DISSERTATIONES
MATHEMATICAE
UNIVERSITATIS
TARTUENSIS
143

JUNMING KE

Codes for Distributed Storage

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
143

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
143

JUNMING KE

Codes for Distributed Storage

UNIVERSITY OF TARTU
Press
1632

Institute of Mathematics and Statistics, Faculty of Science and Technology, University of Tartu, Estonia.

The thesis has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in Mathematics on 8 October 2024 by the Council of the Institute of Mathematics and Statistics, University of Tartu.

Supervisors: Dr. Ago-Erik Riet
Institute of Mathematics and Statistics
University of Tartu, Estonia

Dr. Henk D.L. Hollmann
Institute of Computer Science
University of Tartu, Estonia

Opponent: Dr. Ragnar Freij-Hollanti
Department of Mathematics and Systems Analysis
Aalto University, Finland

Commencement will take place on November 4, 2024, at 16:15 in Narva mnt 18-1020.

The publication of this thesis was financed by the Institute of Mathematics and Statistics, University of Tartu.

ISSN 1024-4212 (print)
ISBN 978-9916-27-693-3 (print)
ISSN 2806-240X (pdf)
ISBN 978-9916-27-694-5 (pdf)

University of Tartu Press
http://www.tyk.ee

Acknowledgement

First, I would like to express my deepest gratitude to my Ph.D. advisors, Ago-Erik Riet, and Hendrik Dirk Lodewijk Hollmann, for giving me the opportunity to embark on this Ph.D. journey. Ago not only guided me in the completion of this thesis but also helped me settle in Estonia. He provided me with the freedom and support to explore my research topics while steering me toward personal and professional growth. Henk taught me how to conduct research and engage in academic thinking, shaping me into a better researcher with refined professional skills. This journey would not have been as fulfilling without their invaluable support.

I also want to extend my thanks to the institute for its kindness and support, as well as the warm and gracious assistance I received. I am grateful to the professors at the University of Tartu, who have shown passion and dedication in their teaching. In particular, I would like to especially thank professors Vitaly Skachek, Irina Bocharova, and Boris Kudryashov, who taught me the foundational topics necessary for conducting research in coding theory. I would also like to extend my thanks to professors Viktor Abramov, Valdis Laan, Alexander Stolin, Lauri Tart, Krista Fischer, Märt Möls, Kaur Lumiste, Nele Taba, and Eero Vainikko, who provided me with valuable guidance and support in developing my professional skills.

I gratefully acknowledge the funding and travel grants I received for my research from the Estonian Education Department, the University of Tartu Foundation, the XRP Ledger Foundation, the Estonian Doctoral School, the IEEE Estonia Section, the Institute of Mathematics and Statistics at the University of Tartu, the IEEE Information Theory Society, and several universities.

I am incredibly fortunate to have met many wonderful individuals during my Ph.D. years, with whom I shared memorable moments: Sander Mikelsaar, Karan Khathuria, Zahra Alijani, Hina Arif, Shahid Mubasshar, Anastassia Kolde, Fan Zhang, Zhigang Yin, Jieru Li, Shuai Bi, Binghua Liu, Dongdong Chen, Kailu Chen, and Wenyi Fang. I also want to thank those who offered me encouragement during my Ph.D. studies: Xiangfu Song, Ye Dong, Yifan He, Chao Yin, Zheng Yang, Dandan Yuan, Yingying Yao, Yilei Wang, Shi Zhang, and many others who provided me with their warm support.

Finally, my deepest and most heartfelt appreciation goes to my family. I am profoundly grateful for their continuous and unwavering love, help, and support.

Abstract

Nowadays, vast amounts of data are generated by video, voice, text, web content, and other information sources, and these amounts are still growing each year. The storage of this data demands sustainable mechanisms to ensure data integrity and availability. A distributed storage system (DSS) provides a low-cost, but efficient and reliable approach to storage. In a DSS, the data is stored in encoded form across a number of storage units, often referred to as *(storage) nodes*, in a process that adds *redundancy* to the data. Coding techniques are indeed crucial to ensure the reliability of the system, for example for the following reasons. Data in a DSS is often updated, thus making efficient update mechanisms essential. In addition, the repair facilities for addressing potential node failures during the maintenance of the DSS are a critical aspect of DSS design.

