the s columns with single erasures, and finally the m erased columns. □

The ability to correct m column erasures and s additional erasures in distinct columns means that the code is a distinct-sector-disk code (DSD code) as defined in [14]. The construction of DSD codes in [14] makes use of Cauchy matrices. The use of Cauchy matrices guarantees that the subcode $C_0 \subset C$ is again an MDS code and the construction in [14] is a special case of the above construction. The use of Cauchy matrices requires a field of size at least $n + m + s$ whereas the construction in Definition 20 can use Reed-Solomon codes of length n.

Example 22. For MDS codes $C_0 \subset C$ of type $[5, 1]$ and $[5, 3]$, respectively, the 3×5 array code with respect to $C_0 \subset C$ can recover the row sum for the array below from the symbols c_0 and then the full array from the additional symbols c. Erasures tolerated by the code are marked as $\times$.

$$\begin{pmatrix} \times & \times & \times & \times & c_0 \\ \times & \times & c & c & c_0 \\ \times & \times & c & c & c_0 \end{pmatrix}$$

Example 18 and Example 22 give two different codes that can each repair a given pattern of two column erasures and two additional erasures. Neither of the codes can repair both patterns. Sector-disk codes, to be discussed in Section 2.3, are able to correct any combination of two column erasures and any two additional erasures.

2.2 Array codes from pairs of RS-codes

The parity checks for the array codes in the previous section have a convenient polynomial description if we apply the construction with RS-codes. Label the columns of an $r \times n$ array by $A = \{\alpha_1, \ldots, \alpha_n\} \subset K$ and the rows by $B = \{\beta_1, \ldots, \beta_r\} \subset K$. Data in each row is protected by a $[n, n-m]$ code C with parity matrix

$$H = \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n \end{pmatrix}.$$

For RS-codes we set $\mathbf{h}_i = (\alpha_i, \ldots, \alpha_i^m)^T$. For the construction in Definition 15, the column sum is protected by a $[r, r-s]$ code C' with parity checks

$$H' = \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_r \end{pmatrix}.$$

For RS-codes we set $\mathbf{b}_j = (1, \beta_j, \ldots, \beta_j^{s-1})^T$. For the construction in Definition 20, the row sum is protected by a $[n, n-m-s]$ subcode $C_0 \subset C$, defined as subcode by the extra parity checks

$$H_0 = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{pmatrix}.$$

For RS-codes we set $\mathbf{a}_i = (\alpha_i^{m+1}, \ldots, \alpha_i^{m+s})^T$. The parity checks for the $r \times n$ array codes in Definition 15 and Definition 20 depend on H, H' and H_0. They are written out for the row concatenated version of an array code in Equation (2.1).

$$\left[\begin{array}{cccc|cccc|c|cccc} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n & & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ & & & & & & & & \ddots & & & & \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & & \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc|c|cccc} \mathbf{b}_1 & \mathbf{b}_1 & \cdots & \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_2 & \cdots & \mathbf{b}_2 & \cdots & \mathbf{b}_r & \mathbf{b}_r & \cdots & \mathbf{b}_r \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc|c|cccc} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n & \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n & \cdots & \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{array}\right] \tag{2.1}$$

Basic parity checks for the row concatenated version of an $r \times n$ array code.
– Rows of the array are $\perp \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n \end{pmatrix}$
– Column sum for the array is $\perp \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \end{pmatrix}$
– Row sum for the array is $\perp \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{pmatrix}$

The row space for the row parity checks in (2.1) is the linear span of the evaluation in $A \times B = (\alpha_1, \beta_1), (\alpha_1, \beta_2), \ldots, (\alpha_n, \beta_r)$ of the following rm monomials.

$$x^i y^j, \quad 1 \leq i \leq m, 0 \leq j < r. \tag{2.2}$$

The column sum parity checks in (2.1) are the evaluation of monomials

$$y^j, \quad 0 \leq j < s. \tag{2.3}$$

The row sum parity checks in (2.1) are the evaluation of monomials

$$x^{m+i}, \quad 1 \leq i \leq s. \tag{2.4}$$

Example 23. For codes C and C' of type $[5, 3]$ and type $[3, 1]$, respectively, the array code $C' \circ C$ admits as parity checks the evalutations in $A \times B$ of the monomials

$$\begin{pmatrix} 1 & x & x^2 & \cdot & \cdot \\ y & xy & x^2y & \cdot & \cdot \\ \cdot & xy^2 & x^2y^2 & \cdot & \cdot \end{pmatrix}$$

The six monomials in rows 1 and 2 provide checks to repair $s = 2$ erasures (e.g. in Example 18, the erasure in (α_3, β_2) is repaired using the evaulation of $(x-\alpha_1)(x-\alpha_2)(y-\beta_1)$ which is zero in the other erasures). The six monomials in columns 2 and 3 then provide checks to repair the $m = 2$ column erasures (e.g. in Example 18, the erasure in (α_2, β_3) uses the evaluation of $x(x - \alpha_1)(y - \beta_1)(y - \beta_2)$).

Example 24. A 3×5 array code with respect to $C_0 \subset C$, for codes C_0 and C of type $[5, 2]$ and type $[5, 4]$, respectively, admits as parity checks the evalutations in $A \times B$ of the monomials

$$\begin{pmatrix} \cdot & x & x^2 & x^3 & x^4 \\ \cdot & xy & x^2y & \cdot & \cdot \\ \cdot & xy^2 & x^2y^2 & \cdot & \cdot \end{pmatrix}$$

The four monomials in row 1 provide checks to repair $s = 2$ erasures (e.g. in Example 22, the erasure in (α_4, β_1) is repaired using the evaulation of $x(x - \alpha_1)(x - \alpha_2)(x - \alpha_3)$ which is zero in the other erasures). The six monomials in columns 2 and 3 then provide checks to repair the $m = 2$ column erasures (as in the previous example).

The arguments used in the last two examples generalize easily to give alternative proofs for Proposition 17 and Proposition 21, respectively, for array codes constructed with RS-codes. We will use similar arguments in the next section to prove erasure tolerance for sector-disk codes and partially MDS codes.

2.3 Array codes with $s = 2$

The array codes in the previous section protect either the column sum or the row sum with s parities. Protecting the column sum allows the code to repair s additional erasures in distinct rows. Protecting the row sum allows the code to repair s additional erasures in distinct columns. It remains to construct a code that is able to protect s additional erasures of general type. Two such codes are defined in [5]. Sector-disk codes (SD codes) are defined as array codes able to correct a combination of m column erasures and any s additional erasures, with minimal redundancy $rm + s$. Partially MDS codes are array codes able to correct any combination of m erasures per row and s additional erasures, with minimal redundancy $rm + s$. Construction A and its modification (both in [14]) give constructions for both types of codes when $s = 2$. We describe the construction in the monomial form of the previous section and include proofs for the error tolerance.

The solution to correct any $s = 2$ additional erasures uses a combination of the two previous constructions. Instead of protecting either the column sum or the row sum by two parities we protect each of them by a single parity. The resulting parity matrix is again of the form given by Equation (2.1). The first block is the same as before and is included to protect each row against m erasures. The row span of this block of checks is given by the evaluation of the rm monomials in (2.2). Whereas the previous constructions add either one or the other block of s checks in (2.1), for the $s = 2$ solution we add a single check of each type, a check with $\mathbf{b}_j = \beta_j$ and a check with $\mathbf{a}_i = \alpha_i^{m+1}$. The first check is obtained by evaluating in $A \times B$ the monomial y chosen from (2.3), and the second check by evaluating in $A \times B$ the monomial x^{m+1} chosen from (2.4).

Example 25. A 3×5 array code with $C_0 \subset C$ of type $[5, 3]$ and type $[5, 4]$, and column sum protected by C' of type $[3, 2]$ admits as parity checks the evaluations in $A \times B$ of the monomials

$$\begin{pmatrix} \cdot & x & x^2 & x^3 & \cdot \\ y & xy & x^2y & \cdot & \cdot \\ \cdot & xy^2 & x^2y^2 & \cdot & \cdot \end{pmatrix}$$

Assume that $m = 2$ column erasures occurred in columns c_1 and c_2 and that $s = 2$ additional erasures occurred outside these columns in the positions (a_1, b_1) and (a_2, b_2). Let $\sigma(x) = (x - c_1)(x - c_2)$. We use the monomials in row 1, resp. row 2, to form the two polynomials

$$x(x - c_1)(x - c_2) = x\sigma(x)$$
$$y(x - c_1)(x - c_2) = y\sigma(x)$$

The evaluations of these polynomials provide us with two checks that are both zero in the two erased columns. The checks are independent in the two additional erasures, and together can correct both erasures, if and only if

$$\begin{pmatrix} a_1\sigma(a_1) & a_2\sigma(a_2) \\ b_1\sigma(a_1) & b_2\sigma(a_2) \end{pmatrix}$$

is invertible. Thus, it suffices to choose A and B such that any two distinct pairs (a_1, b_1), $(a_2, b_2) \in A \times B$ are linearly independent.

Proposition 26. (Sector-disk codes with $s = 2$) Define an $r \times n$ array code with checks obtained by evaluating the monomials $\{x^i y^j : 1 \leq i \leq m, 0 \leq j < r\} \cup \{x^{m+1}, y\}$ in the points $A \times B \subset K \times K$. If the map $A \times B \to K$ with $(a, b) \to ab$ is injective then the code is an $(m, 2)$ SD code.

Proof. Let $C^* \subset A$ be a subset of size m containing the locations of the column erasures. Let $\sigma(x) = \prod_{c \in C^*}(x - c)$. Then $x\sigma(x)$ and $y\sigma(x)$ are in the linear span of the set of monomials and the evaluation of $x\sigma(x)$ and $y\sigma(x)$ in $A \times B$ defines two checks for the array code. Both checks are zero in the erased columns C^*. Together the checks are able to correct erasures in (a_1, b_1) and (a_2, b_2), both outside the columns C^*, if and only if

$$\det \begin{pmatrix} a_1\sigma(a_1) & a_2\sigma(a_2) \\ b_1\sigma(a_1) & b_2\sigma(a_2) \end{pmatrix} \neq 0$$

if and only if $a_1 b_2 \neq a_2 b_1$. □

Proposition 27. (Partially MDS codes with $s = 2$) Define an $r \times n$ array code with checks obtained by evaluating the monomials $\{x^i y^j : 1 \leq i \leq m, 0 \leq j < r\} \cup \{x^{m+1}, y\}$ in the points $A \times B \subset K \times K$. If the products $(\prod_{a \in A^*} a)b$ are distinct for all $m + 1$-subsets A^* of A and all elements $b \in B$ then the code is an $(m, 2)$ PMDS code.

Proof. For the special case that one row contains $m + 2$ erasures the proof reduces to the proof given for sector-disk codes. Thus, after repairing erasures in rows with at most m erasures, we may assume that the remaining erasures occur in rows b_1 and b_2 and that each row contains $m + 1$ erasures. Let

$$e_1(y) = (y - b_2)/(b_1 - b_2)$$
$$e_2(y) = (y - b_1)/(b_2 - b_1)$$

Choose erasures (a_1, b_1) in row b_1 and (a_2, b_2) in row b_2, and let $C_1^* \subset A$ and $C_2^* \subset A$ contain the locations of the m other erasures in row 1 and row 2, respectively. For $i = 1, 2$, let $\sigma_i(x) = \prod_{c \in C_i^*}(x - c)$ and let

$$p_i(x) = (x^m - \sigma_i(x))x$$
$$q_i(x) = (\sigma_i(0) - \sigma_i(x))/\sigma_1(0)$$

The polynomials p_i and q_i are in the span of the monomials $x, x^2, \ldots, x^m$ and satisfy $p_i(c) = c^{m+1}$ and $q_i(c) = 1$ for $c \in C_i^*$. The polynomials

$$x^{m+1} - p_1(x)e_1(y) - p_2(x)e_2(y),$$
$$y - q_1(x)e_1(y)b_1 - q_2(x)e_2(y)b_2,$$

define two checks that are both zero in the positions C_1^* of row b_1 and the positions C_2^* of row b_2. In the positions (a_1, b_1) and (a_2, b_2) the checks take the values

$$\begin{pmatrix} a_1\sigma_1(a_1) & a_2\sigma_2(a_2) \\ b_1\sigma(a_1)/\sigma_1(0) & b_2\sigma(a_2)/\sigma_2(0) \end{pmatrix}$$

The determinant is nonzero if and only $(\prod_{c \in C_1^*} c)a_1b_2$ and $(\prod_{c \in C_2^*} c)a_2b_1$ are distinct. □

Example 28. Let $r = 3, n = 5$ and let α be a primitive element for $GF(16)$. The condition does not hold with $m = 1$ for $A = \{1, \alpha, \alpha^2, \alpha^3, \alpha^4\}$, $B = \{1, \alpha^5, \alpha^{10}\}$. The construction in [14] requires using a larger field ([id., Example 6]). However, the condition does hold with $m = 1$ for $A = \{1, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}\}$. Moreover, with this choice the code is cyclic.

2.4 Array codes with $s = 3$

Modifying our code construction for $s = 2$, we construct codes with $s = 3$ by adding a third global parity. The pattern of monomials $x^i y^j$ used in Example 25 is covered by $m = 2$ columns and $s = 2$ rows. We replace the pattern with an equivalent pattern and then add a monomial halfway between the two columns and halfway between the two rows.

Example 29. Define a 5×7 array code with parity checks the evaluations in $A \times B$ of the monomials

$$\begin{pmatrix} \cdot & \cdot & x^2 & \cdot & x^4 & \cdot & x^6 \\ \cdot & \cdot & x^2y & x^3y & x^4y & \cdot & \cdot \\ y^2 & \cdot & x^2y^2 & \cdot & x^4y^2 & \cdot & \cdot \\ \cdot & \cdot & x^2y^3 & \cdot & x^4y^3 & \cdot & \cdot \\ \cdot & \cdot & x^2y^4 & \cdot & x^4y^4 & \cdot & \cdot \end{pmatrix}$$

For $A = \{a : a^7 = 1\}$ and $B = \{b : b^5 = 1\}$, the code is a cyclic sector-disk code with $m = 2$ and $s = 3$.

The format in the example uses that the map $x \to x^2$ is injective on A. We make use of the same format in the next theorem and assume that A satisfies this assumption.

Theorem 30. (Sector-disk codes with $m = 2$, $s = 3$) Define an $r \times n$ array code with checks obtained by evaluating the monomials $\{x^2y^j, x^4y^j : 0 \leq j < r\} \cup \{y^2, x^3y, x^6\}$ in the points $A \times B \subset K \times K$. For $c_1, c_2 \in A$, let $\sigma(x) = (x - c_1)\,(x - c_2)$. If the matrix

$$\begin{pmatrix} 1 & 1 & 1 \\ b_1/\sigma(-a_1) & b_2/\sigma(-a_2) & b_3/\sigma(-a_3) \\ b_1^2/a_1^2 & b_2^2/a_2^2 & b_3^2/a_3^2 \end{pmatrix} \tag{2.5}$$

is invertible, for all $c_1, c_2 \in A$, and for all (a_1, b_1), (a_2, b_2), $(a_3, b_3) \in A\backslash\{c_1, c_2\} \times B$ then the code is a $(2, 3)$ SD code.

Proof. Assume that $m = 2$ column erasures occured in columns c_1 and c_2, and that $s = 3$ additional erasures occurred outside these columns in the positions (a_1, b_1), (a_2, b_2) and (a_3, b_3). Let $\sigma(x) = (x - c_1)(x - c_2)$. We use the monomials with $j = 0, 1, 2$ (as in rows 1, 2 and 3 in Example 29) to form the three polynomials

$$\begin{aligned} x^2(x^2 - c_1^2)(x^2 - c_2^2) &= x^2\sigma(x)\sigma(-x) \\ x^2y(x - c_1)(x - c_2) &= yx^2\sigma(x) \\ y^2(x^2 - c_1^2)(x^2 - c_2^2) &= y^2\sigma(x)\sigma(-x) \end{aligned}$$

The evaluations of these polynomials provide us with three checks that are zero in the two erased columns. The checks are independent in the three additional erasures, and together can correct all three erasures, if and only if

$$\det \begin{pmatrix} 1 & 1 & 1 \\ b_1/\sigma(-a_1) & b_2/\sigma(-a_2) & b_3/\sigma(-a_3) \\ b_1^2/a_1^2 & b_2^2/a_2^2 & b_3^2/a_3^2 \end{pmatrix} \neq 0.$$

□

Example 31. For $A = \{a : a^5 = 1\}$, the theorem can be used to construct $r \times 5$ array codes by reducing x^6 to x, or by using an equivalent format such as

$$\begin{pmatrix} 1 & \cdot & \cdot & \cdot & x^4 \\ y & xy & \cdot & \cdot & x^4y \\ y^2 & \cdot & x^2y^2 & \cdot & x^4y^2 \\ y^3 & \cdot & \cdot & x^3y^3 & x^4y^3 \\ \vdots & \cdot & \cdot & \cdot & \vdots \\ y^{r-1} & \cdot & \cdot & \cdot & x^4y^{r-1} \end{pmatrix}$$

For $r = 17$ and $B = \{b : b^{17} = 1\}$ we obtain a cyclic 17×5 (2,3) SD code. The code is defined over a field of size 2^8.

Example 32. The code in Example 29 can be enlarged to a 65×7 $(2, 3)$ SD code with $B = \{b : b^{65} = 1\}$. The code is defined over a field of size 2^{12}.

CHAPTER 3

CODES FOR REGENERATION, PART I

3.1 Binary Constructions for $v = 3$

3.1.1 The Σ Construction

First thing we can try, to obtain a $(n, n-2, n-1), v = 3$ code from a $(n, n-1, n-1), v = 3$ code is through giving each disk an additional Σ parity. We illustrate the construction using the $(n, k, d) = (5, 4, 4)$ example.

Example 33. In order to reduce $k = 4$ to $k = 3$, we need to add one symbol to each disk. We call this additional symbol Σ_i for disk D_i. Disk D_i contains six symbols from six distinct groups. We can choose Σ_i to be the sum of six symbols, one from each group such that it is not already contain in disk D_i. For example, $\Sigma_1 = A_2 + B_2 + C_2 + E_3 + X_3 + Y_4$.

	45	35	34	25	24	23	15	14	13	12	
D_1	A_1	B_1	C_1	E_1	X_1	Y_1					Σ_1
D_2	A_2	B_2	C_2				Z_2	U_2	V_2		Σ_2
D_3	A_3			E_3	X_3		Z_3	U_3		W_3	Σ_3
D_4		B_4		E_4		Y_4	Z_4		V_4	W_4	Σ_4
D_5			C_5		X_5	Y_5		U_5	V_5	W_5	Σ_5

3.1.2 Data Collection and Repair

For data collection, we take any three disks. Since the five disks are symmetric, we can assume WLOG we have D_1, D_2 and D_3. We have two symbols in the groups labeled by A, B, C, E, X, Z and U, so we can recover the symbols in these groups. Using the symbols in groups A, B, C, E, X together with Σ_1, we can isolate the symbol Y_4 in the Y group. Combining with Y_1 in D_1, we

can get all the Y symbols. Similarly, using the A, B, C, Z, U symbols, Σ_2 and V_2 on D_2 allows us to recover the V symbols. Finally we get the W symbols using the A, E, X, Z, U symbols, Σ_3 and W_3 to get all W symbols. Once the 30 symbols are recovered, we can use them to reconstruct Σ_4 and Σ_5.

For repair, recall that any symbol of the failed disk, besides Σ, can be recovered by receiving the three other symbols in its layer. Upon more careful inspection, we see that the helper data is also sufficient for reconstructing the sum. As the helper data contains the two other symbols for each symbol in the failed disk, and we just need one of the two for the sum. WLOG, take D_5 as the failed disk. We will receive C_1 from D_1 and C_2 from D_2 to recover C_3, but Σ_5 only need one of C_1 or C_2. The same goes for the X, Y, U, V and W symbols.