In this thesis, we present explicit update-efficient codes that focus on the update performance. The codes are derived from *finite projective planes*. These codes also feature efficient *local repair* and increased *availability* properties, and allow a short description. We provide pairs of sparse generator matrices G and sparse parity-check matrices H for the codes. We have some freedom in the choice of G, thus enabling unequal update frequencies for different data symbols. Moreover, we also evaluate the update and repair performance of a DSS based on these codes.

Two important notions in storage codes are the *node capacity*, the amount of data that can be stored on a storage node, and the *repair bandwidth*, the total amount of data communicated between nodes during a node repair. The trade-off possibilities between node capacity and repair bandwidth in coding schemes are restricted by the *cut-set bound*. The achievable region takes the form of a convex figure bounded by a piecewise linear function. Two of the extreme points or *corner points* of this convex region are of special significance. The extreme point where the node capacity is minimal is referred to as the *Minimum Storage Regenerating* (MSR) point; the extreme point where the repair bandwidth is minimal is referred to as the *Minimum Bandwidth Regenerating* (MBR) point.

Previous research has primarily focused on *exact* repair, which aims to restore the lost content at the failed nodes *exactly*. Most interior points on the cut-set bound curve, and in particular all extreme points other than the MSR and MBR points, cannot be attained using exact repair codes. Interestingly, they can be achieved by a repair strategy called *functional repair*, where the lost content is not recovered exactly, but still data integrity and all further functionality can be maintained over time. We will restrict our studies of functional-repair codes to *linear codes*. Such codes can be described in terms

of collections of vector spaces called *coding spaces* inside the message space, where at each time instant, the actual data stored at a node is described by the coding space currently associated with that node.

Once we have constructed a code for every extreme point of on the cut-set bound, we can construct codes for all other points on the cut-set bound by employing a process termed *timesharing*. It is therefore of the utmost importance to construct codes for each of the extreme points. Constructing *practical* functional-repair codes that allow *efficient* repair is relatively easy for the MSR and MBR points, and this has indeed been done for various sets of parameters, but it seems to be very difficult to achieve for the other extreme points. Indeed, after 15 years of research, only one such example has been found. This functional-repair code, based on a certain vector space partition of a 5-dimensional binary space into eight subspaces, features both a small field size and an efficient repair algorithm, with parameters at a corner point on the cut-set bound distinct from the MSR and MBR points. We construct another such code, which is based on a vector space partition of a 9-dimensional binary space into 73 subspaces of dimension 3. This vector space partition is strongly related to the projective plane $\mathrm{PG}(2, 8)$. The new storage code is equipped with an efficient repair algorithm that can be described using the underlying geometry. Additionally, we provide both geometric and algebraic descriptions of the codes, along with efficient repair methods.

Parts of this work have been presented at the international conferences IEEE International Symposium on Information Theory (ISIT) 2022 and ISIT 2024, and have appeared in their proceedings. Another part of this work is submitted for publication in the international journal Designs, Codes and Cryptography (DCC).

Contents

Acknowledgement **5**

Abstract **6**

1 Introduction **11**
1.1 Communication Channel 12
1.2 Error Correcting Codes 13
1.3 Distributed Storage Systems 15
1.4 Coded Storage 17
1.4.1 Linear Codes 17
1.4.2 Storage Codes 22
1.4.3 Update Performance 29
1.5 Projective Planes 30
1.6 Repair Locality and Availability 35
1.7 Group Actions 38
1.8 Author's Contributions 40
Bibliography 42

2 Update and Repair Efficient Storage Codes with Availability via Finite Projective Planes **51**
Abstract 52
2.1 Introduction 52
2.1.1 Contribution 53
2.2 Preliminaries 54
2.2.1 Distributed storage system, update efficiency, repair locality and repair availability 54
2.2.2 Finite projective plane $\mathrm{PG}(2, q)$ 55
2.3 Linear codes from the finite projective plane $\mathrm{PG}(2, q)$ 55
2.3.1 Incidence matrix A_q 55
2.3.2 Circulant structure of A_q 56

2.3.3 Parity-check matrix H for the linear code $\mathrm{C}^{\perp}(2,q)$ for q prime . . . 57
2.3.4 Generator matrices G for the linear code $\mathrm{C}^{\perp}(2,q)$ for q prime . . . 57
2.3.5 Short description . . . 58
2.3.6 Choice for G: support for unequal update frequencies . 59
2.4 Minimum distance of $\mathrm{C}(2,q)$ and $\mathrm{C}^{\perp}(2,q)$ for q prime, efficient update and local repair with availability, operation of the DSS 59
2.4.1 Data delta updates . . . 60
2.4.2 Repair performance . . . 60
2.4.3 Operation of the DSS: server load, probabilistic models of repair and update, and experimental results . . . 62
2.5 Conclusions and future work . . . 64
Bibliography . . . 65

3 A Binary Linear Functional-repair Regenerating Code on 72 Coding Spaces Related to $\mathrm{PG}(2,8)$ **69**
Abstract . . . 70
3.1 Introduction . . . 70
3.1.1 Storage codes . . . 70
3.1.2 Exact and functional repair . . . 71
3.1.3 The cutset bound and regenerating codes . . . 71
3.1.4 Our contribution . . . 72
3.2 Linear storage codes . . . 73
3.3 A new optimal FR storage code . . . 76
3.4 The automorphism group of the code . . . 78
3.5 Conclusions . . . 81
3.6 Acknowledgment . . . 81
Bibliography . . . 82

4 An Optimal Binary Linear Functional-repair Storage Code with Efficient Repair Related to $\mathrm{PG}(2,8)$ **85**
Abstract . . . 86
4.1 Introduction . . . 86
4.1.1 Storage codes . . . 86
4.1.2 Exact and functional repair . . . 87
4.1.3 The cut-set bound and regenerating codes . . . 87
4.1.4 Our contribution . . . 89
4.2 Linear storage codes . . . 91
4.3 Constructions of linear storage codes using automorphisms . . 95
4.3.1 The type of a proto-state or a state . . . 95

4.3.2 Linear automorphisms 96
4.4 The known regenerating FR storage code 98
4.4.1 The construction 99
4.4.2 The algebraic description 100
4.4.3 The repair algorithm 102
4.5 The new regenerating FR storage code 103
4.6 An algebraic description of the new FR storage code 106
4.7 The automorphism group of the new code 110
4.8 Conics in PG(2, 8) and their relation with the storage code . . 114
4.9 Efficient algebraic repair 117
4.10 The distinguished coding space in a proto-state 121
4.11 Conclusion 122
4.12 Appendix: Details concerning repair 123
Bibliography 128

Summary **132**

Kokkuvõte **133**

Curriculum Vitae **134**

Elulookirjeldus **135**

List of original publications **136**

Chapter 1

Introduction

1.1 Communication Channel

Claude Shannon introduced the main concepts and theorems of what is known as *information theory* in his paper "A Mathematical Theory of Communication" [76]. Information theory attempts to analyze the communication between a *source* and a *destination* through an unreliable *channel*. A typical communication system, as described in information theory, can be illustrated through a block diagram as shown in Figure 1.1.

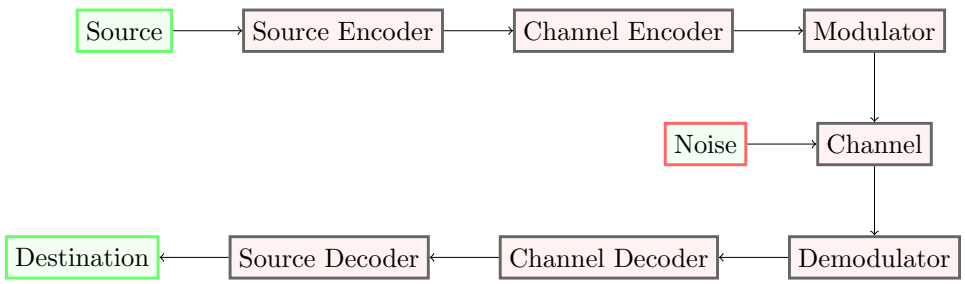

Figure 1.1: Communication system model.

Analog signals, such as a voice picked up by a microphone or light entered into a digital camera, are first digitalized by an Analog-to-Digital (AD) converter, then often compressed, perhaps encrypted, or otherwise processed. All these processes take place in the *source encoder*; the resulting output is a digital data stream referred to as *source data*.