Remark 34. For any system with parameter (n, k, d) and $k = d = n - 1$, we have multiple corner points that bound the region of optimal trade-off for α and β. They are given as $(\alpha, \beta, B) = (\binom{k}{v-1}, \binom{k-1}{v-2}, (v-1)\binom{k+1}{v}))$, where $2 \leq v \leq k$. We see that in the example above, $v = 3$ is the size of the parity check in each column. The construction is actually general for any parameters (n, k, d) with $k = n - 2, d = n - 1$ and $m = 2$. These parameters guarantee that any $n - 2$ disks will have two symbols per group except $\binom{n}{1}$ symbols spread evenly, one per disk. The Σ construction then works to recover these groups disk by disk. At the cost of n additional symbols spread evenly across the n disks, we lower the access number by one.

3.2 Binary Construction for $v = 4$

Following the idea of the construction above, we modify the known layered code construction with $(n, k, d) = (6, 5, 5)$. However, for $v = 4$, we cannot simply use the Σ construction, since each column/layer has 4 symbols. When we are trying to collect data from 4 out of 6 disks, there will be $6 = \binom{4}{2}$ out of the $15 = \binom{6}{4}$ columns having only 2 symbols in the disks contacted. We need to know 1 more symbol per column to recover those 6 layers. So we would need to spread the 6 missing symbols across the 4 disks. Putting the symbols unevenly would result in potential problems since we need to treat the disks the same way to deal with all possible data collection scenarios. This points us to dividing the symbols, and this can be done using sub-packetization.

3.2.1 The Construction for (6,4,5), $v = 4$

We first look at the $(6, 5, 5)$ layered code construction, with focus on the checks involving disk D_1.

	45	35	34	25	24	23	56	46	36	26	12	...	...	16
D_1	A_1	B_1	C_1	E_1	X_1	Y_1	Z_1	U_2	V_1	W_1				
D_2	A_2	B_2	C_2				Z_2	U_2	V_2					
D_3	A_3			E_3	X_3		Z_3	U_3		W_3				
D_4		B_4		E_4		Y_4	Z_4		V_4	W_4				
D_5			C_5		X_5	Y_5		U_5	V_5	W_5				
D_6	A_6	B_6	C_6	E_6	X_6	Y_6					*	*	*	*

We construct the additional information for disk D_1 and the other disks will have a similar construction by symmetry. Take the group of symbols labeled by A for example. We need to have the 4 symbols to be on the 4 disks D_1, D_2, D_3, D_6 so that any 2 disks can be used to recover the 4 symbols. As mentioned above we use a sub-packetization level of 4, so each of the 4 symbols is turned into 4 sub-symbols in a different field. The following table illustrates the distribution of the A sub-symbols. Each row has 4 sub-symbols equivalent to 1 A symbol. The labelings in the boxes indicate the disk(s) that store(s) the sub-symbol.

D_1	12	1	16	13
D_2	26	23	2	21
D_3	3	36	31	32
D_6	61	62	63	6

So we have 30 extra sub-symbols that disk D_1 need to be able to provide. We can view the 10 groups of 3 sub-symbols as 3 vectors of length 10, forming a 3×10 matrix. Taking the product with a 10×3 MDS matrix M gives us a 3×3 matrix. We store this matrix on D_1. This matrix consisting of 9 sub-symbols plays a similar role as Σ_1 in the previous construction for $m = 2$.

3.2.2 Data Collection and Repair

For data collection, we take any four disks. Since the six disks are symmetric, we can assume WLOG we have D_3, D_4, D_5 and D_6. Since each column sums to zero, we can recover all groups except A, B, C, Z, U and V. In the diagram

above we omitted the labeling on the last 5 columns. They do not involve disk D_1 so each column will have at least 3 symbols shared among disks D_3, D_4, D_5 and D_6, and hence can be recovered.

For each of the A, B, C, Z, U and V groups, we need help from the additional information stored on disks D_3, D_4, D_5 and D_6. We can focus on the A group first. We need to have disk D_3 and D_6 use their additional information to help the A_3 and A_6 symbols which by themselves are not sufficient for recovering the entire A group. Disk D_6 has additional information stored in a 3×3 matrix. It is the product of the 3×10 matrix formed by the construction above. Since we know disks D_3, D_4, D_5 and D_6, we know 7 of the 10 columns in the 3×10 matrix, in particular the columns corresponding to groups E, X, Y and the 4 groups labeled by $*$ at the end in the schematic above. The 7 known columns together with the MDS matrix allow us to find the 3 unknown columns in the 3×10 matrix from the 3×3 matrix stored on D_6. Once we have the three columns, the column corresponding to the A group provide the 3 sub-symbols on the last row of the 4×4 schematics above. Similarly we can get the second last row from D_3, and hence the entire A group. The same process works for the B, C, Z, U and V groups, since the A group was chosen WLOG.

Similar to the previous construction, we see that repair is not an issue. The original layered code structure allows us to repair a failed disk with $d = 5$ and $\beta = 6$. The only thing to check is that the extra sub-symbols part can also be reconstructed from the repair information. This is certainly the case as we get all the symbols we need from the repair process. Take disk D_1 for example. We need all the labeled symbols in the schematic. They are indeed available as the repair process for layered code uses all 30 labeled symbols from disks D_2, D_3, D_4, D_5 and D_6 to recover the 10 labeled symbols in disk D_1. The 3×3 matrix we add to D_1 is precisely constructed from these 40 symbols.

Remark 35. We see that we can stay in the binary field when the "pre-parities" are generated for each disk. We only need a bigger field when we use a MDS matrix to generate the parities. However, this approach is not the most economical in terms of field size and computational complexity, as the MDS codes used are way more powerful than needed. If we look carefully, the set of missing symbols that could occur during data collection form a

relative small subset of all possible scenarios. This suggests we might be able to do better if we use codes weaker than MDS codes.

The example above prompts us to carry out the construction in two stages.

3.2.3 Generalizations

In the example above, it is convenient for us to construct a graph for each disk by putting the possible data collection scenarios (labeled by pairs of disks not immediately available) as vertices and adding edges between vertices with a common disk. We then partitioned the edges into 6 disjoint groups. This can be done as we know the graph, known as the $J(5,2)$ Johnson graph, is 6-regular and has chromatic index (minimal number of colors needed for an edge-coloring) 6. This leads us to look for such graphs for more general parameters. We claim that such graphs exist for an infinite family of parameters and we can partition the edges into disjoint groups to define parities as in the (6,4,5) case.

Theorem 36. For any $(n, k = n-2, d = n-1)$ with $v = 4$, we have a code over the binary field.

Proof. Following construction in Section 3.3.1, for each disk we will have a 4×4 sub-packetization matrix for each of the $\binom{n-1}{3}$ columns involving it, each contributing 3 extra sub-symbols. We need to protect these $3\binom{n-1}{3}$ sub-symbols with $2\binom{n-3}{1}$ parities. We define the parities via partitions of edges of a graph as in the (6,4,5) case above.

For disk D_i, we construct a graph by taking the 2-element subsets of the set $\{1, 2, \ldots, i-1, i+1, \ldots, n\}$, (corresponding to the $\binom{n-1}{2}$ data collection scenarios where the 2 elements are the inaccessible disks) to be the vertices; and connect two vertices v and w with an edge if they intersect in one element (with the edge corresponding to the common sub-symbol provided by disk D_i for the failure scenarios v and w). There are $\frac{2(n-3)}{2}\binom{n-1}{2} = 3\binom{n-1}{3}$ edges, and the graph is $2(n-3)$-regular.

We claim that the edges can be partitioned into $n-3$ disjoint 2-regular spanning subgraphs. Such a spanning 2-regular subgraph is called a 2-factor and a graph is said to be 2-factorizable if its edges can be partitioned into disjoint 2-factors. That is, we claim our graph, which has the name of the

Johnson $J(n-1,2)$ graph, is 2-factorizable. Indeed this is true by a theorem Petersen proved in 1891 stating that any 2k-regular graph is 2-factorizable.

Finally, for each of the $n-3$ 2-factors, we define 2 binary parities. Each of these 2-factors has $\binom{n-1}{2}$ edges. If $\binom{n-1}{2}$ is odd, we can use the two checks $(1,0,1,\ldots,0,1)$ and $(0,1,0,\ldots,1,0)$. If $\binom{n-1}{2}$ is even, we can use the two checks $(1,1,1,\ldots,1,1)$ and $(1,0,1,0,\ldots,1,0)$. □

3.2.4 An Example: $(n,k,d)=(n,n-2,n-1), v=5$, with Coefficients

For any $(n,k,d)=(n,n-1,n-2)$ and $v=5$, we follow the previous section and have our construction in two stages. For stage 1 we have binary solutions for any $(n,k,d)=(n,n-1,n-2)$. For stage 2 we have binary solutions for $n=7$.

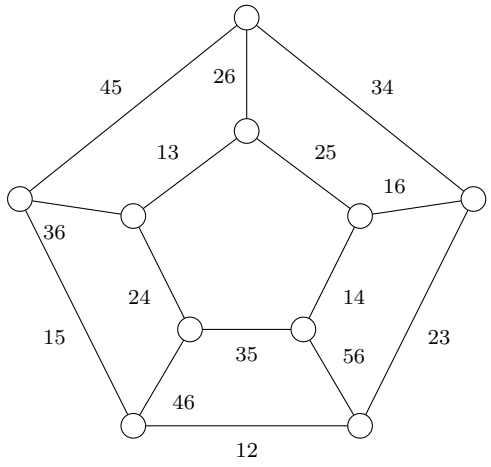
Example 37. For (n,k,d) = (7,5,6) and $v=5$, we use a different approach at the stage of sub-packetization. At each layer, the $v=5$ symbols form a check and once we have a data collection scenario that has 2 of the 5 disks being inaccessible, we need each of the 3 remaining disks to essentially provide "$\frac{1}{3}$ of a symbol". This leads us to a sub-packetization level that is divisible by 3. In this case, for (7,5,6), we can use a sub-packetization level of 3. We define the helper information pre-parities as follows:

	sub-symbols			pre-parity
a	a_1	a_2	a_3	$b_1+c_2+d_3$
b	b_1	b_2	b_3	$c_1+d_2+e_3$
c	c_1	c_2	c_3	$d_1+e_2+a_3$
d	d_1	d_2	d_3	$e_1+a_2+b_3$
e	e_1	e_2	e_3	$a_1+b_2+c_3$

where each column in the sub-symbol section sums to 0. We check that in case any two of the disks a through e are inaccessible, we can recover their content from the pre-parities stored in the 3 accessible disks. Since our construction has a cyclic shift symmetry, it suffices to check only two case. Case 1 we can take WLOG, disks d and e as being inaccessible. We can recover d_3 from the pre-parity stored in disk a together with the known content of disks b and c. Then we can recover the third column and hence e_3. This gives us d_2 from

the pre-parity stored in disk b. Knowing d_2 gives us the second column and hence e_2 and finally together with pre-parity stored on disk c we get d_1. Case 2 we can take WLOG, disks c and e as being inaccessible. We can recover e_1 from the pre-parity stored in disk d together with the known content of disks a and b. Then we can recover the first column and hence c_1. This gives us e_3 from the pre-parity stored in disk b. We can recover c_2 directly from the pre-parity stored in disk a together with the known content of disks b and d.

For each of the 7 disks, we get $\binom{6}{4} = 15$ pre-parities, of which $\binom{4}{2} = 6$ will be needed in any data collection scenario. So it suffices for us to give 6 checks on the 15 pre-parities. We illustrate this by showing how the checks are constructed for disk D_7. We can label each pre-parity by the layer it is in, which can be uniquely determined by a pair of numbers from $\{1, 2, \ldots, 6\}$ representing the 2 disks not involved in this layer. The 6 parity checks can be given conveniently as 6 cycles in the following graph:



Five of them are along the band formed by the inner and outer pentagons, and the last cycle is the inner pentagon. We can check that for any group of 6 edges corresponding to a data collection scenario, the symbols can be recovered using only these 6 checks. For example, if we take disks 1,2,3,4, and 7, disk 7 will need the pre-parities 12, 13, 14, 23, 24, 34. These can be recovered from known pre-parities one at a time.

3.3 General Constructions

3.3.1 Stage 1: Sub-packetization and Constructing Pre-Parities

As mentioned in Section 3.2, we formulate the problem of constructing our code in two stages. In stage 1 we generate, for each disk $a\binom{n-1}{v-1}$ "pre-parities", one for each layer, where a is the auxiliary sub-packetization parameter. Since the layers are independent, we can focus on a single layer.

Given v and ℓ, we need to find a and a' such that $(v-1)a$ information symbols can be encoded into a $v \times a'$ array such that the following holds: for any $1 \leq t \leq \ell$, any $v-1-t$ rows of the array will contain the $(v-1)a$ information symbols.

3.3.2 Local Codes via Chinese Remainder Theorem

We can construct codes that satisfy the requirements above by using the Chinese Remainder Theorem.

Construction 38. Take $a = lcm\{v-1-t \mid 1 \leq t \leq \ell\}$ and $a' = (v-1)a$. We construct an array of size $v \times a' = v \times (v-1)a$ containing $(v-1)a$ information symbols as follows.

First we construct a degree $(v-1)a-1$ polynomial, $p(x)$, with the $(v-1)a$ coefficients being the $(v-1)a$ information symbols we want to store in the array.

The i^{th} row of the array, $i = 1, 2, \ldots, v$, gets $a' = (v-1)a$ derivatives of $p(x)$ evaluated at $x = \alpha_i$, where α_i's are pairwise coprime.

Theorem 39. The $v \times a'$ array constructed above from the $(v-1)a$ information symbols has the property that for any $t = 1, 2, \ldots \ell$, any $v-1-t$ rows of the array will allow for recovery of the $(v-1)a$ information symbols.

Proof. For any $t = 1, 2, \ldots \ell$, we have $v-1-t$ rows each containing $(v-1)a$ remainders of $p(x)$ divided by $(x-\alpha_i)^j$, $j = 0, 1, \ldots, (v-1)a-1, i = 1, 2, \ldots, v-1-t$. We can take $v_t = \frac{(v-1)a}{(v-1-t)}$ many remainders each of $v-1-t$ rows. This is possible as $a = lcm\{v-1-t \mid 1 \leq t \leq \ell\}$. From the i^{th} row,

we get $\{a_{ij}\}, j = 0, 1, \ldots, v_t$, we can construct:

$$p_i(x) = \sum_{j=0}^{v_i-1} \frac{a_{ij}}{j!}(x - \alpha_i)^j$$

and view $p_i(x)$ as the Taylor polynomial of order $v_t - 1$ at α_i, of an unknown polynomial $q(x)$. Then we have

$$q(x) \equiv p_i(x) \pmod{(x - \alpha_i)^{v_t}}.$$

The Chinese remainder theorem asserts that there exists exactly one polynomial $q(x)$ of degree less than or equal to $(v-1)a - 1$ which satisfies these $v - 1 - t$ congruences. Since our original polynomial $p(x)$ satisfy these congruences by construction, we must have $q(x) = p(x)$. Therefore we have recovered the $(v-1)a$ information symbols as coefficients of $p(x)$. □

Example 40. Take for example $v = 5$ and $\ell = 2$. We can construct a 5 by $6 = lcm\{5 - 1 - t \mid 1 \leq t \leq 2\}$ array with $a' = 4 \times 6 = 24$ information symbols $c_0, c_1, \ldots, c_{23}$. We first let $p(x) = c_0 + c_1 x + c_2 x^2 \cdots + c_{23} x^{23}$. The array is filled as the following:

$p(\alpha_1)$	$p'(\alpha_1)$	$p''(\alpha_1)$	$\ldots$	$p^{(23)}(\alpha_1)$
$p(\alpha_2)$	$p'(\alpha_2)$	$p''(\alpha_2)$	$\ldots$	$p^{(23)}(\alpha_2)$
$p(\alpha_3)$	$p'(\alpha_3)$	$p''(\alpha_3)$	$\ldots$	$p^{(23)}(\alpha_3)$
$p(\alpha_4)$	$p'(\alpha_4)$	$p''(\alpha_4)$	$\ldots$	$p^{(23)}(\alpha_4)$
$p(\alpha_5)$	$p'(\alpha_5)$	$p''(\alpha_5)$	$\ldots$	$p^{(23)}(\alpha_5)$

In other words, the array has $a_{ij} = p^{(i)}(\alpha_j)$, $j = 0, 1, \ldots, a' - 1, i = 1, 2, \ldots, v$.

Suppose 3 of the 5 rows are not accessible, we have the 2 remaining rows can each provide its first 12 symbols to reconstruct the message by the theorem above.

When 2 of the 5 rows are not accessible, we have that the 3 remaining rows can each provide its first 8 symbols to reconstruct the message by the theorem above.

Remark 41. The above code construction actually has a more flexible access structure than what we strictly need in our stage 1 problem. In a way it can be seen to have similar properties to the Fountain codes. Each row of our array can be viewed as having a' pieces of information and when $v - 1 - t$

of them each provides any v_t pieces of information, we can reconstruct the original message of length $(v-1)a$.

3.3.3 Stage 2: Putting Codes on Pre-Parities

In stage 1 of the improved layered code over small field construction, we have a disk D_i storing lower order coefficients of $p_t(\alpha_i)$, where $t \in S$ and S is the set containing all $v-1$ subsets of $[n] \setminus \{i\}$ corresponding to the $\binom{n-1}{v-1}$ layers that D_i participates. Suppose we store coefficients up to order d. In order to accommodate a lowered k, we also need D_i to store some helper information as the higher order coefficients for terms of order $r \geq d$.

In stage 2 our goal is to have, for each $r, d \leq r \leq v-1$, a code on the helper information pre-parities across all layers, such that in any download scenario we can recover all helper information from the encoded helper information stored in the accessible layers. The layers are naturally organized on the vertices of a Johnson graph. We can put a code on the vertices of this Johnson graph to encode the helper information across layers. This brings us to the construction of Johnson graph codes.

3.4 Johnson Graph Codes

In order to define the code, we first establish some structure of the Johnson graph.

Definition 42. The **Johnson graph** $J(n, v)$ is the undirected graph with vertex set all v-subsets of a given n-element set such that two vertices are adjacent if they intersect in $v-1$ elements. The default choice for an n-element set is the set $\{0, 1, \ldots, n-1\}$.

	accessed nodes	missed	distance
#	$0,1,2,3,4,5,6$	7	r
21	$-\ -\ *\ *\ *\ *\ *$	$-$	0
35	$-\ -\ -\ *\ *\ *\ *$	$*$	1

Table 3.1: Accessing a code with layers of size $v = 5$ in $k = 7$ nodes.

	accessed nodes	missed	distance
#	$0,1,2,3$	$4,5,6,7$	r
4	$*\ *\ *\ *$	$*\ -\ -\ -$	0
24	$-\ *\ *\ *$	$*\ *\ -\ -$	1
24	$-\ -\ *\ *$	$*\ *\ *\ -$	2
4	$-\ -\ -\ *$	$*\ *\ *\ *$	3

Table 3.2: Accessing a code with layers of size $v = 5$ in $k = 4$ nodes.

We encounter Johnson graphs in two different ways. For a storage system with n nodes and for a parameter $2 \leq v \leq n$, layered codes are defined by $\binom{n}{v}$ disjoint checks of length v, one for each v-subset of nodes. The v-subsets are called layers. They form a set of vertices for a Johnson graph $J(n, v)$. A second graph $J(n, k)$ occurs as follows. For a regenerating code of type (n, k, d) data is collected by contacting any k nodes. The possible data collection scenarios are the $k-$subsets of an $n-$element set. They form the vertices of a Johnson graph $J(n, k)$.