The *channel encoder* then transforms the sequence of (source) data into a sequence of *channel data*. During this process, *redundancy* is added, which is crucial for ensuring accurate transmission in the presence of errors that may occur on the channel, a process facilitated by *error-correcting codes*. The *modulator* and *demodulator* are electronic circuits that convert channel data into physical signals after adapting them for transmission over the specific channel, usually by employing modulation coding. The *channel* is the communication pathway through which data is transmitted in some physical form from the sender to the receiver. Here, the sender and receiver may be separated by distance (*transmission*) or by time (*storage*). Due to the physical nature of the channel, the signal carrying the information will be corrupted during transmission by *noise*. Examples of possible causes of error events include lightning interference during transmission or disk areas that become damaged between the write and read operation. The task of the system is to ensure reliable communication, even when the channel is noisy. The *demodulator* converts the received physical signals back into digital data.

The *channel decoder* exploits the redundancy in the received sequence to correct errors caused by noise and produces an estimate of the original source data. Finally, the *source decoder* performs the inverse operation of the source encoder, delivering the decoded (analogue) output to the destination.

When transmitting a string of coded channel symbols over a channel, we typically distinguish five types of errors.

1. *Error*: The channel may change a coded symbol in the string.
2. *Erasure*: The channel may blur a coded symbol in the string, i.e., erase a coded symbol.
3. *Deletion*: The channel may delete a coded symbol from the string.
4. *Insertion*: The channel may insert a coded symbol into the string.
5. *Interchange*: The channel may interchange coded symbols in the string.

Example 1.1.1 *Suppose the sender transmits the string "11010011". Then the receiver may obtain*
"10010011" (a single error), "10010101" (multiple errors),
"1101?011" (an erasure), "11?1?0?1" (multiple erasures),
"1110011" (a deletion), "111011" (multiple deletions),
"110110011" (an insertion), "10101100101" (multiple insertions),
"11010101" (an interchange), "11100101" (multiple interchanges).

In this thesis, our primary focus is on erasure correcting codes, a specialized type of error correction code (ECC) that specifically addresses erasure problems during data storage.

In this introduction, we present several modern techniques used in data transmission and storage. We begin by discussing error correction, one of the oldest techniques and a foundation for many others. Next, we provide an overview of distributed storage systems, followed by a detailed discussion on storage codes and update efficient codes. This thesis utilizes projective planes to design and explain codes in various contexts, and therefore we will explore some general properties of projective planes. We then examine two key metrics in the repair process: repair locality and repair availability. We also introduce group action as a handy tool, both for code construction and for describing the automorphisms of our constructed code objects. Finally, we conclude with a list of the author's contributions.

1.2 Error Correcting Codes

Coding theory, as pioneered by Claude E. Shannon [76], explores the efficient transmission and storage of information using mathematical codes, essentially by adding redundancy to the transmitted or stored message. The

groundbreaking work of Shannon in the mid-20th century laid the foundation for understanding how to design codes that maximize data transmission rates while ensuring reliable communication in the presence of noise. His insights into information theory revolutionized telecommunications and formed the basis for modern coding techniques as used in digital communication systems, data compression, error correction, machine learning, and data storage [27, 32, 87]. Coding theory continues to influence advancements not only in efficient and robust information transmission technologies but also in related techniques across information processing fields like cryptography, digital communication, and distributed storage systems [14, 21].

In digital communication, there have been developments in implementing Error Correcting Codes (ECC's) to ensure that communications are not compromised by errors [88]. There are many types of ECC's, with different types of codes excelling in different scenario's [69]. Among these, Reed-Solomon (RS) codes and their derivatives are probably the most extensively studied and utilized in modern technologies [63]. The most notable advantage of RS codes is their Maximum Distance Separable (MDS) property, which makes RS codes optimal in the sense of delivering the best error-correction potential for their size. In addition, RS codes are known for their efficient encoding and decoding algorithms [49]. Generalized Reed-Solomon (GRS) codes, which are a generalization of RS codes, are also optimal in the sense of meeting the Singleton bound on the minimum distance of an $[n, k]_q$ code (we will explain it in more detail in Section 1.4.1). Bose-Chaudhuri-Hocquenghem (BCH) codes form a class of cyclic ECC constructed using polynomials over a finite field [13]. Both RS and GRS codes are special cases of BCH codes, and most of their error-correction algorithms are specializations of similar methods for BCH codes.

The error correcting codes mentioned before are *block codes*, where blocks of data symbols, each of the same size, are encoded into codewords, also all of the same (longer) size. Convolutional codes are a different type of ECC's. In convolutional codes, a number of parallel sequences of data symbols are encoded into a larger number of parallel sequences of coded symbols, where each coded symbol is a function not only of the present data symbols, but also of a fixed number of "earlier" data symbols [55]. These convolutional codes transmit more data symbols per unit of time than the original amount of data symbols per unit of time. The resulting redundancy can then be used for error correction. Turbo codes, a newer form of convolutional-type codes, are easier to decode and can closely approach the theoretical limits imposed by Shannon [86]. The Viterbi algorithm is widely used as an efficient method for decoding convolutional codes [29]. The Viterbi algorithm is equivalent to a dynamic programming solution to the problem of finding the shortest path

through a weighted graph [60]. Indeed, the Viterbi algorithm was recognized as a maximum likelihood decoding algorithm for convolutional codes, that is, the decoder consistently outputs the most likely original input sequence given the output sequence [29].

Turbo codes compete with Low-Density Parity-Check (LDPC) codes, which offer similar performances. Such a code is built using a sparse *Tanner graph*, a special type of bipartite graph without small cycles. The LDPC concept was developed by Robert G. Gallager in his 1960 doctoral dissertation. It was not widely used for decades because the amount of hardware needed for such an algorithm was not feasible at that time, but nowadays this is no longer an issue. Gallager showed that LDPC codes can achieve the *Gilbert–Varshamov bound* for linear codes over binary fields with high probability [30]. About 30 years later, inspired by Turbo codes, an efficient iterative decoding algorithm for LDPC codes was proposed, and this raised renewed interest in these codes. Today, LDPC codes are widely used because of their low Bit Error Rate (BER) values [54].

1.3 Distributed Storage Systems

In this section, we will introduce Distributed Storage Systems and discuss some of their general aspects. Data storage refers to the recording of information on computers, servers, or other devices with the aim of preserving the data over time [79]. Before the information age, data was stored on analog videotapes with low capacity, such as Video Home System (VHS) cassettes. More recently, such low-capacity storage methods have been inadequate for managing the large amounts of data that are currently generated. Globally, the amount of data in need of storage has grown at 28 percent per year [40], and is estimated to grow to 125 ZB (1 ZB = 10^9 TB) by the end of 2024, to 175 ZB in 2025, and to 393 ZB at the end of 2028 [90]. In the past decades, efficient, reliable, and high-capacity data storage methods have been developed, and various techniques have been proposed to handle these enormous amounts [9, 16].

Now we discuss Distributed Storage Systems (DSS's), which have emerged over the past few decades as a widespread solution for large-scale data storage [27]. A DSS employs numerous physical storage units, typically data racks or servers in a data center, often referred to as (storage) nodes. Incoming data in need of storage is first encoded and then partitioned into blocks, each of which is stored on a node. The system performs as a cluster of interconnected physical storage units, providing mechanisms for data synchronization and coordination between the various nodes within the clus-

ter [17]. Many well-known DSS's, such as Google File System [31], Facebook Hadoop Distributed File System (HDFS) [70], and Microsoft Azure [15, 44], use low-cost, unreliable components to reduce costs, which increases the likelihood of failures. As a result, effective error-handling methods are necessary.

A smart use of added redundancy can enhance the reliability of data stored in a DSS. The amount of redundancy that is added during encoding is referred to as the *overhead*, and the fraction of information contained in the data is referred to as the *rate* of the encoding in use. Of course, the game that needs to be played is to get the most reliability from the smallest overhead. A simple way to protect the data is by using *repetition*, where nodes maintain multiple copies of the data. If one node is lost, the original data can still be restored from other nodes. However, it turns out that this simple method is not very efficient in terms of required overhead, and indeed much more efficient methods are available. Typically, in these methods, more advanced coding techniques offer the same level of reliability as repetition but with significantly less overhead.

Codes have been commonly employed in situations where the unreliability of the available communication channel causes erasure(s) and error(s) during transmission [89]. As we saw before, the sender encodes the original data before transmission or storage. Later, the receiver decodes the received or stored data to restore the original information. This concept of sender, receiver, and transmission between them has been extensively studied from various perspectives, and plays a fundamental role, particularly in DSS's.