For each collection scenario, i.e., for each k-subset $A \subset \{0, 1, \ldots, n-1\}$, we define a partition of the layers in the layered code, i.e., of the vertices in $J(n, v)$. Table 3.1 and Table 3.2 illustrate vertex partitions into shells for the cases $k > v$ and $k < v$.

For $k \geq v$, layers L with $L \subset A$ are fully accessible. They form the shell $S_0(A) = \{L : L \subset A\}$. Layers at distance r from $S_0(A)$ form the shell $S_r(A) = \{L : |L \backslash A| = r\}$. For $k \leq v$, layers are accessible in at most k symbols. The maximally accesible layers are those with $A \subset L$. They form the shell $S_0(A) = \{L : L \supset A\}$. The shell at distance r from these layers is

$S_r(A) = \{L : |A \backslash L| = r\}$. In general, since $k - v = |A \backslash L| - |L \backslash A|$,

$$L \in S_r(A) \quad \Leftrightarrow \quad r = \min(|L \backslash A|, |A \backslash L|). \tag{3.1}$$

For $L \in J(n, v)$, let $B_r(L)$ be the neighborhood of radius r around L.

Definition 43. For the Johnson graph $J(n, v)$ and for a subset A of $\{0, 1, \ldots, n-1\}$, define a **generalized neighborhood** as a union of vertex neighborhoods.

$$B_r(A) = \begin{cases} \cup_{L_0 \subset A} B_r(L_0) & (|A| \geq v). \\ \cup_{L_0 \supset A} B_r(L_0) & (|A| \leq v). \end{cases}$$

Definition 44. A **Johnson graph code** $JGC(n, v, k, r)$ on the vertices of the Johnson graph $J(n, v)$ is a code of length $N = \binom{n}{v}$ and dimension $K = |B_r(A)|$ such that, for any k-subset A of $\{0, 1, \ldots, n-1\}$, $B_r(A)$ forms an information set.

It is worth noting that the definition only refers to matroid properties of the code. In matroid terms it asks that the set of bases for the matroid of the code is large enough that it includes all generalized neighborhoods $B_r(A)$. The definition includes as trivial examples all MDS codes of length N and dimension K. Any Johnson graph code, and in particular any length N, dimension K MDS code, can be used in the construction of concatenated layered codes in Section 3.8. The codes constructed in the next section have the advantage of being defined over a field of size n instead of $N = \binom{n}{v}$ and moreover have a parity check structure that allows efficient erasure decoding. We include some lemmas for later use and prove in Proposition 51 that the dual of a Johnson graph code is again a Johnson graph code.

Lemma 45. Let $k = |A|$.

$$|B_r(A)| = \begin{cases} \sum_{i=0}^{r} \binom{n-k}{i} \binom{k}{v-i} & (k \geq v). \\ \sum_{i=0}^{r} \binom{k}{i} \binom{n-k}{n-v-i} & (k \leq v). \end{cases}$$

Proof. The first sum counts L with $|L \cap A^c| = i \leq r$ and $|L \cap A| = v - i$. The second sum counts L with $|L^c \cap A| = i \leq r$ and $|L^c \cap A^c| = n - v - i$. $\square$

We introduce the following notation. Let

$$d_1 = \min(v, n-k), \; d_2 = \min(k, n-v), \tag{3.2}$$

and let $R = \min(d_1, d_2)$.

Lemma 46. $S_r(A) = \emptyset$ for all $r > R$.

Proof. Use (3.1) with $\min(|L\backslash A|, |A\backslash L|) \leq \min(d_1, d_2) = R$. □

Lemma 47. $S_r(A) = S_{R-r}(A^c)$.

Proof. For $L \in J(n, v)$, let

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} |L \cap A^c| & |L \cap A| \\ |L^c \cap A^c| & |L^c \cap A| \end{pmatrix}$$

Then $L \in S_r(A)$ for $r = \min(a, d)$ and $L \in S_s(A^c)$ for $s = \min(b, c)$. Where r and s are related to R via $r + s = \min(a + b, a + c, b + d, c + d) = \min(v, n - k, k, n - v) = R$. □

The lemma gives a dual description for the shell $S_r(A)$ in terms of the complement A^c of A.

Example 48. The four layers $S_0(0123) = \{01234, 01235, 01236, 01237\} \subset J(8, 5)$ in the top row of Table 3.2 can be described dually as $S_3(4567)$.

A different dual description is obtained by considering the set $\{L^c : L \in S_r(A)\}$ as a subset of vertices in the Johnson graph $J(n, n-v)$. Note that this graph is isomorphic to $J(n, v)$ with only difference that the vertex labels $L \subset \{0, 1, \ldots, n-1\}$ are replaced with their complements $L^c = \{0, 1, \ldots, n-1\}\backslash L$. Two v-subsets are adjacent in $J(n, v)$ if and only if their complements are adjacent in $J(n, n - v)$.

Lemma 49. For shells $S_r(A) \subset J(n, v)$ and $S_{R-r}(A) \subset J(n, n - v)$,

$$L \in S_r(A) \;\Leftrightarrow\; L^c \in S_{R-r}(A).$$

Proof. As in the proof of Lemma 47, $L \in S_r(A)$ for $r = \min(a, d)$ and $L^c \in S_s(A)$ for $s = \min(c, b)$. So that $r + s = R$. □

Lemma 50.

$$B_r(A)^c = B_{R-r-1}(A^c).$$

Proof. Use that $B_r(A)^c = \cup_{t>r} S_t(A)$ and apply Lemma 47. □

Proposition 51. The dual code of a Johnson graph code $JGC(n, v, k, r)$ is a code $JGC(n, v, n-k, R-r-1)$.

Proof. The code has the required length and dimension. Lemma 50 and the fact that the complement of an information set is an information set for the dual code, proves that $B_{R-r-1}(A^c)$ is an information set for any k-subset A. □

3.5 Construction of Johnson Graph Codes

We give a general construction for Johnson graph codes of length $N = \binom{n}{v}$ from MDS codes of length n. Section 3.7 describes in more detail the special case where the length n MDS code is of Reed-Solomon type. Codewords of length N will have their coordinates indexed by the vertices of the Johnson graph $J(n, v)$. The codes are generated by the coordinatization vectors of suitably chosen matrices M. For a $v \times n$ matrix M,

$$\pi(M) = (\det(M_L) : L \in J(n, v)),$$

where M_L is the $v \times v$ minor of M with columns in the v-subset $L \subset \{0, 1, \ldots, n-1\}$.

Definition 52. For given n and v, let C_0 be a code of length n and let $t \geq 0$. Define a code C of length N as the span of the vectors $\pi(M)$ for all $v \times n$ matrices M with at least t rows in C_0.

We show in two steps that the code C is a Johnson graph code $JGC(n, v, k, r)$ with $k = \dim C_0$ and $r = \min(v, \dim C_0) - t$.

Lemma 53. The code C has dimension $K = |B_r(I_0)|$, where $r = \min(v, \dim C_0) - t$ and $I_0 \subset \{0, 1, \ldots, n-1\}$ is of size $\dim C_0$.

Proof. Let $g = \{g_0, g_1, \ldots, g_{n-1}\}$ be a basis for F^n and let $\{g_i : i \in I_0\}$ be a basis for C_0. The vectors $\pi(M)$, for matrices M with rows $\{g_i : i \in L\}$ such that $|L \cap I_0| \geq t$, form a basis for C. Furthermore,

$$|L \cap I_0| = t \quad \Leftrightarrow \quad \min(|L \backslash I_0|, |I_0 \backslash L|) = \min(v - t, \dim C_0 - t).$$

And thus, using (43) with $r = \min(v, \dim C_0) - t$,

$$\{\ L \in J(n,v) \mid |L \cap I_0| \geq t\ \} = B_r(I_0).$$

□

From the lemma it is clear that the dimension of the code C only depends on $\dim C_0$ and not on C_0. The occurrence of information sets in C on the other hand depends on the choice of the code C_0.

Lemma 54. For C as in the definition, and for an information set $A_0 \subset \{0, 1, \ldots, n-1\}$ for C_0, the set

$$\{\ L \in J(n,v) \mid |L \cap A_0| \geq t\ \} = B_r(A_0)$$

is an information set for C.

Proof. The size of $B_r(A_0)$ equals the dimension of C. It therefore suffices to prove the independence of the coordinates in $B_r(A_0)$. For each $L \in B_r(A_0)$ construct a codeword $\pi(M)$ from a $v \times n$ matrix M with two types of rows. For $j \in L \cap A_0$ include as a row in M the unique codeword in C_0 that is 1 in j and 0 in $A_0 \backslash j$. For $j \in L \backslash A_0$ include as row in M the unit vector that is 1 in j and 0 elsewhere. Up to a permutation of columns that puts the columns in $A_0 \backslash L$ in the leading positions, followed by columns in $L \cap A_0$ and $L \backslash A_0$, M will be of the form

$$M = \left(\begin{array}{c|c|c|c} O & I & X & Y \\ \hline O & O & I & O \end{array}\right), \quad M_L = \left(\begin{array}{c|c} I & X \\ \hline O & I \end{array}\right).$$

Clearly, $\det(M_L) = 1$. And $\det(M_{L'}) = 0$ for $L' \neq L$ with $|L' \cap A_0| \geq |L \cap A_0|$. The constructed codewords $\pi(M)$, for $L \in S_r(A_0), \ldots, S_1(A_0), S_0(A_0)$, in that order, form an invertible triangular matrix in the positions $B_r(A_0)$. And thus $B_r(A_0)$ is an information set for C. □

Proposition 55. For an MDS code C_0, the code C in Definition 52 is a Johnson graph code.

Proof. Lemma 53 and Lemma 54. □

Lemma 56. For each $L \in S_r(A_0)$, the proof of Lemma 54 constructs a codeword $\pi(M)$ that depends on L. It has the special property that the coordinate $\pi(M)_L$ is the unique nonzero coordinate in the positions $B_r(A_0)$. Moreover the weight of $\pi(M)$ is at most

$$\binom{2t + n - \dim C_0 - v}{t}.$$

Proof. For $r = \min(v, \dim C_0) - t$, we have, as in the proof of Lemma 53, that $L' \in B_r(A_0)$ if and only if $| L' \cap A_0 | \geq t$. For $L \in S_r(A_0)$, equality holds and $| L \cap A_0 | = t$. It follows that $|L' \cap A_0| \geq |L \cap A_0|$ for all $L' \in B_r(A_0)$, and thus, as in the proof of Lemma 54, that $\det(M_{L'}) = 0$ for all $L' \in B_r(A_0)$, $L' \neq L$. For the weight of $\pi(M)$, note that a full minor $M_{L'}$ in the matrix M is nonsingular only if $M_{L'}$ contains all columns in $L \backslash A_0$ and no columns in $A_0 \backslash L$, i.e., only if

$$L \cap A_0^c \subset L' \subset L \cup A_0^c.$$

For $| L \cap A_0 | = t$, $| L \cap A_0^c | = v - t$ and $| L \cup A_0^c | = n - \dim C_0 + t$. The number of L' such that $M_{L'}$ is nonsingular is therefore at most

$$\binom{2t + n - \dim C_0 - v}{t}.$$

□

Example 57. For the Johnson graph $J(n = 5, v = 3)$, let

$$g = \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Let $C_0 = \text{span}(g_0, g_1)$, $t = 2$. Then $r = 0$, $I_0 = \{0, 1\}$, $B_0(I_0) = \{012, 013, 014\}$,

and the code C is spanned by the first block of rows in the matrix

012	1	·	·	1	1	·	1	0	·	1
013	·	1	·	0	·	1	1	·	0	1
014	·	·	1	·	0	1	·	1	1	1
023	·	·	·	1	·	·	·	·	·	0
024	·	·	·	·	1	·	·	·	·	1
034	·	·	·	·	·	1	·	·	·	1
123	·	·	·	·	·	·	1	·	·	1
124	·	·	·	·	·	·	·	1	·	1
134	·	·	·	·	·	·	·	·	1	0
234	·	·	·	·	·	·	·	·	·	1

Matrix entries · are 0 independent of the choice of the code C_0. Lemma 56 gives the number of remaining entries in a row: $\binom{4}{2}$, $\binom{2}{1}$ or $\binom{0}{0}$, for rows in the first, second, or third block of rows. The presence of additional zeros in the matrix is due to C_0 not being MDS. The sets $A_0 = \{0, 2\}$ and $A_0 = \{1, 4\}$ are not information sets for C_0. For each of the remaining $A_0 \in J(5, 2)$, $B_0(A_0)$ is an information set for C.

For $t = 0$, and given a basis $g = \{g_0, g_1, \ldots, g_{n-1}\}$ for F^n, the proof of Lemma 54 assigns to each $L \in J(n, v)$ a unique matrix M and vector $\pi(M)$. The vectors $\pi(M)$ form an invertible $N \times N$ matrix $\Lambda(g)$. The assignment of $\Lambda(g)$ to an ordered list of vectors g is functorial. That is, if g and h are two ordered lists of vectors and both are interpreted as $n \times n$ matrices, with product gh as $n \times n$ matrices, then $\Lambda(gh) = \Lambda(g)\Lambda(h)$ as product of $N \times N$ matrices. The functoriality property says that the determinant of a $v \times v$ minor in gh can be expressed in terms of determinants of $v \times v$ minors in g and h. Both properties are classical and have short proofs.

Lemma 58 (Cauchy-Binet formula)**.** For a $v \times n$ matrix A and $n \times v$ matrix B,

$$\det(AB) = \pi(A) \cdot \pi(B^T).$$

Proof. Compare determinants in the $(v + n) \times (v + n)$ matrix product

$$\left(\begin{array}{c|c} I_v & A \\ \hline 0 & I_n \end{array}\right) \left(\begin{array}{c|c} 0 & -A \\ \hline B & I_n \end{array}\right) = \left(\begin{array}{c|c} AB & 0 \\ \hline B & I_n \end{array}\right).$$

using a cofactor expansion over the minors $-A_L$ of $-A$, for $L \in J(n, v)$, for the second matrix.

$$\det\left(\begin{array}{c|c} 0 & -A \\ \hline B & I_n \end{array}\right) = \sum_L \det\left(\begin{array}{c|c|c} 0 & -A_L & 0 \\ \hline B_L & 0 & 0 \\ \hline 0 & 0 & I_{n-v} \end{array}\right)$$
$$= \sum_L \det(A_L)\det(B_L) = \pi(A)\cdot\pi(B^T).$$

In the first equality, the blocks $-A_L$ and B_L can be assumed to be in the given positions after replacing the matrix with a conjugate, i.e. by applying the same permutations to both rows and columns. □

Proposition 59. Let D_0 be the dual code of C_0. The dual code D of C is generated by vectors $\pi(M)$, for all $v \times n$ matrices M with at least $v + 1 - t$ rows in D_0.

Proof. For generators $\pi(A) \in C$ and $\pi(B) \in D$, the $v \times v$ matrix AB^T has a $t \times (v+1-t)$ all zero submatrix. The extended $t \times v$ submatrix of AB^T is of rank at most $t-1$ and thus $\det(AB^T) = 0$. With the lemma, the generators are orthogonal. □

Lemma 60 (Functoriality of exterior algebras)**.** The assignment $g \mapsto \Lambda(g)$, that maps a $n \times n$ matrix g to a $N \times N$ matrix $\Lambda(g)$, satisfies

$$\Lambda(gh) = \Lambda(g)\Lambda(h).$$

Proof. Evaluate entries of $\Lambda(gh)$ using the Cauchy-Binet formula. □

The functorial property $\Lambda(gh) = \Lambda(g)\Lambda(h)$ does not depend on the chosen orderering for the rows and columns in $\Lambda(g)$, i.e., on the ordering of the vertices in $J(n, v)$. However, the matrix $\Lambda(g)$, and in particular its shape, depends on the chosen ordering. The ordering that we use to define Johnson graph codes is in general different form the lexicographic ordering and depends on k. Let $E_0 = \{0, \ldots, k-1\}$. A vertex L' precedes L if $|L' \cap E_0| > |L \cap E_0|$. In case of intersections of equal size we order L' and L lexicographically. The partitioning of $J(n, v)$ into shells $S_r(E_0)$, for $r = 0, 1, \ldots, R$, gives the matrix $\Lambda(g)$ a block structure, such that shells appear in the order $S_0(E_0), S_1(E_0), \ldots, S_R(E_0)$, and such that rows and columns

within the same shell are ordered lexicographically. We call the modified order the k-lexicographic order, with notation $L' \leq_k L$. It is a total order on the vertices of $J(n, v)$.

Lemma 61. For an upper triangular matrix g, the matrix $\Lambda(g)$ is upper triangular whenever the vertex ordering on $J(n, v)$ is a refinement of the Bruhat order, defined by the rule $(\beta_0, \ldots, \beta_{v-1}) \leq (\alpha_0, \ldots, \alpha_{v-1})$ if $\beta_i \leq \alpha_i$, for all i. Both the lexicographic and the k-lexicographic order on $J(n, v)$ have this property.

Proof. The entry $\Lambda(g)_{\beta,\alpha} = \det(g_{ij} : i \in \beta, j \in \alpha)$ is nonzero only if $\beta_i \leq \alpha_i$ for all i, i.e. only if $\beta \leq \alpha$ in the Bruhat order. Clearly this implies $\beta \leq \alpha$ in the lexicographic order as well as $|\beta \cap E_0| \geq |\alpha \cap E_0|$. And therefore $\beta \leq_k \alpha$. □

A square matrix has all its pivots on the main diagonal if it is full rank and if it reduces to echelon form without row permutations. Clearly this is the case if and only if the matrix has a LU factorization.

Proposition 62. Let $\Lambda(g)$ be defined with an ordering of rows and columns that refines the Bruhat order. If the matrix g has a LU decomposition then so has the matrix $\Lambda(g)$.

Proof. If $g = LU$ then by functoriality $\Lambda(g) = \Lambda(L)\Lambda(U)$. For the given order, $\Lambda(L)$ is lower triangular and $\Lambda(U)$ is upper triangular by Lemma 61 □

Let $g = \{g_0, g_1, \ldots, g_n\}$ be a basis for F^n. We say that g is in block form, with respect to a code C_0 and a choice of information set A_0 for C_0, if the coordinates are ordered such that $A_0 = \{0, 1, \ldots, k-1\}$, if C_0 is the span of rows $\{g_0, g_1, \ldots, g_{k-1}\}$, and if the remaining rows are zero in the positions A_0.

$$\text{block form: } g = \left(\begin{array}{c|c} G_0 & G \\ \hline 0 & G_1 \end{array}\right), \qquad \text{systematic form: } g = \left(\begin{array}{c|c} I & G \\ \hline 0 & -I \end{array}\right).$$

A matrix in block form can be reduced, using a combination of column permutations within bocks of columns and row operations within blocks of rows, to a block form with $G_0 = I$ and $G_1 = -I$. In the reduced systematic

form, $g^2 = I_n$. Thus, by Lemma 60, also $\Lambda(g)^2 = I_N$, independent of the ordering of vertices in $J(n, v)$.

Lemma 63. For a matrix g in block form, and for a $k-$lexicographic ordering of the vertices in $J(n, v)$, the matrix $\Lambda(g)$ is an upper triangular block matrix. Moreover, if g is in systematic form then the diagonal blocks are identity matrices up to sign.

Proof. As in the proof of Lemma 61, $\Lambda(g)_{\beta,\alpha}$ is nonzero only if $|\beta \cap E_0| \geq |\alpha \cap E_0|$. The blocks I and $-I$ in g imply that the entry $\Lambda(g)_{\beta,\alpha}$ is nonzero for $|\beta \cap E_0| = |\alpha \cap E_0|$ only if $\beta = \alpha$. □

3.6 Dual constructions

In the previous section we defined codes of length $N = \binom{n}{v}$ with coordinates indexed by the vertices of the graph $J(n, v)$. Codes are generated by vectors

$$\pi(M) = (\det(M_L) : L \in J(n, v)),$$

for $v \times n$ matrices M. A vertex $L \in J(n, v)$ has complement $L^c \in J(n, n - v)$. As pointed out earlier replacing L with L^c amounts to a relabeling of vertices for the same graph. With the relabeling, the matrices in the code construction are replaced by $n - v \times n$ matrices. We describe the relation between the two different cases of the same construction and include a formal verification of their equivalence. In particular, for the code C and its dual code D, we find an equivalent pair of dual codes C' and D'.

For a layer L with complement L^c reorder the columns in a $n \times n$ identify matrix I_n as $\left(\begin{array}{c|c} I_L & I_{L^c} \end{array} \right)$ and write $\text{sign}(L) = \det(I_L | I_{L^c})$. The coordinatization vector $\pi(M)$ is defined in terms of $v \times v$ determinants.

$$\pi(M)_L = \det(M_L) = \det(M I_L).$$

A different way to coordinatize the matrix M is by extending M with unit vectors e_i, for $i \in L^c$, and then taking the determinant of a $n \times n$ matrix.

$$\pi'(M)_L = \det \left(\frac{M}{{I_{L^c}}^T} \right).$$

The two approaches are equivalent and give vectors that are equal up to sign in each coordinate. From

$$\left(\begin{array}{c} M \\ \hline {I_{L^c}}^T \end{array}\right)\left(\begin{array}{c|c} I_L & I_{L^c} \end{array}\right) = \left(\begin{array}{c|c} M_L & M_{L^c} \\ \hline 0 & I_{n-v} \end{array}\right)$$

it follows that

$$\pi'(M)_L = \mathrm{sign}(L)\pi(M)_L. \tag{3.3}$$

The function $\mathrm{sign}(L) = \det(I_L \mid I_{L^c})$ depends only on $\sum_{i\in L} i \bmod 2$ and is normalized such that $\mathrm{sign}(L) = 1$ for $L = \{0, 1, \ldots, v-1\}$.

Definition 64. Given a code C with coordinates indexed by $J(n, v)$, define the signed version C' of C as the equivalent code obtained by changing codewords in each of the coordinates $L \in J(n, v)$ by a factor $\mathrm{sign}(L)$.

For a code C generated by vectors $\pi(M)$, the code C' is the code generated by vectors $\pi'(M)$, for the same matrices M. Let D be the dual code of C with signed version D'. We use the following lemma to describe D' directly in terms of C, for the Johnson graph code C in Definition 52.

Lemma 65. Let A be a $v \times n$ matrix and let B be a $(n-v) \times n$ matrix such that the row spaces of A and B have nonzero intersection. Then

$$\sum_L \pi'(A)_L \pi(B)_{L^c} = \sum_L \mathrm{sign}(L) \det(A_L) \det(B_{L^c}) = 0.$$

Proof. The first equality uses (3.3). The expression in the middle is the cofactor expansion, over the minors A_L of A, for $L \in J(n, v)$, for the determinant

$$\det\left(\begin{array}{c} A \\ \hline B \end{array}\right) = 0.$$

□

Let C_0 be a code of length n. The code C in Definition 52 is generated by vectors $\pi(M)$, for all $v \times n$ matrices M with at least t rows in C_0.

Proposition 66. The signed version D' of the dual code D of C is generated by vectors $\pi(M)$, for all $(n-v) \times n$ matrices M with at least $\dim C_0 + 1 - t$ rows in C_0.

	C_0	# rows in $C_0 \geq$	D_0	# rows in $D_0 \geq$
$J(n, v)$	C	t	D	$v + 1 - t$
$J(n, n - v)$	D'	$\dim C_0 + 1 - t$	C'	$\dim D_0 - v + t$

Table 3.3: Code construction for the codes C, D, D' and C'.

Proof. Let $\pi(A)$ and $\pi(B)$ be generators, such that A has at least t rows in C_0 and B has at least $\dim C_0 + 1 - t$ rows in C_0. We may assume that both A and B have independent rows, for otherwise $\pi(A) = 0$ or $\pi(B) = 0$. Thus the row spaces of A and B have a nonzero intersection inside C_0. With the lemma, the signed vector $\pi'(A)$ and the vector $\pi(B)$ are orthogonal. Equivalently, the vector $\pi(A)$ and the vector $\pi'(B)$ are orthogonal. Thus C and D' are orthogonal spaces. Moreover, having complementary dimensions, they are dual codes. □

Table 3.3 summarizes the choice of generators $\pi(M)$ for the codes C, D, D' and C' using the construction in Definition 52. Proposition 59 relates constructions in the same row and Proposition 66 relates constructions in the same column. The codes C and D, and therefore also the equivalent codes C' and D', are different from the trivial codes 0 and F^N only if $0 < t \leq \dim C_0$ and $0 < v + 1 - t \leq \dim D_0$.

For comparison, Lemma 54 uses generators $\pi(M)$ for C with M of the form

$$M = \left(\begin{array}{c|c|c|c} O & I & X & Y \\ \hline O & O & I & O \end{array} \right).$$

The codes C and D are nontrivial only if the column block sizes, from left to right, satisfy

$$\dim C_0 - t \geq 0,\ t > 0,\ v - t \geq 0,\ \dim D_0 - v + t > 0.$$

The construction for D in Table 3.3 is given by Proposition 59 and the construction for D' by Proposition 66. The construction for C' uses the two propositions in combination, either by considering C' as dual of D', or by considering C' as signed dual of D. We include a direct proof for the construction of C'. It is based on the following lemma.

Lemma 67. Let M and N be full rank matrices of sizes $v \times n$ and $(n-v) \times n$, respectively, with orthogonal row spaces. Then $\pi(N)$ is equal to $\pi'(M)$ up to a scalar factor.

Proof. A $v \times v$ minor in M is singular if and only if the complimentary $(n-v) \times (n-v)$ minor in N is singular. Thus $\pi(N)$ and $\pi'(M)$ have zeros in the same positions. For a pair of invertible minors M_1 and M_2 in M, with columns in L_1 and L_2, respectively, denote the complementary minors in N by N_1 and N_2. We need to show that $\text{sign}(L_1)\det(M_1)\det(N_2) = \text{sign}(L_2)\det(M_2)\det(N_1)$. It suffices to prove the special case where L_1 and L_2 are adjacent. The claim is not affected by row operations on M or on N, or by a permutation of the columns in M and N. Without loss of generality, we may therefore assume that $L_1 = \{0, 1, \ldots, v-1\}$, $L_2 = \{1, 2, \ldots, v\}$, and that

$$M = \left(\begin{array}{c|c} I_v & X \end{array} \right), \quad N = \left(\begin{array}{c|c} -X^T & I_{n-v} \end{array} \right).$$

So that M_1 and N_1 are identity matrices. The claim reduces to $\det(N_2) = \text{sign}(L_2)\det(M_2)$ and this is true for $\det(N_2) = -X_{1,1}$, $\text{sign}(L_2) = (-1)^v$ and $\det(M_2) = (-1)^{v-1}X_{1,1}$. □

Proposition 68. The signed version C' of the code C is generated by vectors $\pi(M)$, for all $(n-v) \times n$ matrices M with at least $\dim D_0 - v + t$ rows in D_0.

Proof. We give a proof that uses Lemma 67. For dual codes C_0 and D_0, with information sets in the leading and trailing positions, respectively, define a pair of $n \times n$ kernel matrices

$$g = \left(\begin{array}{c|c} I_k & B \\ \hline 0 & -I_{n-k} \end{array} \right), \quad h = \left(\begin{array}{c|c} -I_k & 0 \\ \hline -B^T & I_{n-k} \end{array} \right).$$

Such that rows $\{g_1, \ldots, g_k\}$ generate C_0 and rows $\{h_{k+1}, \ldots, h_n\}$ generate D_0. Generators $\pi(M)$ for C are chosen with matrices M that contain v rows in g with at least t rows in $\{g_1, \ldots, g_k\}$. The rows in the complementary positions in h form a matrix N that has at least $\dim D_0 - v + t$ rows in $\{h_{k+1}, \ldots, h_n\}$. The matrices M and N have orthogonal row spaces. By Lemma 67, $\pi(N)$ is proportional to $\pi'(M)$. It follows that the vectors $\pi(N)$ generate C'. □

A weaker claim for the proposition is that the construction yields a Johnson graph code with the same parameters as C and the same information sets. As

Johnson graph code, C is of type $JGC(n, v, k, r)$ for $r = \min(v, \dim C_0) - t$. The constructed code is of type $JGC(n, n-v, n-k, s)$ for $s = \min(n - v, \dim D_0) - (\dim D_0 - v + t) = \min(\dim C_0, v) - t = r$. Equality of the information sets, up to the graph isomorphism between $J(n, v)$ and $J(n, n - v)$, follows from

$$\begin{aligned} L \in S_r(A) \subset J(n, v) &\Leftrightarrow r = \min(|L \cap A^c|, |A \cap L^c|) \\ &\Leftrightarrow L^c \in S_r(A^c) \subset J(n, n - v). \end{aligned}$$

Theorem 69 gives a refinement of Lemma 56. As in (3.2), let $d_1 = \min(v, n - k)$ and $d_2 = \min(k, n - v)$. And let $R = \min(d_1, d_2)$. For an $n \times n$ matrix

$$h = \left(\begin{array}{c|c} I_{n-k} & B \\ \hline 0 & -I_k \end{array} \right),$$

the $N \times N$ matrix $\Lambda(h)$ with rows of the form $\pi(M)$ is such that the leading $|B_r(I_0)|$ rows, for $I_0 = \{0, 1, \ldots, n - k - 1\}$, span $JGC(n, v, n - k, r)$, for $0 \leq r \leq R$. The rows of $\Lambda(h)$ partition into shells $J(n, v) = \cup S_i(I_0)$, $0 \leq i \leq R$. For $A_0 = \{0, 1, \ldots, n - k - 1\}$, the columns of $\Lambda(h)$ partition into shells $J(n, v) = \cup S_j(A_0)$, $0 \leq j \leq R$. The partition of rows and columns gives the matrix $\Lambda(h)$ a block structure.

Theorem 69. The $i-$th block of rows in $\Lambda(h)$ contains

$$|S_i(I_0)| = \binom{n-k}{d_1 - i} \binom{k}{d_2 - i} \tag{3.4}$$

rows. The $j-$th segment in a row of the $i-$th block of rows in $\Lambda(h)$ contains at most

$$\binom{d_1 - i}{j - i} \binom{d_2 - i}{j - i} \tag{3.5}$$

nonzero entries. The total number of nonzero entries in a row of the $i-$th block is

$$\binom{d_1 + d_2 - 2i}{R - i}. \tag{3.6}$$

Proof. A row $\pi(M)$ in $\Lambda(h)$ belongs to the $i-$th block if

$$M = \left(\begin{array}{c|c|c|c} O & I & X & Y \\ \hline O & O & I & O \end{array} \right)$$

has column block sizes, from left to right,

$$\dim D_0 - d_1 + i,\ d_1 - i,\ v - d_1 + i,\ \dim C_0 - v + d_1 - i.$$

The size of the last block can be written $k - v + \min(v, n-k) - i = \min(k, n-v) - i = d_2 - i$. This proves the number of choices for M in (3.4). A layer $L \in S_j(A_0)$ with $\det(M_L) \neq 0$ has number of coordinates in each block, from left to right,

$$0,\ d_1 - j,\ v - d_1 + i,\ i - j.$$

This proves the number of choices for L in (3.5). Summation of (3.5) over $i \leq j \leq R$ yields (3.6). □

3.7 Construction using Reed-Solomon codes

We consider the construction of Johnson graph codes $JGC(n, v, k, r)$ in Definition 52 for the special case that the MDS code C_0 is a Reed-Solomon code. In general, a code of length n and dimension k is MDS if a codeword is uniquely determined by any k of its n coordinates. For distinct field elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \in F$, and for $f \in F[x]$, let

$$ev(f) = (f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_{n-1})).$$

For $0 \leq k \leq n$, a Reed-Solomon code of dimension k is defined as the space of vectors $\{ev(f) : \deg f < k\}$. A codeword $ev(f)$ is uniquely determined by any k of its coordinates and thus the code is MDS. The vectors $ev(x^i)$, for $i = 0, 1, \ldots, n-1$, form a basis for F^n. They form a $n \times n$ Vandermonde matrix g with rows $g_i = ev(x^i)$ such that the leading k rows of g span a Reed-Solomon code of dimension k, for any $0 \leq k \leq n$.

In the previous section we used the matrix g in systematic form to describe relations among a Johnson graph code C, its dual code D, and the equivalent codes C' and D'. Theorem 69 uses the dual matrix h in systematic form to obtain sparse parity check vectors for efficient erasure correction. In this section we consider different properties and interpretations that occur when g is a matrix of Vandermonde type, including connections with principal subresultants, monomial symmetric functions, Schubert codes, Chinese

remainder based codes, and product matrix codes. When we row reduce the matrix g it will in general be to block form rather than systematic form.

For the special case of Reed-Solomon codes, the construction in Definition 52 assigns to a subset $I \in J(n,v)$ a coordinatization vector $\pi(M)$, for the matrix M with rows $\{ev(x^i)\}_{i\in I}$. The vector $\pi(M)$ has coordinates, for $L \in J(n,v)$,

$$\pi(M)_L \;=\; \det(\,\alpha_j^i : i \in I, j \in L\,) =: \det(x^I; \alpha_L).$$

By Proposition 55, the vectors

$$\{\ (\det(x^I;\alpha_L)) : L \in J(n,v))\ :\ I \cap \{0,1,\ldots,k-1\} \geq t\ \}$$

form a basis for a Johnson graph code C of type $JGC(n,v,k,r)$, where $r = \min(v,k) - t$. By Proposition 59, the vectors

$$\{\ (\det(x^I;\alpha_L) : L \in J(n,v))\ :\ I \cap \{0,1,\ldots,n-k-1\} \geq v+1-t\ \}$$

form a basis for a dual Johnson graph code D of type $JGC(n,v,n-k,R-r-1)$.

Example 70. The code $C = JGC(5,2,2,1)$ and its dual D, both defined with $J(5,2)$, are generated by vectors

$$\pi(M) = (\,\det \begin{pmatrix} \alpha_{j_0}^{i_0} & \alpha_{j_1}^{i_0} \\ \alpha_{j_0}^{i_1} & \alpha_{j_1}^{i_1} \end{pmatrix}\ :\ 0 \leq j_0 < j_1 \leq 4\,).$$

$$C = JGC(5,2,2,1)\ :\ 0 \leq i_0 < i_1 \leq 4,\ i_0 \leq 1.$$
$$D = JGC(5,2,3,0)\ :\ 0 \leq i_0 < i_1 \leq 4,\ i_1 \leq 2.$$

In general, the dual code D_0 of a Reed-Solomon code C_0 is equivalent to a Reed-Solomon code. For the general case, when the n elements form a subset $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\} \subset F$, let $p(z) = (z-\alpha_0)(z-\alpha_1)\cdots(z-\alpha_{n-1})$. If C is defined with g then D is defined with $g\Delta$, for
$\Delta = -\mathrm{diag}(p'(\alpha_0), p'(\alpha_1), \ldots, p'(\alpha_{n-1}))^{-1}$.

Let $E = \{0,1,\ldots,n-1\}$. The $n \times n$ Vandermonde matrix g has rows $\{ev(x^i)\}_{i\in E}$. Let $f_i(x) = \prod_{j<i}(x-\alpha_j)$, for $i \in E$. The rows $\{ev(f_i)\}_{i\in E}$

describe the matrix g in row reduced upper triangular form. A matrix g in row reduced block form depends on a choice of information set $A \subset E$ for C_0. After reordering columns we may assume $A = \{0, 1, \ldots, k-1\}$. Let

$$h_i(x) = \begin{cases} x^i, & 0 \leq i \leq k-1. \\ f_k(x)x^{i-k}, & k \leq i \leq n-1. \end{cases}$$

The rows $\{ev(h_i)\}_{i \in E}$ describe the matrix g in row reduced block form.

Remark 71. The two reduced forms for g, the upper triangular form and the block form, are obtained with lower triangular row operations and the three matrices share the same row spaces for a set of k leading rows, for any $0 \leq k \leq n$.

For $g = \{ev(f_i)\}_{i \in E}$ in upper triangular form, the matrix $\Lambda(g)$ has entries

$$\Lambda(g)_{I,L} = \det(f_i(\alpha_j) : i \in I, j \in L) =: \det(f_I; \alpha_L).$$

For $g = \{ev(h_i)\}_{i \in E}$ in block form, the entries are

$$\Lambda(g)_{I,L} = \det(h_i(\alpha_j) : i \in I, j \in L) =: \det(h_I; \alpha_L).$$

The matrix g in block form depends on a choice of information set $A \in J(n,k)$. The matrix entry $\det(h_I; \alpha_L)$ further depends on a row index $I \in J(n,v)$ and a column index $L \in J(n,v)$.

Lemma 72. Let

$$I = \{0, 1, \ldots, t-1\} \cup \{k, k+1, \ldots, k+v-t-1\}.$$

That is, I is minimal in $J(n,v)$ such that $I \cap \{0, 1, \ldots, k-1\} = t$. For all $L \in J(n,v)$, $\det(f_I; \alpha_L) = \det(h_I; \alpha_L)$, and the upper triangular form for g and the block form for g share the same I-th row in $\Lambda(g)$.

Proof.

$$\begin{aligned} \langle\, f_i : i \in I \rangle &= \langle\, f_i : 0 \leq i \leq t-1 \,\rangle + \langle\, f_i : k \leq i \leq k+v-t-1 \,\rangle \\ &= \langle\, x^i : 0 \leq i \leq t-1 \,\rangle + \langle\, f_k x^{i-k} : k \leq i \leq k+v-t-1 \,\rangle \\ &= \langle\, h_i : i \in I \rangle. \end{aligned}$$

□

For the choice of I in the lemma, $\det(h_I; \alpha_L) = 0$ if and only if the t-th principal subresultant is zero for polynomials $q(x) = f_k(x)$ and $p(x) = \prod_{j \in L}(x - \alpha_j)$.

Example 73. Let $I = \{0, 3, 4\}$, i.e., the case $t = 1, k = 3, v = 3$ in the lemma.

$$\det(h_I; x_0, x_1, x_2) = \det \begin{pmatrix} 1 & 1 & 1 \\ q(x_0) & q(x_1) & q(x_2) \\ q(x_0)x_0 & q(x_1)x_1 & q(x_2)x_2 \end{pmatrix}.$$

Where $q(x) = f_3(x) = (x-\alpha_0)(x-\alpha_1)(x-\alpha_2)$. Let $p(x) = (x-x_0)(x-x_1)(x-x_2)$ and let $d(x) = \gcd(p(x), q(x))$. For $\deg d(x) > 1$, e.g., for $q(x_0) = q(x_1) = 0$, the matrix is singular. For $\deg d(x) = 1$, e.g., for $q(x_0) = 0, q(x_1), q(x_2) \neq 0$, the matrix is regular. For $q(x) = \sum_i q_i x^{k-i}$, $p(x) = \sum_i p_i x^{v-i}$,

$$\det(h_I; x_0, x_1, x_2) = 0 \;\Leftrightarrow\; \det \left(\begin{array}{cccc} p_0 & p_1 & p_2 & p_3 \\ 0 & p_0 & p_1 & p_2 \\ \hline q_0 & q_1 & q_2 & q_3 \\ 0 & q_0 & q_1 & q_2 \end{array} \right) = 0.$$

The right side is the first principal subresultant for $p(x)$ and $q(x)$.