Indeed, a DSS can be seen as a communication channel that involves time, where the receiver is just the sender itself, but at a later time. As we saw before, the user encodes the original data and then partitions the coded symbols into data blocks, which are each stored on a storage node. After some time, a user can collect the data stored in surviving nodes and then decode that data to restore the original data. Even when some nodes provide incorrect coded symbols, or no symbols at all, the original data can still be recovered due to the employed ECC's. However, the storage model is more difficult to describe because it involves more operations during the maintenance of the DSS.

The first additional operation is *update*: a user sometimes may want to make a change in the data that the user stored previously, thus necessitating an update of the encoding of that data as stored in the DSS. The second additional operation is *repair*, which may be needed due to the failure of a storage node. When a node fails, the DSS introduces a *newcomer node* into the system to replace the failed node. An obvious problem is to determine what data should then be stored in the newcomer. We will discuss the storage process and explain update and repair operations in more detail later.

1.4 Coded Storage

1.4.1 Linear Codes

In a storage system, text, numbers, images, audio, and virtually any other type of information can be converted into a string of bits. Usually, *binary strings* are used, where each bit is a variable on 0/1. For instance, characters in text are often encoded using standards like ASCII or Unicode, images are converted into binary through formats like JPEG or PNG, and audio is digitized using formats such as MP3 or WAV. We refer to such a string of bits as *data*.

To prepare for data storage, a DSS first partitions the data into blocks, often referred to as *data symbols*. As already discussed in Section 1.3, applying replication to store data symbols is a viable, but inefficient method. A better solution is letting the DSS store some well-chosen linear combinations of the data symbols. We will provide further details on this solution in this section.

To begin with, we need some preparations. Let $\mathbb{F}_q$ denote the finite field of q elements. Such a field exists if and only if q is a power of a prime p [25]. We denote the space of row vectors of length n over $\mathbb{F}_q$ by $\mathbb{F}_q^n$, or, sometimes, by $V(n, q)$. A subset S of $\mathbb{F}_q$ is a *subspace* of $\mathbb{F}_q^n$ if, for any two elements $s_1, s_2 \in S$ and any two scalars $\lambda_1, \lambda_2 \in \mathbb{F}_q$, the linear combination $\lambda_1 s_1 + \lambda_2 s_2$ also belongs to the subset S. A set B of vectors in a vector space S is called a *basis* if elements of B are *linearly independent* and every element in S is a *linearly combination* of elements of B. The *dimension* of a subspace S is defined as the number of elements in a basis for S. For more details on vector spaces, subspaces, and definitions of all these notions, see [25].

Definition 1.4.1 (Block Code) *Let n be a positive integer and let q be a prime power. A q-ary block code C over $\mathbb{F}_q$ is a nonempty subset of $\mathbb{F}_q^n$. Sometimes we omit the reference to the field size q and simply speak of a* block code.

We will mostly focus on linear codes, which are defined below.

Definition 1.4.2 (Linear Code) *Let n and k be positive integers and let q be a prime power. A linear code of length n and dimension k over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The dimension of the subspace C is referred to as the* dimension *of the code and is denoted by* $\dim(C)$. *The* rate *of the code is k/n.*

We refer to the elements of the code as codewords, and we refer to a q-ary linear code of length n and dimension k as an $[n,k]_q$ code, or simply an $[n,k]$ code if the field can be deduced from the context. Sometimes, we will refer to elements of $\mathbb{F}_q^n$ as q-ary *words of length* n , or simply, as words of length n if the value of q is clear from the context. A *generator matrix* for a code C is a $k \times n$ matrix with as rows the vectors that form a basis for C.

Let $\boldsymbol{x} = (x_1, ..., x_k)$ be a row vector of k data symbols over $\mathbb{F}_q$. When an $[n,k]_q$ linear code is employed to encode data, a row vector of k data symbols $\boldsymbol{x} = (x_1, ..., x_k)$ over $\mathbb{F}_q$ is encoded into a codeword $\boldsymbol{c} = (c_1, ..., c_n)$, where each coded symbol c_i $(i = 1, ..., n)$ is a linear combination of $x_1, ..., x_k$.

As a consequence, we can write the codeword $\boldsymbol{c}$ as

$$\boldsymbol{c} = \boldsymbol{x} \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix} = \boldsymbol{x}G, \tag{1.1}$$

where G is a generator matrix for C. Generator matrices are important since they describe the mapping from data symbols to coded symbols, that is, they describe the *encoder* for the code.