Let $\Lambda(g)$ be the $\binom{n}{v} \times \binom{n}{v}$ matrix $\{\det(x^{L_1}; \alpha_{L_2})\}_{L_1, L_2 \in J(n,v)}$. As possible orderings for the rows and columns in $\Lambda(g)$ we consider the lexicographic and k-lexicographic orderings for the vertices of $J(n, v)$. By Lemma 61 and Proposition 62, the principal minors for $\Lambda(g)$ are invertible for either of the two orderings. We prove, for the case of a Vandermonde matrix g, that for each of the two orderings the principal minors of $\Lambda(g)$ factor completely into a product of binomial differences $\alpha_i - \alpha_j$.

Lemma 74. Let $L = (\ell_0, \ell_1, \ldots, \ell_{v-1}) \in J(n, v)$. The number of vertices $L' \in J(n, v)$ that are adjacent to L and that precede L in the lexicographic order is $|L| := \sum_{i=0}^{v-1}(\ell_i - i)$.

Proof. To replace $\ell_i \in L$ with $\ell_j \notin L$ such that $\ell_j < \ell_i$ there are $\ell_i - i$ choices. The total number of choices for L' is therefore $\sum_{i=0}^{v-1}(\ell_i - i)$. □

The k-lexicographic order was defined before Lemma 61. Let $E_0 = \{0, 1, \ldots, k-1\}$. Recall that $L' \leq_k L$ if $|L' \cap E_0| > |L \cap E_0|$, or, in case of equality, if $L' \leq L$ lexicographically.

Lemma 75. For adjacent vertices $L', L \in J(n, v)$,

$$L' < L \;\Leftrightarrow\; L' <_k L \;\Leftrightarrow\; |L'| < |L|.$$

Proof. Let $a = L' \backslash L$ and $b = L \backslash L'$. Clearly, $L' < L$ if and only if $a < b$ if and only if $|L'| < |L|$. Moreover, $L' <_k L$ if and only if ($a < k \leq b$ or $a < b \leq k-1$ or $k \leq a < b$) if and only if $a < b$. □

Proposition 76. Let g be a Vandermonde matrix. Let rows and columns in $\Lambda(g)$ share the same ordering in which row L' precedes row L whenever $L', L \in J(n, v)$ are adjacent and $|L'| < |L|$. Then, for every $L \in J(n, v)$, the determinant of the submatrix $\Lambda(g)_{\leq L} = \Lambda(g)_{L_1, L_2 \leq L}$ factors as a product of binomial differences $\alpha_i - \alpha_j$, $i \neq j$.

Proof. We prove by induction to L that $d_{\leq L} = \det \Lambda(g)_{\leq L}$ factors completely, as polynomial in $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, into factors $\alpha_i - \alpha_j$. In general factors appear with multiplicities. For the base case $L_0 = \{0, \ldots, v-1\}$, the scalar $\Lambda(g)_{\leq L_0} = \det(x^{L_0}; \alpha_{L_0})$ is a Vandermonde determinant of degree $d_0 = \sum_{i=0}^{v-1} i$. For the inductive case, observe that each new column L contributes two types of linear factors to $d_{\leq L}$. There is a first contribution of d_0 factors because every entry in column L is divisible by $\det(x^{L_0}; \alpha_L)$ of degree d_0. Secondly, there is a contribution of one linear factor $\alpha_i - \alpha_j$ for each $L' < L$ that is adjacent to L. By Lemma 74 and Lemma 75 the number of such L' is $|L|$. As $\Lambda(g)_{<L}$ is enlarged to $\Lambda(g)_{\leq L}$, both the degree of the determinant and its number of linear factors increases by $|L| + d_0$. □

The proposition applies to all principal minors of the matrix $\Lambda(g)$. In particular, for the full matrix $\Lambda(g)$ it follows that $\det \Lambda(g) = (\det g)^m$, for $m = \binom{n-1}{v-1}$.

Let $I_0 = (0, 1, \ldots, v-1)$ and let $\pi_0 = (\det(x^{I_0}; \alpha_L) : L \in J(n, v))$ be a vector of $v \times v$ Vandermonde determinants. Replacing the determinant $\det(x^I; \alpha_L)$ in a generator $\pi(M) = (\det(x^I; \alpha_L) : L \in J(n, v))$ for a Johnson graph code with its Schur polynomial $\det(x^I; \alpha_L) / \det(x^{I_0}; \alpha_L)$ gives an equivalent set of rescaled generating vectors $\pi(M)\Delta_0$, for the diagonal matrix

$\Delta_0 = \text{diag}\,(\pi_0)^{-1}$. This replaces the code with an equivalent code with the same properties but normalized such that the leading generator for the code is the allone vector.

Schur polynomials are homogeneous symmetric polynomials with unique expressions on a basis of monomial symmetric functions [24, Corollary 7.10.6]. We point out and illustrate with an example that the triangular matrix that transforms Schur polynomials into monomial symmetric functions is in general not compatible with the k-lexicographic order that we use for the generators of a Johnson graph code. Thus, in general, replacing Schur functions with the leading monomial symmetric functions in their expansion will change the code. It can be shown however that these different codes are again Johnson graph codes.

Example 77. Consider the Johnson graph code $JGC(6,3,3,1)$. Let $A = \{0,1,2\}$. The Schur polynomials of degree 4 and 5, and their expansion as sums of monomial symmetric functions are

$$\begin{array}{rrlllllll}
I=\{0,2,5\} & r=1 & s_{310} = & m_{310} & + & m_{220} & + & 2m_{211}, \\
\{0,3,4\} & 2 & s_{220} = & & & m_{220} & + & m_{211}, \\
\{1,2,4\} & 1 & s_{211} = & & & & & m_{211}. \\
I=\{0,3,5\} & r=2 & s_{320} = & m_{320} & + & m_{311} & + & 2m_{221}, \\
\{1,2,5\} & 1 & s_{311} = & & & m_{311} & + & m_{221}, \\
\{1,3,4\} & 2 & s_{221} = & & & & & m_{221}.
\end{array}$$

The Schur polynomials $s_{310}, s_{211}, s_{311}$ have $I = 025, 124, 125 \in S_1(012)$ and the corresponding generators $\pi(M)$ are generators for $JGC(6,3,3,1)$. They cannot be expressed as linear combinations of the corresponding monomial symmetric functions $m_{310}, m_{211}, m_{311}$. The triangular transformation that relates Schur polynomials and monomial symmetric functions of the same degree in general involves Schur polynomials from different shells.

We mention three types of codes that share common properties with Johnson graph codes: Grassmann codes and Schubert codes, Polynomial Chinese remainder codes, and product matrix MSR codes.

For $v \leq n$, rational points on the Grassmann variety $G_{v,n}$ over a finite field F correspond to v dimensional F-linear subspaces of F^n. The variety has a smooth embedding, using Plücker coordinates, into projective space $\mathbb{P}^{N-1}$, for

$N = \binom{n}{v}$. The Grassmann code is the row space of the $N \times K$ matrix whose columns are the images in $\mathbb{P}^{N-1}$ of rational points $P_1, P_2, \ldots, P_K \in G_{v,n}$. Over a field F of size q, $K \leq \#G_{v,n}(F) = \binom{n}{v}_q$. Let $g = \{g_1, \ldots, g_n\}$ be a basis for F^n. For $\alpha \in J(n, v)$, write the elements in α in increasing order $1 \leq \alpha_1 < \cdots < \alpha_v \leq n$. The Schubert variety $\Omega_\alpha \subset G_{v,n}$ is the subvariety of subspaces W whose intersection with the flag generated by g reaches dimension i after at most α_i steps.

$$\Omega_\alpha = \{\, W \in G_{v,n} \,:\, \dim(W \cap \langle g_1, \ldots, g_{\alpha_i} \rangle) \geq i, \text{ for } i = 1, \ldots, v. \,\}.$$

A Schubert code is defined by projecting the $N \times K$ matrix onto a $N \times K_\alpha$ submatrix with columns in Ω_α. In general this reduces the rank of the matrix to k_α, and yields a Schubert code of dimension k_α and length K_α. With hindsight, our construction of Johnson graph codes matches the column space of the $N \times K_\alpha$ matrix, for the maximal α in the Bruhat order with $\alpha_t = k$. The Johnson graph code, as row space of a $K_\alpha \times N$ matrix of rank k_α, has dimension k_α and length N. In our description, we find generators for the code by taking the top $k_\alpha = |B_r(\{0, 1, \ldots, k-1\}|$ rows in a $N \times N$ matrix $\Lambda(g)$. With the above we can interpret information sets for the Johnson graph code as projections of the Plücker embedding of a Schubert variety that separate points.

We give a different construction for codes $J(n, v, v, 1)$. The proof for the general case follows after the example.

Example 78. Example 70 describes a code $C = JGC(5, 2, 2, 1)$ and the equivalent code $C' = JGC(5, 3, 3, 1)$. The first code uses an embedding of the pair $\{x, y\} \subset \{\alpha_0, \alpha_1, \ldots, \alpha_4\}$ with Plücker coordinates $\det_I(x, y)$, $I \in \{01, 02, 03, 04, 12, 13, 14\}$. The second code uses an embedding of the triplet $\{x, y, z\} \subset \{\alpha_0, \alpha_1, \ldots, \alpha_4\}$ with Plücker coordinates $\det_I(x, y, z)$, $I \in \{012, 013, 014, 023, 024, 123, 124\}$. A different embedding that also yields a code $JGC(5, 2, 2, 1)$ is as vector of coefficients for the polynomial

$$(1 + xt + x^2t^2 + x^3t^3)(1 + yt + y^2t^2 + y^3t^3) \qquad (n = 5, v = 2).$$

Similarly the coefficients for the polynomial

$$(1 + xt + x^2t^2)(1 + yt + y^2t^2)(1 + zt + z^2t^2) \qquad (n = 5, v = 3).$$

give an embedding for $\{x, y, z\}$ that yields a code $JGC(5, 3, 3, 1)$.

Proposition 79. Let $v \leq n$. Let $f_0, f_1, \ldots, f_{n-1} \in F[x]$ be n polynomials of degree $n - v$, such that any two polynomials are relative prime and any $n - v + 1$ polynomials are linearly independent. For each $L \in J(n, v)$, let $f_L = \prod_{i \in L} f_i$. Then $\deg f_L = v(n - v)$. For any given $A \in J(n, v)$, the collection $\{f_L : |L \cap A| \geq v - 1\}$ forms a basis for $F[x]_{\leq v(n-v)}$.

Proof. The size of the collection $\{f_L\}$ is $1+v(n-v)$, which is the dimension of $F[x]_{\leq v(n-v)}$. It suffices to prove that the f_L in the collection span $F[x]_{\leq v(n-v)}$. For any $i \in A$, the collection contains $n-v+1$ multiples $(f_A/f_i) \cdot f_j$, for $j = i$ or $j \notin A$. Since the f_j span $F[x]_{\leq n-v}$, the collection contains all multiples of f_A/f_i, for all $i \in A$. Since f_j and f_i are relative prime, the collection spans all multiples of $f_A/f_i f_j$, for all pairs $i, j \in A$. With induction it follows that the collection spans all multiples of 1, i.e., spans $F[x]_{\leq v(n-v)}$. □

Corollary 80. A special case of the proposition is $f_i = 1 + \alpha_i x + \cdots + \alpha_i^{n-v} x^{n-v}$, for distinct $\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \in F$, such that $\gcd(n-v+1, |F|-1) = 1$.

A special case of a Johnson graph code is the following. Let $\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \in F$ be distinct field elements. Let S be a symmetric matrix of size $k - 1 \times k - 1$, with associated symmetric form

$$f(x, y) = (1, x, \ldots, x^{k-2}) S (1, y, \ldots, y^{k-2})^T.$$

Label vertices $\{i, j\} \in J(n, 2)$ with $f(\alpha_i, \alpha_j)$. Then S, and thus $f(\alpha_i, \alpha_j)$ for all $\{i, j\} \in J(n, 2)$, is uniquely determined by the set of values $f(\alpha_i, \alpha_j)$, $i, j \in A$, for any k-subset A of $\{0, 1, \ldots, n - 1\}$. Thus the code is of type $JGC(n, 2, k, 0)$. In [22], two copies of the code are combined to form an MSR regenerating code. Let the second copy be defined with a matrix T and symmetric form $g(x, y)$. For each $i \in \{0, 1, \ldots, n - 1\}$, node i stores the polynomial $f(\alpha_i, y) + \alpha_i g(\alpha_i, y)$, which is a polynomial of degree $k - 2$ in the single variable y. When data is collected from k nodes A, any two accessed nodes $i, j \in A$ can recover $f(\alpha_i, \alpha_j)$ and $g(\alpha_i, \alpha_j)$ from $f(\alpha_i, \alpha_j) + \alpha_i g(\alpha_i, \alpha_j)$ and $f(\alpha_j, \alpha_i) + \alpha_j g(\alpha_j, \alpha_i)$, that is together they can decouple their stored values. With the decoupled values, the matrices S and T can be recovered as before. In this construction, the Johnson graph code is used as inner code

and the coupling as outer code. In the next section, Johnson graph codes are used as outer codes and layered codes as inner codes.

3.8 Concatenated Layered Codes

For parameters $(n, k, d) = (n, n-1, n-1)$, layered codes provide optimal trade-off between storage overhead and repair bandwidth. More precisely, the convex region of achievable combinations of storage overhead versus bandwidth has a piece-wise linear boundary with layered codes at the corner points (Figure 1.1). One of the main applications of Johnson graph codes is to concatenate different layered codes into new codes that offer data collection from $k < n-1$ nodes.

Definition 81. Given a $(n, n-1, n-1)$ layered code with layer size v and upon contacting k disks, we call a layer L **sufficiently-accessed** if the content of L can be recovered directly from the accessed symbols in the k disks and the single parity check on the symbols in L. A layer that can not be recovered directly in this way is called **under-accessed**.

In particular, upon contacting the k disks $A \subset \{0, 1, \ldots, n-1\}$, a layer L of size v is sufficiently-accessed if $|L \cap A| \geq v - 1$ and under-accessed if $|L \cap A| < v - 1$. A layer L is sufficiently-accessed if and only if $L \in B_s(A)$, where $s = 0$ if $v = k + 1$ and $s = 1$ if $v \leq k$.

For given $k < n-1$, and for $v \leq k+1$, the process of modifying a layered code of type $(n, n-1, n-1)$ with layer size v into a regenerating code of type $(n, k, n-1)$ will be carried out recursively. Assume that the modification has been completed for layered codes with layer size $w < v$. When a modified code with layer size w is contacted in k disks those disks will have sufficient added data to completely recover all data in all layers. Since $w < v \leq k+1$, upon contacting k disks, $\binom{k}{w}$ layers of size w will have all their w symbols on the k disks. For those layers the parity check sum for the symbols in the layer is redundant.

Let $J(n-w, v-w)$ be the subgraph of $J(n,v)$ that is spanned by the layers that contain a given sublayer L_w of size w. Label each vertex L in the graph with a recovery symbol for layer L that it can retrieve in case it is insufficiently-accessed. The task of the layer L_w is to store all the recovery symbols for all layers L that contain L_w. In case of access in k disks that contain L_w, layers L such that $L \supset L_w$ and $|L \cap A| \geq v-1$ need no recovery symbol. These are precisely the layers in $B_s(A) \subset J(n-w, v-w, k-w)$. Thus a Johnson graph code can be used to obtain the remaining $|B_s(A)^c|$ symbols from the known recovery symbols and from $|B_s(A)^c|$ stored parities. We use $|B_s(A)^c|$ copies of a modified $(n, n-1, n-1)$ layered code with layer size w. And for each version of the layer L_w we replace the zero parity check sum with a sum that adds to a parity.

Lemma 82. A layered code of type $(n, n-1, n-1)$ becomes a $(n, k, n-1)$ regenerating code by adding content to each disk such that

1. the modified content on any k disks gives each layer access to at least $v-1$ symbols.

2. the new content on any given disk adds more information from layers that already store information on that disk but no information from other layers.

Proof. For all the sufficiently-accessed layers we have the $v-r$ accessed disks can already provide $v-1$ symbols from stored information, and hence using the parity check on the layer, we can recover the whole layer.

For the under-accessed layers we do not have enough stored information in the layered code, but if we can modify the layers to have the the $v-r$ accessed disks to provide $v-1$ data symbols from the same layer, the parity check on the layer will allow us to recover these layers. So we recover the whole file by contacting k disks.

For repair, we have that in the layered code disk i gets help in the form of all the other symbols in layers in which disk i participates. If for each $1 \leq i \leq n$ the changes in the content of disk i only depends on the layers in which disk i participates, repairing the modified part would require only a subset of the repair information in the original layered code.

Since we have data collection from k disks without changing the repair information, we get a $(n, k, n-1)$ regenerating code with exact repair. $\square$

3.8.1 Cascade layered codes

Cascade codes are introduced in [20], [21] as concatenated versions of determinant codes. Determinant codes have the same properties as layered codes but have no restriction on the number of nodes n. We describe cascade codes for the case of layered codes as a special type of improved layered code. A layered code with parameters n and v is concatenated with other layered codes defined with the same n but with smaller v. Codes with a high v have a high ratio of information symbols to parity symbols and suffer from not retrieving all information symbols when collecting data from fewer than k nodes. Codes with a low v have a low ratio of information symbols to parity symbols, which benefits efficient disk repair but leads to downloading redundant parities instead of information symbols when collecting data. The idea is to link the two types of codes so that downloading data from low v codes can provide information that is missing after downloading data from high v codes. The situation resembles optimizing checks in an ldpc code using density evaluation.

Lemma 83. In a $J(n, v)$ Johnson graph, the subgraph spanned by vertices containing a subset $I \subset \{0, 1, \ldots, n-1\}$ form a $J(n - |I|, v - |I|)$ Johnson graph.

Lemma 84. For $k \leq n - 1$, given an access set A with $|A| = k$, a layer $L_v \in S_r \subset J(n, v)$ of the $(n, k, n-1)$ layered code with layer size v determines uniquely a layer $L_j \in S_0 \subset J(n, j)$ of the $(n, k, n-1)$ layered code with layer size $j = |L \cap A|$. We have $j = v - 1 - r$ for $v > k$ and $j = v - r$ for $v \leq k$.

That is, we have a well defined map $f : S_r \subset J(n, v) \to S_0 \subset J(n, j)$, given by $f(L_v) = L_v \cap A$.

Moreover, for any layer $L \in S_i \subset J(n, j)$ with $i > 0$, we have $f^{-1}(L) \in S_{i'} \subset J(n, v)$ with $i' < r$.

Construction 85. Given with a $(n, n-1, n-1)$ layered code with layer size $1 \leq v \leq n$ and $1 \leq k \leq n - 1$, and $A \subset \{0, 1, \ldots, n\}$ with $|A| = k$ labeling the set of accessed disks, for $r \in R_u$ and for each layer $L \in S_r$ with $j = |L \cap A|$ accessed disks, by Lemma 83, we have an induced Johnson graph $J(n-j, v-j)$ with vertices consist of size j subsets of $\{0, 1, \ldots, n\} \setminus (L \cup A)$. Each vertex in this graph corresponds to a layer with exactly j symbols from the set $L \cup A$. For each such vertex we can generate a parity such that it

forms a $[v-j+1, v-j]$ code with the $v-j$ symbols in the layer with labels not in $L \cup A$.

We then can use the Johnson graph code $JGC(n-j, v-j, k-j, v-j-2)$ to encode the above parities and generate $r\binom{\ell+1}{v-j}$ new parities, for the shell $S_r \subset J(n-j, j)$, where $\ell = n-k-1$. For each such layer L, we get a unique layer $f(L)$, as in Lemma 84. We modify $f(L)$ of the $(n, n-1, n-1)$ layered code with layer size j so that instead of a single parity check, we have a $[j+1, j]$ code on the $j+1$ symbols consists of the symbols in the layer and this parity. This way we use $r\binom{\ell+1}{v-j}$ copies of the $(n, n-1, n-1)$ layered code with layer size j.

We proceed from codes of bigger layer size down to cover all layers of each helper code of smaller layer size.

Proposition 86. The construction above gives a $(n, k, n-1)$ regenerating code with 1 copy of layered code with top layers size v_0 and concatenation of a_j copies of layered of with layer size $0 < j < v_0$, $j = |L \cup A|$, satisfying the recurrence relation:

$$a_j = \sum_{v>j}^{v_0} a_v j \binom{l+1}{j+1}.$$

Proof. By Lemma 82 we need to show that layers with $r \in R_u$ to provide $v-1$ data symbols from the same layer for each of the code we use in the concatenation. We first show that the code with layer size v_0 can be recovered. By Lemma 84, we have that a size v_0 layer $L \in S_r$ with $j_0 = v_0 - 1 - r$ accessed disks determines uniquely a layer in S_0 of the $(n, k, n-1)$ layered code with layer size j_0. In a data collection scenario, we have the size j_0 layer is completely accessed and with the $[j_0+1, j_0]$ code we put on it in the above construction, we can recover the parity generated by the $JGC(n-j_0, j_0, k-j_0, v-j-2)$ code. We have altogether $r\binom{\ell+1}{v-j_0}$ layers in $S_r \subset J(n, v)$, and each can be recovered with the corresponding layer in the size j_0 providing the parity. We can recover parities from all shells sequentially starting from $r = 0$ by construction of the Johnson graph code.

We now recover codes with layer size $j_0 = v_0 - 1 - r < k$ with increasing radii. The shell S_0 consists of layers that are completely accessed. The shell S_1 can be recovered from the parities computed from the ball $B_r(A) \subset J(n, v_0)$ in the code with layer size v_0, where A is the set of disks accessed.

The rest of the shells with bigger radii will receive helping parities generated by the Johnson graph code $JGC(n, j_0, k, 1)$ from codes of smaller layer size and be recovered.

By Lemma 82, we get a $(n, k, d = n-1)$ regenerating code with exact repair, as long as we can show that the cascade process terminates and the numbers of copies of layered code of different layer sizes are all finite. But this is true since the concatenated layer of size j_0 is getting strictly smaller in the range of r above, and each concatenation only result in using finitely many codes of smaller layers.

For the recurrence relation, we see that a code with layer size j is used by a layer $L_v \in S_r$ in a code with layer size $v > j$ when $j = |L_v \cap A|$ and it is used $j\binom{l+1}{j+1}$ times. □

It can be verified that the recurrence relation has the following closed form solution presented compactly as a generating function.

Proposition 87. Let

$$f_\ell(t) = \frac{1}{(1-\ell t)(1+t)^\ell} = \sum_{i \geq 0} a_i t^i.$$

After concatenation with a_i layered codes $L_{v-i}(n)$, for $0 < i < v$, and suitable linkage of layers in the concatenated code, the layered code $L_v(n)$ of type $(n, n-1, n-1)$ improves to a code of type $(n, n-1-\ell, n-1)$. So that data for the concatenated code is available from any $n-1-\ell$ nodes.

Note that the coefficients in the concatenation depend on ℓ but not on n or v.

Corollary 88. After concatenation and linkage of layers, the code $L_v(n)$ improves to an MSR code of type $(n, k = v-1, d = n-1)$ with

$$\alpha = (d-k+1)^k, \quad \beta = (d-k+1)^{k-1}.$$

Proof. Observe that α is the coefficient of $f_\ell(t)(1+t)^{n-1}$ at t^{v-1} and β is the coefficient of $f_\ell(t)(1+t)^{n-2}$ at t^{v-2}. □

For the same parameters $(n, k, d) = (8, 4, 7), v = 5$, the cascade construction focuses on layers and generate parities for each layer that are cascaded to the next level.

Example 89. The cascade construction for a $(8, 4, 7)$ code uses $(8, 7, 7)$ layered codes with layers of size $5, 3, 2$, and 1. Since each layered code has complete symmetry among the layers, WLOG we can illustrate how the smaller layers are modified to provide help by taking an arbitrary data collection scenario. Here we take $A = \{0, 1, 2, 3\}$ to be the label of the disks accessed.

$(v = 5)$

	nodes		distance
#	$0, 1, 2, 3$	$4, 5, 6, 7$	r
4	$a***$	$b---$	0
24	$-***$	$c*--$	1
24	$--**$	$***-$	2
4	$---*$	$****$	3

$(v = 3)$

	nodes		distance
#	$0, 1, 2, 3$	$4, 5, 6, 7$	r
4	$-***$	$----$	0
24	$--**$	$*---$	1
24	$---*$	$**--$	2
4	$----$	$***-$	3

$(v = 2)$

	nodes		distance
#	$0, 1, 2, 3$	$4, 5, 6, 7$	r
6	$--**$	$----$	0
16	$---*$	$*---$	1
6	$----$	$**--$	2

$(v = 1)$

	nodes		distance
#	$0, 1, 2, 3$	$4, 5, 6, 7$	r
4	$---*$	$----$	0
4	$----$	$*---$	1

The $v = 5$ code is used once. The $v = 3$ code is used six times, its 4 layers with $r = 0$ help the 24 layers in the $v = 5$ code with $r = 1$.

We see that the layer (123) in the $v = 3$ code is one of the four completely

accessed in this data collection scenario, we will modify it to help one of the six layers in the $v = 5$ code with $r = 1$ with labels containing $\{1, 2, 3\}$. Looking at the subgraph in the Johnson graph $J(8, 5)$ spanned by vertices with labels containing $\{1, 2, 3\}$, we get a $J(5, 2)$ Johnson graph, having the following shell structure:

$\{1, 2, 3\}$

	nodes		distance
#	0	$4, 5, 6, 7$	r
4	a	$b - --$	0
6	$-$	$c * --$	1

We generate one parity for each size two layer. For any such layer L, we have $f^{-1}(L)$ is a layer of size $v = 5$, where f is the map in Lemma 84. The layer $f^{-1}(L)$ has five symbols with a single parity check, and we can make it a $[6, 4]$ code to get one parity for the layer L. Using a $JGC(5, 2, 1, 0)$ code, we can encode these parities and generate six new parities so that together with the top four layers, which are available from the $v = 5$ code in this access scenario (illustrated by a and b, corresponding to the a and b in the $v = 5$ table), we can recover the bottom six layers. For each copy of the $v = 3$ code, we use one of the six parities from this $JGC(5, 2, 1, 0)$ code by putting a $[4, 3]$ MDS code on the three symbols in the layer (123) plus the parity. So once we access the layer (123) fully, we get the parity and with the $JGC(5, 2, 1, 0)$ code, we can recover the $r = 1$ layers in the code $v = 5$ containing $\{1, 2, 3\}$.

The $v = 2$ code is used eight times, its 6 layers with $r = 0$ help the 24 layers layers in the $v = 5$ code with $r = 2$, each needing 2 extra symbols.

We see that the layer (23) in the $v = 2$ code is one of the six completely accessed in this data collection scenario. Looking at the subgraph in the Johnson graph $J(8, 5)$ spanned by vertices with labels containing $\{2, 3\}$, we get a $J(6, 3)$ Johnson graph having the following shell structure:

		nodes		distance
	#	$0,1$	$4,5,6,7$	r
$\{2,3\}$	4	$**$	$*---$	0
	12	$-*$	$**--$	1
	4	$--$	$***-$	2

We generate one parity for each size three layer. Using a $JGC(6,3,2,1)$ code, we can encode these parities and generate four new parities so that together with the top $4+12=16$ layers, which are available jointly from the $v=5$ code and the helping $v=3$ code in this access scenario, we can recover the bottom four layers. For each copy of the $v=2$ code, we use one of the eight parities from the $JGC(6,3,2,1)$ code by putting a $[3,2]$ MDS code on the two symbols in the layer (23) plus the parity.

Finally the $v=1$ code is used 3 times, its 4 layers with $r=0$ to help the 4 layers layers in the $v=5$ code with $r=3$, each needing 3 extra symbols.

We see that the layer (3) in the $v=1$ code is one of the six completely accessed in this data collection scenario. Looking at the subgraph in the Johnson graph $J(8,5)$ spanned by vertices with labels containing $\{3\}$, we get a $J(7,4)$ Johnson graph having the following shell structure:

		nodes		distance
	#	$0,1,2$	$4,5,6,7$	r
$\{3\}$	4	$***$	$*---$	0
	18	$-**$	$**--$	1
	12	$--*$	$***-$	2
	1	$---$	$****$	3

We generate one parity for each size four layer. Using a $JGC(7,4,3,2)$ code, we can encode these parities and generate a new parities so that together with the top $4+18+12=34$ layers, which are available jointly from the $v=5$ code and the helping $v=3$, $v=2$ codes in this access scenario, we can recover the last layer on the bottom. For each copy of the $v=2$ code, we use the parity from the $JGC(7,4,3,2)$ code by putting a $[2,1]$ MDS code on the symbols in the layer (3) plus the parity.

The $v = 1$ code is used another six times for each of the six $v = 3$ code, its 4 layers with $r = 0$ to help the 24 layers in the $v = 3$ code with $r = 2$.

$\{3\}$

	nodes		distance
#	$0, 1, 2$	$4, 5, 6, 7$	r
3	$-**$	$----$	0
12	$--*$	$*---$	1
6	$---$	$**--$	2

We generate one parity for each size two layer. Using a $JGC(7, 2, 3, 1)$ code, we can encode these parities and generate six new parities so that together with the top $3 + 12 = 15$ layers, which are available from the $v = 3$, code in this access scenario, we can recover the last six layer on the bottom. For each copy of the $v = 1$ code, we use one of the six parities from the $JGC(7, 4, 3, 2)$ code by putting a $[2, 1]$ MDS code on the symbols in the layer (3) plus the parity.

The $v = 1$ code is used $39 = 3 + 6 \times 6$ times in total.

So overall we have $M = 56 \times 4 + 52 \times 2 \times 6 + 22 \times 1 \times 8 = 1024 = 4^5, \alpha = 35 + 21 \times 6 + 7 \times 8 + 39 = 256 = 4^4, \beta = 20 + 6 \times 6 + 1 \times 8 = 64 = 4^3$. Note that when we count M and α we exclude the last row from $v = 3, 2, 1$, as they are not accessible in a data collection scenario. This is accommodated by shortening the code before we put them into layers.

Data collection works in rounds. Upon accessing 4 disks, in the zero-th round the layers that can provide $v - 1$ symbols can be recovered by their parity checks. The recovered layers are marked by a star and the ones needing help are marked by a dash in the schematic below. In the subsequent rounds we put $v.r$ for the help from the S_r shell in the code of layer size v.

In the first round we compute from the obtained information the parities we need in the first round. The $v = 5$ code gets help from the S_0 shell of codes of smaller layer size and is completely recovered. In the second round, the S_1 layers of the $v = 3$ code recover from getting their parities computed from the $v = 5$ code. The S_2 layers of the $v = 3$ code recover from getting their parities computed from the $v = 5$ code, and getting another parities from the $v = 1$ code. The S_1 layers of the $v = 2$ code also recover from

getting their parities computed from the $v = 5$ code.

	nodes		recovery rounds		
#	$0,1,2,3$	$4,5,6,7$	0	1	2
4	$****$	$*---$	$*$	$*$	$*$
24	$-***$	$**--$	$-$	3.0	$*$
24	$--**$	$***-$	$-$	2.0	$*$
4	$---*$	$****$	$-$	1.0	$*$
4	$-***$	$----$	$*$	$*$	$*$
24	$--**$	$*---$	$-$	5.0/3.0	$*$
24	$---*$	$**--$	$-$	$-$	1.0/3.0/3.1
4	$----$	$***-$	$*$	$*$	$*$
6	$--**$	$----$	$*$	$*$	$*$
16	$---*$	$*---$	$-$	5.0/5.1/2.0	$*$
6	$----$	$**--$	$*$	$*$	$*$
4	$---*$	$----$	$*$	$*$	$*$
4	$----$	$*---$	$*$	$*$	$*$

3.8.2 Improved Layered Construction

In the improved layered construction, we focus on a single disk and use the Johnson graph to generate all the necessary backup information symbols for that disk.

Construction 90. Given with a $(n, n-1, n-1)$ layered code with layer size $v \leq n$ and $k \leq n-1$, we construct backup information for individual disks. Let A be the k disks accessed. By Lemma 82, the layers containing disk i form a $J(n-1, k-1)$ Johnson graph. We have a $(n-1, n-2, n-2)$ layered code with layer size $v-1$ and layers corresponding to the vertices of this $J(n-1, k-1)$ Johnson graph. In this $(n-1, n-2, n-2)$ layered code, for all $r \in R_u$ we can get the first $m = (v-2)!$ pre-parities and the next $m_r = m(v-1)(\frac{1}{j} - \frac{1}{j+1})$ pre-parities for each layer in $B_{r-1}(A)$, where $j = |L \cap A|$ for any $L \in B_r(A)$. We take the pre-parities, m_r from each layer in $B_{r-1}(A)$ and use the Johnson graph code $JGC(n-1, v-1, k-1, r-1)$ times to generate $m_r|B_r|$ parities for the next shell S_r.

Proposition 91. With sub-packetization level $m = (v-2)!$, the above construction gives a $(n, k, d = n-1)$ regenerating code with exact repair.

Proof. For each $r \in R_u$, we have that a layer L in the shell S_r need $(v-1)m$ symbols from the $j = |L \cap A|$ accessed disks. For such a layer, each of the j accessed disks can use its parities from the $JGC(n-1, v-1, k-1, t-1)$ codes for each $t \leq r$ to help recover $\sum_i^r m_r = m(v-1)(\frac{1}{j} - \frac{1}{v-11}) = \frac{(v-1-j)m}{j}$ pre-parities in addition to its stored m symbols. By Theorem 39, these $j(\frac{(v-1-j)m}{j} + m) = (v-1)m$ pre-parities are enough to recover the layer. By Lemma 82, this improved layered code is a $(n, k, d = n-1)$ regenerating code with exact repair. □

Remark 92. The sub-packetization level m above is taken to be $(v-2)!$ but can be made smaller in specific examples, often we can get $m \leq lcm(1, 2, \ldots, v-2)$.

Example 93. Consider the following example with $(n, k, d) = (8, 4, 7)$ and $v = 5$. We demonstrate in this example how to generate parities for a given disk, WLOG we focus on disk 7. Let $\{0, 1, 2, 7\}$ be the disks chosen for downloading. The 35 layers containing node 7 are of the following type. In other words, the neighborhoods $B_r(\{0, 1, 2\})$ in the Johnson graph $J(7, 4)$ are as follows:

	nodes			distance
#	$0, 1, 2$	$3, 4, 5, 6$	7	r
4	$***$	$*---$	$*$	0
18	$-**$	$**--$	$*$	1
12	$--*$	$***-$	$*$	2
1	$---$	$****$	$*$	3

We use a sub-packetization of six, where instead of four information symbols in a layer, we have 24. For the top row, we get four out of five disks in those layers, and can get three symbols from each of the four disks to recover those layers completely. The second row we need the three accessed disks each provide eight symbols, so six stored symbols plus two stored parities. The next row we need the two accessed disks to each provide twelve symbols, so six stored symbols plus six stored parities. The last row we need disk 7 to provide all 24 symbols, so six stored symbols plus eighteen stored parities. All together disk 7 is responsible for $18 \cdot 2 + 12 \cdot 6 + 1 \cdot 18 = 126$ parities.

We use two copies of the $JG(7,4,4,0)$ code to generate $2(35-4)=62$ parities for disk 7, which contribute to the first two parities for all 31 layers outside of $B_0(\{0,1,2\})$. We then use the $JG(7,4,4,1)$ code four times to generate $4\cdot(35-4-18)=52$ parities for disk 7, which contribute to the next two parities for the 13 layers outside of $B_1(\{0,1,2\})$. Finally, we use the $JG(7,4,4,2)$ code twelve times to generate $12\cdot(35-4-18-12)=12$ parities for disk 7, which contribute to the last two parities for the 1 layer outside of $B_2(\{0,1,2\})$. This way we get all $62+52+12=126$ parities we need.

This construction gives $M=56\cdot 4\cdot 6=1344$, $\alpha=35\cdot 6+126=336$, $\beta=20\cdot 6=120$. Note that as we remarked above, the sub-packetization level in this example can be made smaller, in particular three will work the same way as six.

CHAPTER 4

CODES FOR REGENERATION, PART II

4.1 Graphical Representation of MSR Codes

Here we focus on the MSR point with $d = n - 1$. We present graphical representations of codes with parameters $((n, k, d), (\alpha, \beta)) = ((qt, q(t-1), qt-1), (q^t, q^{t-1}))$ for small field size. The parameters are the same as the ones found in [18] and [23]. Our codes also have the optimal access property as in [23]. Our introduction of the factor graph approach opens up ways to generalizations and new code constructions.

Recall that an $(n, k, n-1)$ $n \times \alpha$ array code with $M = k\alpha$ data symbols is MSR if the code is row-wise MDS (any $k \times \alpha$ submatrix is a data matrix), and every single erased row can be repaired using no more than $\alpha/(n-k)$ symbols from each of the other rows. Here we aim to construct a $((n, k, d), (\alpha, \beta))$ regenerating code at the MSR point. For positive integers $q \geq 1, t \geq 2$, we obtain codes with $((n, k, d), (\alpha, \beta)) = ((qt, q(t-1), qt-1), (q^t, q^{t-1}))$.

The construction is based on a factor graph structure endowed on a q-ary Hamming graph with dimension t.

Definition 94. For positive integers $q \geq 1, t \geq 2$, the q-ary **Hamming graph** with dimension t, $H(q, t)$ has vertex set $V(q, t) = A_q^t$, where $A_q = \{0, 1, \ldots, q-1\}$. The edge set $E(q, t)$ is defined as

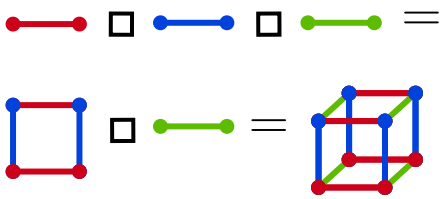
$$E(q, t) = \{(v_1, v_2) \mid d(v_1, v_2) = 1\},$$

where $d : A_q^t \times A_q^t \to \mathbb{N}$ is the Hamming metric.

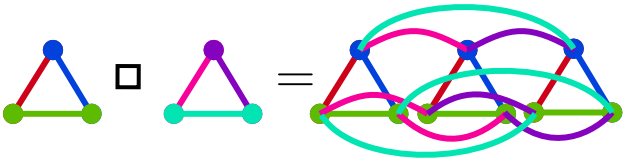
Theorem 95. The q-ary Hamming graph with linking parameter t, $H(q, t)$, is the t-fold graph Cartesian product of the complete graph on q vertices, K_q. Using $\square$ to denote the graph Cartesian product, we have $H(q, t) = K_q \square \ldots \square K_q = K_q^{\square t}$.

Example 96. When $t = q = 2$, we get the product of two lines, which is a square.

When $q = 2, t = 3$, we get the cube graph as the product of the square graph and another line



When $q = 3, t = 2$, we get the product of two triangle graphs.



4.2 Code Construction

4.2.1 Defining Checks

We construct the code by putting a factor graph structure on the q-ary Hamming graph with dimension t. Given any $M = k\alpha = q(t-1)q^t = q^{t+1}(t-1)$ information symbols in the original message, we describe a way to encode them into $n\alpha = qtq^t = q^{t+1}t$ encoded symbols. The encoding process is done by first generating auxiliary symbols, also known as hidden variables in factor graphs, and then using them to generate the rest of the encoded symbols.