We say that a $k \times n$ generator matrix G is in *systematic* form if (up to a permutation of the columns) it has the form $G = [I_k \mid P]$, where I_k denotes the $k \times k$ identity matrix. Note that if the encoder uses a systematic generator matrix, then the data symbols all appear in the corresponding codeword. In this case, such an encoder is called a *systematic encoder*. Note that the generator matrix of a linear code, like a basis for a code, is in general *not* unique. If the generator matrix of an encoder is not systematic, then we say that it is *non-systematic*. If the encoder employs a non-systematic generator matrix to encode the data, then we say that the encoder is *non-systematic*.

When using a systematic encoder, every data symbol appears among the coded symbols, so it is easy to recover the data symbols from the coded symbols. In a non-systematic encoder, we need to compute some non-trivial linear combinations on the coded symbols to recover the data symbols.

Example 1.4.1 *1) Employing the generator matrix*

$$G_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{1.2}$$

we can encode a data vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)$ *into the codeword*

$$\boldsymbol{c} = \boldsymbol{x}G_1 = (x_1, x_2, x_3, x_4, x_5, x_2 + x_4, x_1 + x_3, x_1 + x_5). \tag{1.3}$$

Note that the generator matrix G_1 *is systematic.*
2) Employing the generator matrix

$$G_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{1.4}$$

we can encode a data vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ *into the codeword*

$$\begin{aligned} \boldsymbol{c} = \boldsymbol{x}G_2 = (&x_1 + x_2, x_2 + x_3, x_3 + x_4, x_1 + x_4, \\ &x_1 + x_2 + x_3, x_1 + x_3 + x_4, x_1 + x_2 + x_4, x_2 + x_3 + x_4). \end{aligned} \tag{1.5}$$

Note that in this case, the generator matrix G_2 *is non-systematic.*

Let C be an $[n, k]$ code with $k \times n$ generator matrix G. A full rank $(n-k) \times n$ matrix H such that $HG^\top = \mathbf{0}$ is called a *parity-check matrix* for C. Note that then a vector $\boldsymbol{c} \in \mathbb{F}_q^n$ is a codeword in C if and only if $H\boldsymbol{c}^\top = \mathbf{0}$ [69]. As for the generator matrix, a parity-check matrix of the code C is not unique for a similar reason.

Let C be an $[n, k]$ code with $k \times n$ generator matrix G and $(n - k) \times n$ parity-check matrix H. The *dual* $C^\perp$ of C is the code that has H as its generator matrix. Note that the dimensions of code C and its dual $C^\perp$ add up to the length n, that is,

$$\dim C + \dim C^\perp = n. \tag{1.6}$$

This property is often referred to as the rank–nullity theorem. As a consequence, the dual $C^\perp$ is an $[n, n - k]$ code with generator matrix H and parity-check matrix G. In contrast to the situation for real or complex numbers, for finite fields it may happen that the *hull* $C \cap C^\perp$ of a code C is *nonempty*. For example, over $\mathbb{F}_2$, the linear code $C := \{(0, 0), (1, 1)\}$ is its own dual, hence its hull equals C.

Above we have seen that a linear code can be defined by specifying a generator matrix, or by specifying a parity-check matrix for the code. For example, a Low-Density Parity-Check (LDPC) code is a code specified by a *sparse* parity-check matrix, i.e., a parity-check matrix that has many zeros.

The ability to correct errors depends on how well different codewords can be distinguished in the presence of noise. For this reason, in the field of error

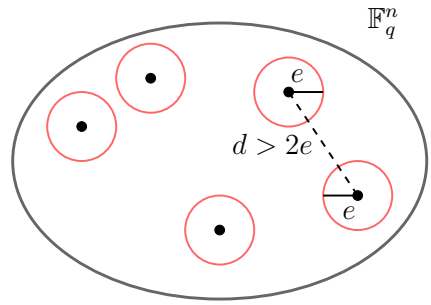

Figure 1.2: In an e-error-correcting code, balls with radius e around codewords have to be disjoint.

correction and detection, the following notions are fundamental. The *Hamming distance* between two codewords is defined as the number of positions in which they differ. Formally, we have the following.