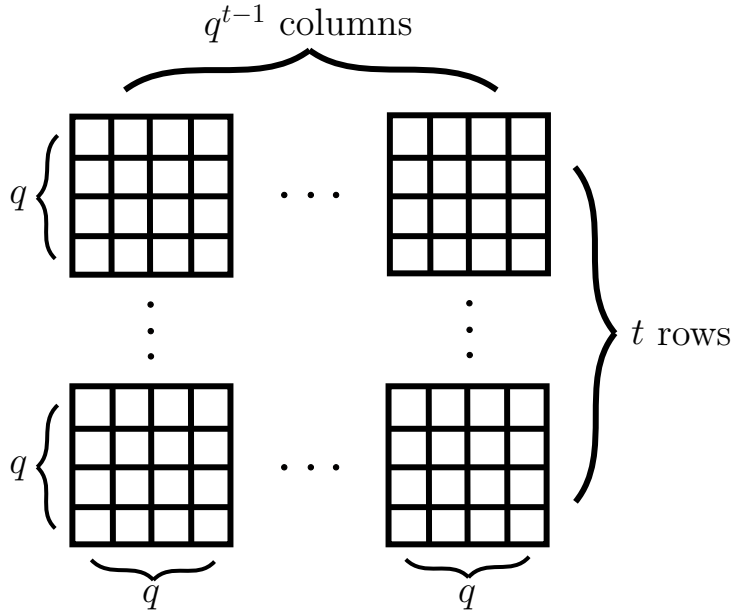
Observe that the q-ary Hamming graph with dimension t, $H(q,t)$ has q^t vertices and $q^t(q-1)t/2$ edges. We attach two variables to each edge and t variables to each vertex. This way we get $q^t(q-1)t$ edge variables and $q^t t$ vertex variables. Together they consist of all $n\alpha = q^{t+1}t$ variables we want for the encoded message.

Definition 97. For each edge (i,j) in $H(q,t)$, $i,j \in V(q,t) = A_q^t$, we define $e(i,j)$, $e(j,i)$ to be the associated **edge variables**. For each vertex $j \in V(q,t)$, we define $v(\ell,j), \ell = 1,2,\ldots,t$, to be the associated **vertex variables**.

These variables are naturally partitioned into $n = qt$ groups. A group corresponds to fixing one of the t positions in $i \in V(q,t) = A_q^t$ and demanding that the position is occupied by one of the q symbols. We make it precise as follows.

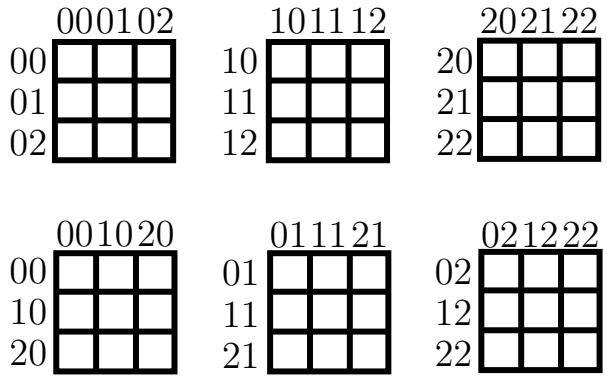
Definition 98. Given $0 \leq m \leq qt-1$ we can write $m = (t_m - 1)q + q_m, 0 \leq q_m < q, 1 \leq t_m \leq t$. The unique pair (t_m, q_m) defines a set V_m of vertex labels $i \in V(q,t)$ with the $(t_m)^{\text{th}}$ position from the right being q_m. Disk $m, 1 \leq m \leq qt$ is defined as the collection of variables $\{e(i,j), v(\ell,j) | j \in V(t_m, q_m)\}$. As a vector $\mathbf{d}_m$, we order the variables $\{e(i,j), v(\ell,j)\}$ with the lexicographical on (i,j) from $e(i,j)$ and (j,j) on $v(\ell,j)$.

In the case of Hamming graph $H(q,t)$, we have a highly regular covering with the subgraphs being q-cliques. Each clique has vertices given by fixing $t-1$ positions of the length t words defining the vertices and looking at the induced subgraph. This gives us tq^{t-1} cliques and each vertex is in precisely q of them. Considering the incidence array of each clique as a $q \times q$ square, we get tq^{t-1} squares. By considering the position of the entry that is varying, we can partition these tq^{t-1} into t rows of q^{t-1} cliques each.

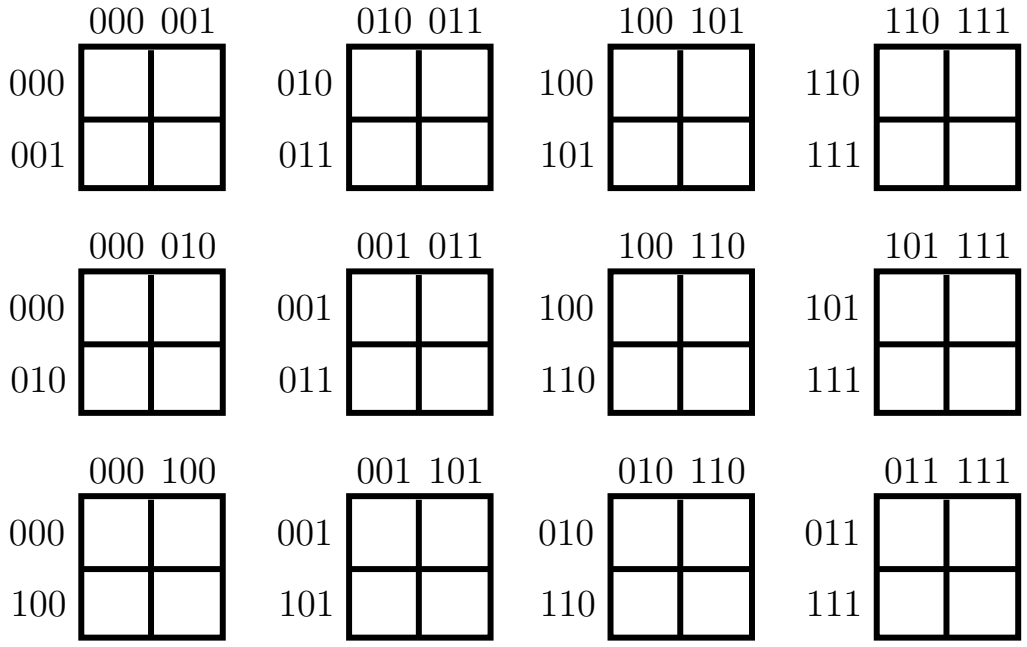


We can label the rows by arranging them to be varying from right to left in positions. The rows are labeled to coincide with the $n = qt$ disks. For example the first row would be labeled by $(-, \ldots, -, 0)$, and the second would be labeled by $(-, \ldots, -, 1)$. Each clique corresponds to fixing $t - 1$ positions. The cliques are labeled horizontally in lexicographical order from left to right; vertically with the missing position going from right to left. For example, the top left clique is $(0, \ldots, 0, -)$. Within each $q \times q$ square, we can label the columns using the lexicographical ordering on the rest of the fixed $t - 1$ positions. We call this labeled version the **array of cliques** presentation of $H(q, t)$. We illustrate the labeling with two examples below.

Example 99. When $q = 3, t = 2$, we have the following:



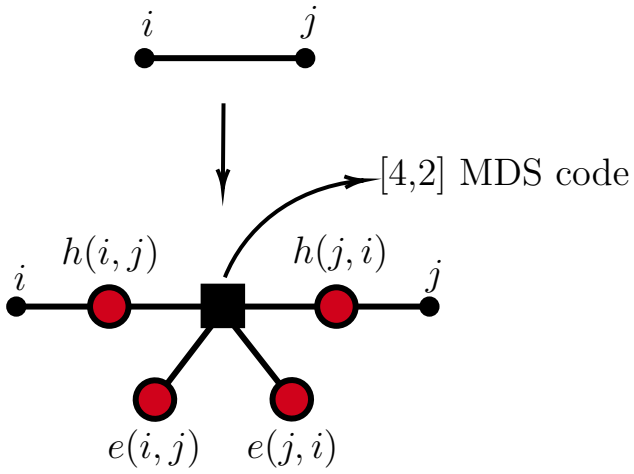
When $q = 2, t = 3$, we have the following:



We see that this covering gives us a natural partition of the $q^t(q-1)t$ edge variables and $q^t t$ vertex variables into $n = qt$ groups. Now we define the hidden variables and checks on $H(q,t)$.

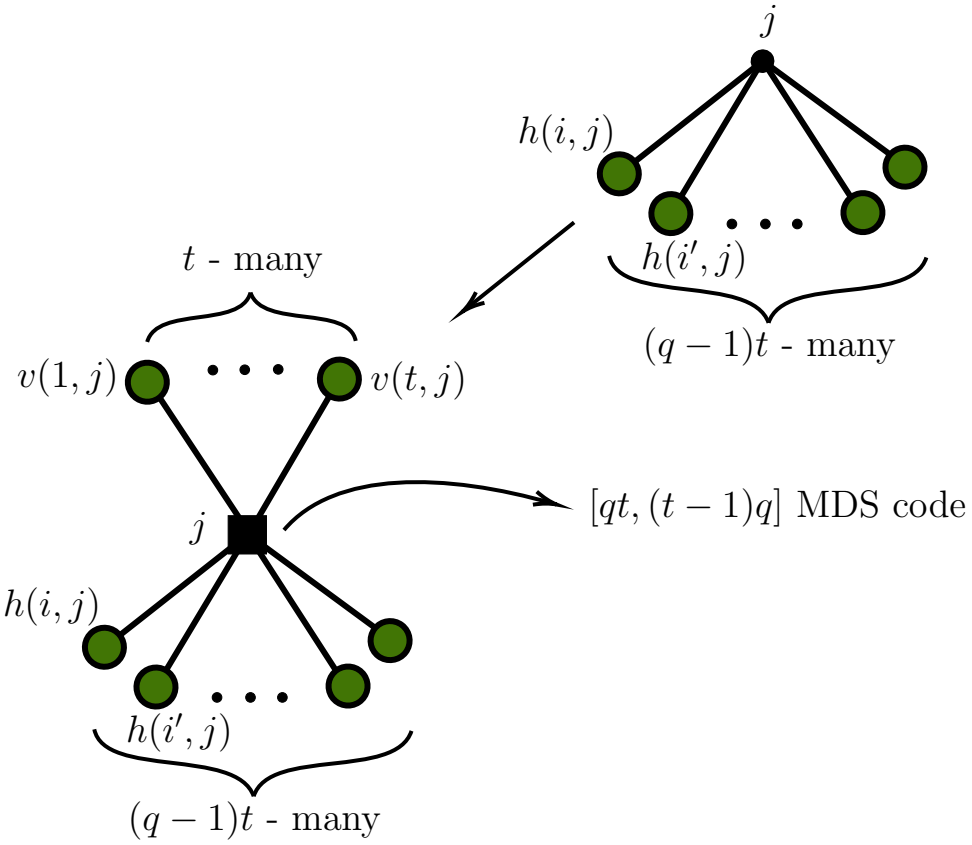
Definition 100. For each edge (i,j) in $H(q,t)$, $i,j \in V(q,t) = A_q^t$, we define $h(i,j)$, $h(j,i)$ to be the associated **hidden variables**.

Definition 101. For each edge (i,j) in $H(q,t)$, we define an **edge check** $E(i,j)$ to be a $[4,2]$ MDS code on its associated edge variables and hidden variables $\{e(i,j), e(j,i), h(i,j), h(j,i)\}$.

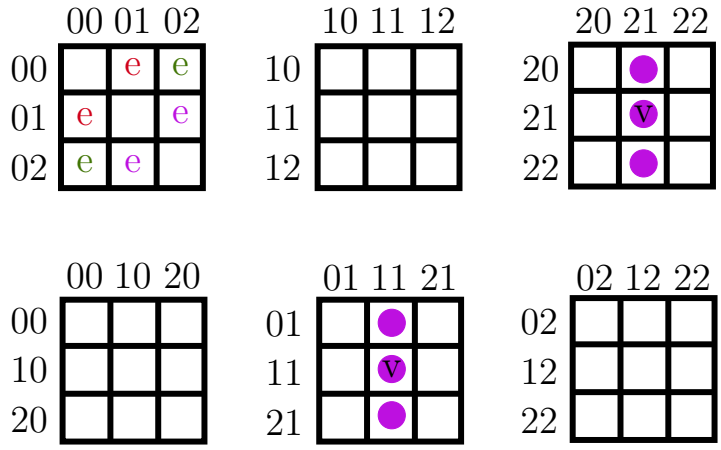


Definition 102. For each vertex $i \in V(q,t)$, we define a **vertex check** $H(i)$ to be a $[qt, (t-1)q]$ MDS code on its associated vertex variables and all hidden

variables with first coordinate being i, that is the set $\{h(i,j), v(\ell,i) | j \in V(q,t), \ell = 1,2,\ldots,t\}$.



On the array of cliques, we have that every edge variable corresponds to exactly one hidden variable. So we can view each diagonal box as storing a vertex variable and each off-diagonal box as storing one edge variable plus one hidden variable. The two kinds of checks can also be viewed as checks on the variables and hidden variables in the boxes. An edge corresponds to two off diagonal entries in reflection across the diagonal in a clique. An edge check corresponds to a $[4,2]$ MDS code on the variables and hidden variables stored on an edge. A vertex check corresponds to all the vertex variables and all the edges hidden variables in a $[qt,(t-1)q]$ MDS code.



4.2.2 Encoding and Dimension of the Code

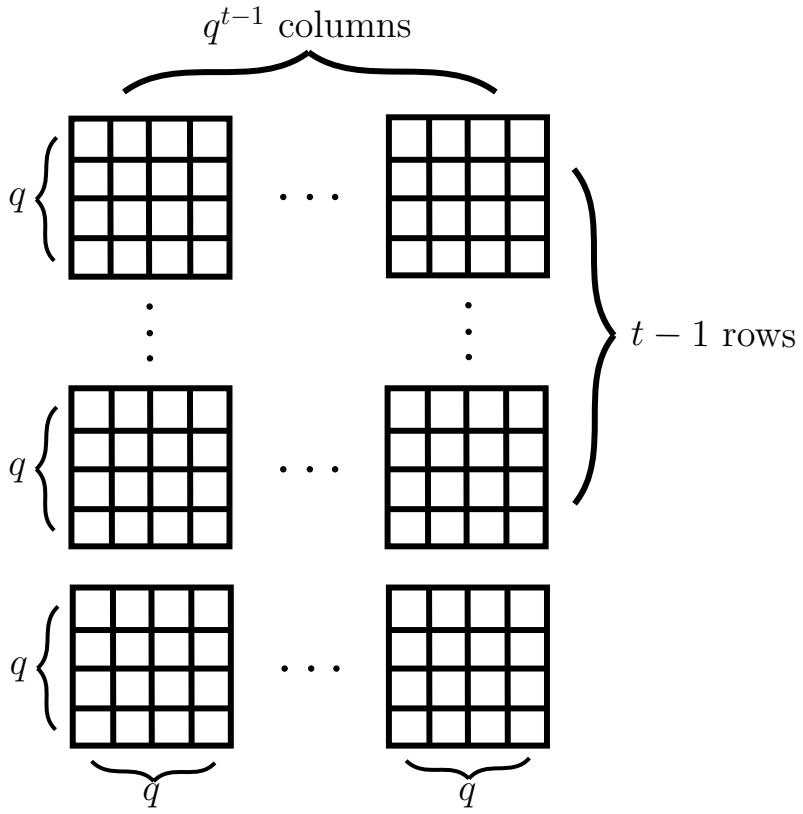
In this part we prove the dimension of the code defined by the checks above. We divide the proof into two propositions. First, we extend an arbitrary message vector of length $k\alpha = (q(t-1))q^t$ uniquely into a vector of length $n\alpha = qtq^t$. This gives an upper bound on the dimension of the code. Then we check that every encoded message is actually a code word, meaning that it satisfies all of the checks we defined above. This gives a lower bound on the dimension of the code. Combining the two propositions and seeing the upper and lower bounds coincide gives that the dimension of the code is $k\alpha = (q(t-1))q^t$, as desired.

Proposition 103. With the edge checks and vertex checks defined above, the code has dimension at most $k\alpha = (q(t-1))q^t$.

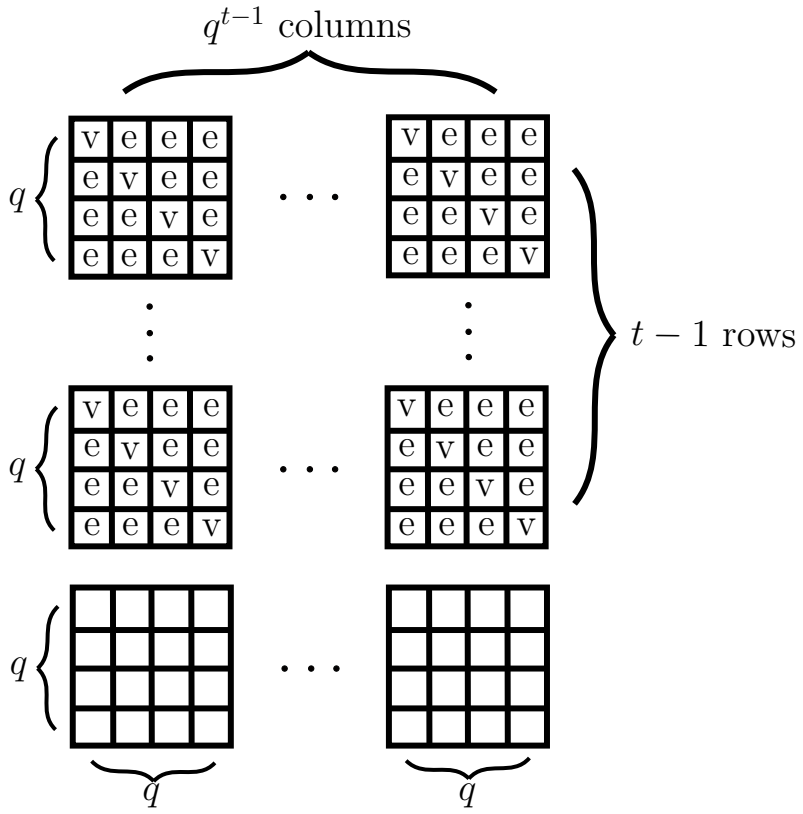
Proof. Given an arbitrary message vector of length $k\alpha = (q(t-1))q^t$, we extend it uniquely into a vector of length $n\alpha = qtq^t$ by the following five steps.

Step 0: We first divide the given arbitrary message vector $\mathbf{v}$ of length $k\alpha = (q(t-1))q^t$ into $k = q(t-1)$ pieces, each of length $\alpha = q^t$. Let $\mathbf{v}_m$ be the m^{th} piece, $1 \leq m \leq q(t-1)$.

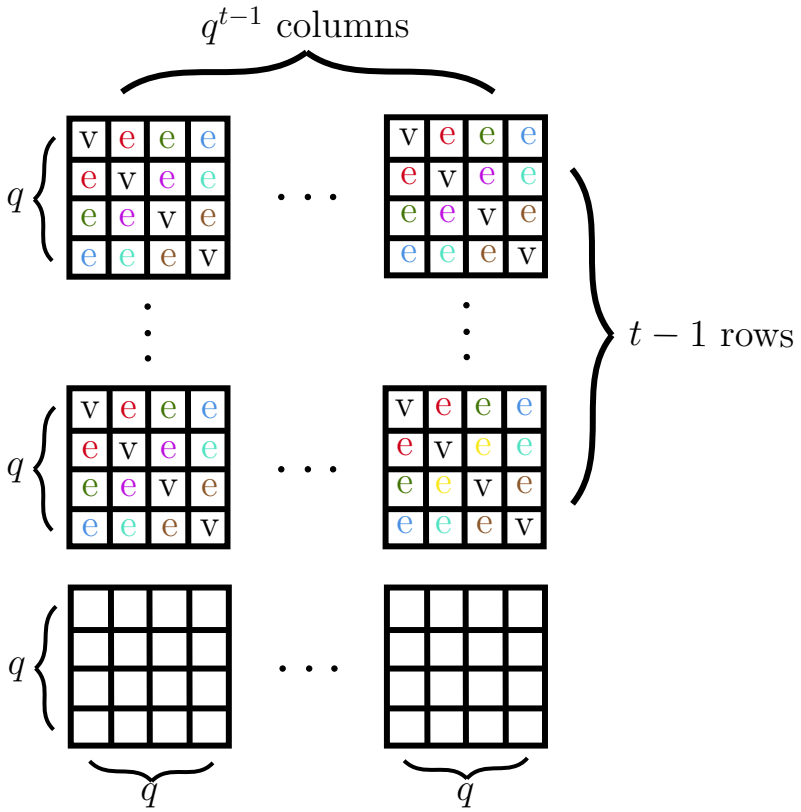
Also recall the array of cliques of $H(q,t)$ has the following form:



Step 1: We take the k vectors from Step 0 and set the content of the first k disks by letting $\mathbf{d}_m = \mathbf{v}_m$, where $\mathbf{d}_m$ is as defined in Definition 98. This can be viewed as putting them into the first k rows of the array of cliques of $H(q,t)$ as the vertex and edge variables. We obtain vertex variables $\{v(\ell,j), 1 \leq \ell \leq t-1, j \in \cup_{m=1}^{q(t-1)} V_m\}$, and the edge variables $\{e(i,j), j \in \cup_{m=1}^{q(t-1)} V_m\}$. They can be visualized as follows in the array of cliques:



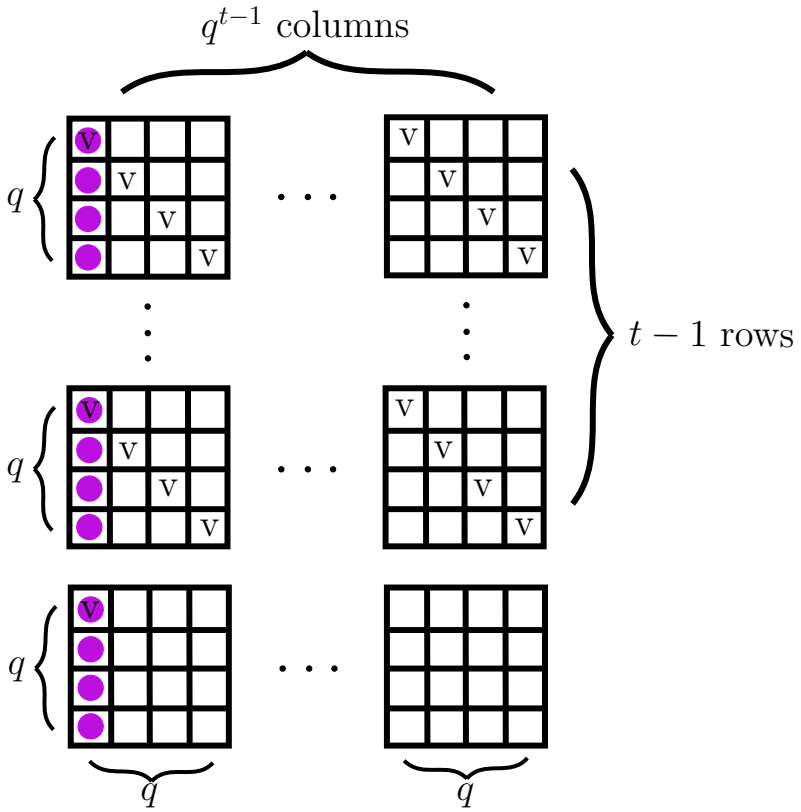
Steps 2: We generate the hidden variables for the first k rows using edge checks. Recall that an edge check is a $[4, 2]$ MDS code on the variables and hidden variables stored in the two off diagonal entries in reflection across the diagonal in a $q \times q$ box. We obtain the hidden variables $h(i, j), j \in \cup_{m=1}^{q(t-1)} V_m$ using the edge checks $\{E(i, j), j \in \cup_{m=1}^{q(t-1)} V_m\}$ in Definition 101. In the array of cliques they can be visualized as follows:



Note that within the same $q \times q$ box the hidden variables with the same color form a $[4, 2]$ MDS code with the edge variables stored at the corresponding positions. This step is unique as we fix the $[4, 2]$ MDS code used for the edge checks.

Step 3: Using the hidden variables that were obtained in the last step, and the vertex checks defined above, we generate hidden variables for the last q rows.

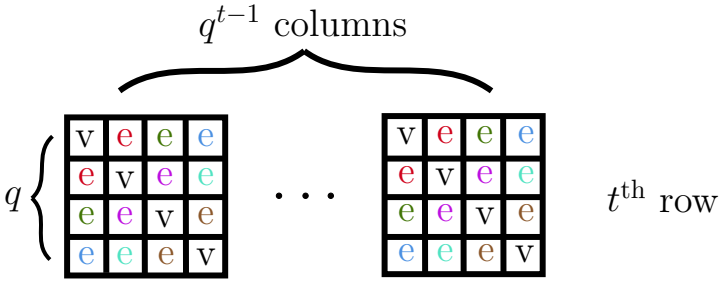
This can be done one (staggered) column at a time, since each of the q^t vertices appears exactly once in each row. For each $i \in V(q, t)$, we have a (staggered) column with $t-1$ vertex variables $\{v(\ell, j), 1 \leq \ell \leq t-1, j \in \cup_{m=1}^{q(t-1)} V_m\}$ obtained from Step 0, and $(t-1)(q-1)$ hidden variables $h(i, j), j \in \cup_{m=1}^{q(t-1)} V_m$ obtained from Step 1. These $q(t-1)$ symbols together allow us to use the vertex check $H(i)$ in Definition 102. Using each edge check exactly once give us the remaining vertex variables $\{v(i, j), j \in V_m, q(t-1)+1 \leq m \leq qt\}$, and the remaining hidden variables $\{h(i, j), j \in V_m, q(t-1)+1 \leq m \leq qt\}$.



This step is unique as we fix the $[qt, (q-1)t]$ MDS code used for the vertex checks.

Step 4 Note that we have all the vertex variables and all hidden variables on edges in the last q rows. We can group the hidden variables in pairs, each occupying the two off diagonal entries in reflection across the diagonal in a $q \times q$ box.

Using the hidden variables $\{h(i,j), j \in V_m, q(t-1)+1 \leq m \leq qt\}$ we obtained from Step 3 and the edge checks $\{E(i,j), j \in V_m, q(t-1)+1 \leq m \leq qt\}$, we obtain the edge variables $\{e(i,j), j \in V_m, q(t-1)+1 \leq m \leq qt\}$.



We can generate the edge variables uniquely form the edge checks and the hidden variables at the corresponding positions.

Every box in the last q rows is filled with the appropriate variables and each step of the process is unique. Combining the vertex variables $v(t,j), j \in V_m, q(t-1)+1 \leq m \leq qt$ obtained in Step 3 and the edge variables $e(i,j), j \in V_m, q(t-1)+1 \leq m \leq qt$ obtained in Step 4, we can fill the entries of the last q rows of the array of cliques. Concatenating the entries of these last q disks with the entries in the first k rows using natural order of the row gives us the missing part of the encoded vector that we need to store. We see that the message vector of length $k\alpha = (q(t-1))q^t$ is extended uniquely into a vector of length $n\alpha = qtq^t$. □

Proposition 104. With the edge checks and vertex checks defined above, the code has dimension at least $k\alpha = (q(t-1))q^t$.

Proof. We can bound the dimension of our code by taking the total number of variables minus the total number of constraints from the checks we used.

The total number of variables is $q^t \times t + (q^t(q-1)t/2) \times 4$. As we have $q^t \times t$ vertex variables, $(q^t(q-1)t/2) \times 2$ edge variables, and $(q^t(q-1)t/2) \times 2$ hidden variables.

The total number of constraints from the checks is $q^t \times q + (q^t(q-1)t/2) \times 2$. As each of the q^t vertex checks contributes to q constraints and each of the $q^t(q-1)t/2$ edge checks each contributes two constraints.

So we have that our code is at least of dimension $q^t \times t + (q^t(q-1)t/2) \times 4 - q^t \times q - (q^t(q-1)t/2) \times 2 = (q(t-1))q^t = k\alpha$. □

4.3 Regenerating Properties: Data Collection and Repair

After constructing the code and showing that it has the right dimension, we show that it has the data collection and repair properties of an exact repair regenerating code.

4.3.1 Data Collection

For data collection, we need that any k out of the n disks can recover the full message. This corresponds to arbitrarily choosing k rows in our array of cliques and recover the other q rows using our checks. Each row in the array of cliques is labeled by a word of length t that is unspecified in all positions except one, where it is occupied by one element of $\{0, 1, \ldots, q-1\}$. We first define a natural notion of distance on the columns with respect to a set of accessed rows to facilitate the proof.

Definition 105. The distance $d_A(i)$ from the column i to an accessed set A of rows is defined to be the minimal Hamming distance to the set of words generated by the labels of rows in A.

Example 106. For the $q = 3, t = 2$ case, let $A = \{(-0), (1-)\}$. We have $d_A((10)) = 0, d_A((00)) = 1$, and $d_A((01)) = d_A((20)) = d_A((22)) = 2$.

For the $q = 2, t = 3$ case, let $A = \{(0--), (-0-), (--0), (--1)\}$, we have $d_A((000)) = d_A((001)) = 0, d_A((010)) = 1$, and $d_A((110)) = d_A((111)) = 2$.

Lemma 107. For a set of accessed rows A with $|A| = k = q(t-1)$, we have $d_A(i) \leq \min(q, t)$, for all i.

Proof. First notice that $d_A(i) \leq t$, for all i and any set of accessed rows A. So the statement is true when $q \geq t$. Now for a set of accessed rows A with $|A| = k = q(t-1)$, we assume $q < t$, and show that $d_A(i) \leq q$, for all i. There are $n = qt$ rows in total and $|A| = k = q(t-1)$. We are missing q rows and since $q < t$, we miss at most q distinct positions for row labels. Hence $d_A(i) \leq q$, for all i. □

Proposition 108. Given the content of any $|A| = k = q(t-1)$ disks, we can recover the content of all $n = qt$ disks. That is, we can recover the missing q rows of the array of cliques from the edge and vertex variables in the rows labeled by elements of A.

Proof. We see that since $|A| = k = q(t-1)$, the labels of rows in A either completely avoid one of the t positions or A has some element with label occupying every position. This gives us two cases.

For the first case, suppose all rows in A avoid one position, we get that all other $q(t-1)$ row labels are in A. That is, we get the edge and vertex

variables in $t-1$ rows of complete blocks in the array of cliques. Data collection in this case is essentially the same as the encoding process. Since the blocks are completely accessed, we can use edge checks to generate all the hidden variables in those $t-1$ rows of complete blocks. This gives us t vertex variables per column and $q(t-1)-t$ hidden variables per column, which allows us to use the $[q(t-1), qt]$ vertex check on the column to recover the last vertex variable and last $q-1$ hidden variables in that column. Finally all hidden variables are recovered in the q rows missing from A. We can use the edge checks in those $q \times q$ cliques to recover the missing edge variables. This shows we can handle the first kind of data collection scenario when A completely avoids one of the t positions.

For the second case, suppose A has some element with label occupying every position. Every column still gets $q(t-1)$ variables but some of them are not in positions where we can generate hidden variables with edge checks. We proceed by columns with increasing distance to A. We start with the following claim:

Claim 109. Any column i with $d_A(i) = 0$ can get its hidden variables completely recovered.

Proof. Suppose row i has $d_A(i) = 0$, we get that in each of the t rows of $q \times q$ blocks, the block with i as the row label has that row with q variables is fully accessed. This means that any variable in position labeled by (i, j) with Hamming distance $d(i, j) \leq 1$ is accessed. In particular, we get the t vertex variables labeled with (i, i). Now observe that for any edge variable labeled with $(j, i), j \neq i$, $d(i, j) = 1$. For the $(q-1)(t-1)$ edge variables in the column labeled by i and are in row $j \in A$, we get that both (j, i) and (i, j) are accessed. So we can use edge checks on these edges to obtain $q(t-1)-t$ hidden variables. Combining the $q(t-1)-t$ hidden variables and the t vertex variables in this column, we can use the vertex check on the vertex labeled by i to obtain hidden variables at the missing q rows. □

Now we extend our proof by showing that columns with $d_A(j) = r+1$ can have its hidden variables completely recovered assuming all columns with $d_A(i) = r$ have their hidden variables completely recovered.

Let us suppose row i has $d_A(i) = r+1$, and all columns j with $d_A(j) = r$ have been completely recovered. Let A_i be the set of row labels on which

A meets the column i. We have the Hamming distance $d(i,k) = 1$, for all $k \in A_i$, and more importantly, $d_A(k) = r$ or $d_A(j) = r+1$. This follows from the way we label the rows and columns in the array of cliques. In the $q \times q$ square containing i and k, we have the following two cases: if the row labeled by i is not accessed we have $d_A(k) = r$, or otherwise we have $d_A(j) = r+1$.

In the first case, by assumption the whole column of k has its hidden variables completely recovered so the the edge variable at position (i,k) together with the hidden variable at position (k,i) allow us to use the edge check to obtain the edge variable and hidden variable at position (i,k).

In the second case, assuming $i \neq k$, since the row labeled by i and k are both accessed, we have one edge variables at position (i,k) and another one at position (k,i). They also allow us to use an edge check to obtain the hidden variables at these two positions. As a subcase of case two, if $i = k$, we just get the vertex variable directly without any computation. So we see that we get a total of $|A_i| = k = q(t-1)$ hidden variables and vertex variables in column i. This allows us to use the vertex check at vertex i and obtain all the hidden variables in that column.

Finally we see that all the missing edge variables for column i with $d_A(i) = r$ will be recovered when using them to generate the hidden variables for column j with $d_A(j) = r+1$. All there is left to show is that for column i with maximum distance to A, we can recover the missing edge variables. But this is true since for any column j with Hamming distance $d(i,j) = 1$, $d_A(j) \leq d_A(i)$, so all the hidden variables we need to use the edge check at the edge (i,j) are already available. □

4.3.2 Repair

For repair we need that any disk can be repaired by receiving $\beta = q^{t-1}$ symbols from each of the remaining $d = n-1 = qt-1$ disks. This corresponds to the following proposition.

Proposition 110. Any row in the array of cliques can be recovered from taking q^{t-1} symbols from each of the remaining $qt-1$ rows.

Proof. Choosing a row in the array of clique corresponds to choosing $m, 1 \leq m \leq qt$. We give the way to choose the helper information from each of the remaining $qt-1$ rows.

First we choose helper information from rows that are not in the same block as m, i.e. for m' where $t_{m'} \neq t_m$. Recall from Definition 98 that V_m has the structure that every $i \in V_m$ has q_m at position t_m. But this gives us that the vertices in V_m can be grouped into q^{t-2} cliques in $t-1$ different ways. Each way corresponds to fixing one of the remaining $t-1$ positions that is not t_m. This corresponds to one of the $t-1$ rows of cliques in the array of cliques. Taking the content of the q^{t-2} cliques is the same as taking $q^2 \times q^{t-2}$ symbols from q rows in a symmetric way, that is q^{t-1} from each row. These are the helper information we take.

For rows labeled by m' with $t_{m'} = t_m$, we see that since they are in the same block, there are exactly q^{t-1} off-diagonal entries in the row m corresponding to off-diagonal entries of row m'. These are the helper information we take.

One can easily check that the helper information chosen above can be used to recover the hidden variables in columns labeled by $\{i \in V_m\}$ using vertex checks $\{H(i), i \in V_m\}$, and the vertex variables $\{v(t_m, j), j \in V_m\}$. Then using the hidden variables in the same block as row m together with the edge variables at the same positions, we can use the edge checks $\{E(i,j), j \in V_m\}$ to recover the edge variables $\{e(i,j), j \in V_m\}$. The vertex variables $\{v(t_m, j), j \in V_m\}$ and the edge variables $\{e(i,j), j \in V_m\}$ together form the content of disk m. $\square$

REFERENCES

[1] J.H. van Lint, *Introduction to Coding Theory*, Springer, 3rd edition, 1999.

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

[3] R. Singleton, "Maximum distance q-nary codes," IEEE Trans. Information Theory, vol. 10, no. 2, pp. 116-118, 1964

[4] J. S. Plank, M. Blaum, and J. L. Hafner, "SD codes: erasure codes designed for how storage systems really fail," in Proceedings of the 11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013, 2013, pp. 95–104. [Online]. Available: http://web.eecs.utk.edu/ plank/plank/papers/FAST-2013-SD.html

[5] M. Blaum, J. L. Hafner, and S. Hetzler, "Partial-mds codes and their application to RAID type of architectures," IEEE Trans. Information Theory, vol. 59, no. 7, pp. 4510–4519, 2013.

[6] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the Locality of Codeword Symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp.6925–6934, 2012.

[7] M. Li and P. P. C. Lee, "Stair codes: A general family of erasure codes for tolerating device and sector failures," Trans. Storage, vol. 10, no. 4, pp. 14:1–14:30, Oct. 2014.

[8] X. Li and I. Duursma, "Sector-Disk Codes with Three Global Parities", Conference proceedings, 2017 IEEE International Symposium on Information Theory. DOI: 10.1109/ISIT.2017.8006601

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

[10] I. Duursma and X. Li, "Johnson Graph Codes", available via arXiv:1912.10388.

[11] I. Duursma and X. Li, "Graphical Representations of MSR Codes", in preparation.

[12] M. Blaum and J. S. Plank, "Construction of two SD codes," CoRR, vol. abs/1305.1221, 2013.

[13] M. Blaum, J. S. Plank, M. Schwartz, and E. Yaakobi, "Partial MDS (PMDS) and sector-disk (SD) codes that tolerate the erasure of two random sectors," in 2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014, 2014, pp. 1792–1796.

[14] M. Blaum, J. S. Plank, M. Schwartz, and E. Yaakobi, "Construction of partial MDS and sector-disk codes with two global parity symbols," IEEE Trans. Information Theory, vol. 62, no. 5, pp. 2673–2681, 2016.

[15] I. Duursma, "Strongly self-orthogonal codes for secure computation." [Online]. Available: http://math.illinois.edu/ duursma/pub/duursma-IMA- 041807.pdf

[16] C. Tian, B. Sasidharan, V. Aggarwal, V. Vaishampayan, and P. V. Kumar, "Layered, exact–repair regenerating codes via embedded error correction and block designs," in *CoRR*, vol. abs/1408.0377, 2014.

[17] K. Senthoor, B. Sasidharan, P. V. Kumar, "Improved layered regenerating codes characterizing the exact-repair storage-repair bandwidth tradeoff for certain parameter sets," 2015.

[18] B. Sasidharan, M. Vajha, P. V. Kumar, "An explicit, coupled-layer construction of a high-rate MSR code with low sub-packetization level, small field size and all-node repair."

[19] M. Elyasi, S. Mohajer, "Determinant coding: a novel framework for exact-repair regenerating codes" *IEEE Trans. Inf. Theory*, vol. 62, no. 12, Dec. 2016.

[20] M. Elyasi, S. Mohajer, "A Cascade code construction for (n, k, d) distributed storage systems", ISIT 2018.

[21] M. Elyasi, S. Mohajer, "Cascade codes for distributed storage systems", arXiv:1901.00911.

[22] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, p. 5227–5239, 2011.

[23] M. Ye, A. Barg "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization", *IEEE Trans. Inf. Theory*, vol. 63, no. 10, p. 6307-6317, 2017.

[24] R. Stanley. *Enumerative combinatorics* Vol. 2, Chapter 7: Symmetric functions, Cambridge Studies in Advanced Mathematics, vol. 62, Cambridge University Press, 1999.