Definition 1.4.3 (Hamming distance) *The Hamming distance $d(\boldsymbol{x}, \boldsymbol{y})$ between two words $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n$ is the number of positions $i \in \{1, 2, ..., n\}$ such that $x_i \neq y_i$. The Hamming weight $w(\boldsymbol{x})$ of a word $\boldsymbol{x} \in \mathbb{F}_q^n$ is the number of positions $i \in \{1, 2, ..., n\}$ such that $x_i \neq 0$; in other words, $w(\boldsymbol{x}) = d(\boldsymbol{x}, \boldsymbol{0})$.*

For any words $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ in $\mathbb{F}_q^n$, Hamming distance satisfies the *triangle inequality* $d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z})$. The *minimum distance* $d(C)$ of the code C is the smallest Hamming distance between two distinct codewords in C, and the Hamming weight $w(C)$ of a code C is the smallest Hamming weight of a non-zero codeword in C. Note that in a linear code, we have $d(C) = w(C)$. In fact, a code with minimum distance d can simultaneously correct e errors and f erasures provided that $2e + f < d$ [69]. This can be simply understood as follows. The *ball* with radius r around a word $\boldsymbol{x}$ is the set $B_n(\boldsymbol{x}, r) := \{\boldsymbol{y} \in \mathbb{F}_q^n \mid d(\boldsymbol{x}, \boldsymbol{y}) \leq r\}$, see Figure 1.2 for an illustration. Let a codeword $\boldsymbol{c}$ from C be transmitted, and let $\boldsymbol{r}$ be the word obtained by the receiver. Suppose that during transmission, f erasures are introduced, together with at most e further errors. Consider the code C' obtained from C by *puncturing* in the positions of the erasure. The resulting code C' has a minimum distance (at least) $d - f$. If $\boldsymbol{c}' \in C'$ and $\boldsymbol{r}'$ are the result of puncturing the codeword $\boldsymbol{c}$ and the received word $\boldsymbol{r}$ in these positions, then the situation is as if $\boldsymbol{c}' \in C'$ was transmitted and $\boldsymbol{r}'$ was received; since $\boldsymbol{r}'$ contains at most e errors, these errors can be corrected as long as $2e < d(C')$, so certainly if $2e < d - f$. The f remaining erasures in $\boldsymbol{r}$ can now be resolved easily. Indeed, if $d > f$, then distinct codewords remain distinct after puncturing in at most f positions; therefore, a code with $d > f$ can correct up to f

erasures. The size of an $[n,k]_q$ code equals q^k [69]. We can obtain a bound for the minimum distance d of an $[n,k]_q$ code C by the following observation. Delete the first $d-1$ coordinates of all codewords (puncturing again). Since any two codewords originally had distance at least d, after deletion any two resulting words are still distinct. Since we can obtain at most q^{n-d+1} distinct words of length $n-d+1$, we conclude that $|C| \leq q^{n-d+1}$. Together with $|C| = q^k$, we obtain the Singleton bound, stating that

$$d \leq n-k+1. \tag{1.7}$$

An $[n,k]_q$ code with a minimum distance d is also referred to as an $[n,k,d]_q$ code, or simply $[n,k,d]$ code. When an $[n,k,d]$ code satisfies $d = n-k+1$, we call this code a Maximum Distance Separable (MDS) code. Given an $[n,k,d]_q$ MDS code, so with $d = n-k+1$, the linear MDS conjecture states that $n \leq q+1$ unless $q = 2^h$ (that is, even) and $k = 3$ or $k = q+1$, in which case $n \leq q+2$ [71]. Recently, this conjecture has been proved for the case where q is a prime [7]. One of the most famous families of MDS codes are Reed-Solomon codes [67]. These codes exist if $n \leq q+1$ [73]. Note that any k columns of the generator matrix for an MDS code are linearly independent, and any $n-k$ columns of the parity-check matrix for an MDS code are also linearly independent [85]. In other words, for an MDS code, the k original data symbols can be constructed from any given k coded symbols.

Example 1.4.2 *1) Consider a systematic generator matrix*

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{1.8}$$

over $\mathbb{F}_2$. The code C with generator matrix G has length 7 and dimension 3. The minimum distance of this code is 4, so this code is not an MDS code. A parity-check matrix of this code is

$$H_1 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.9}$$

Another parity-check matrix of this code is

$$H_2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{1.10}$$

Both H_1 and H_2 satisfy $HG^\top = 0$ (i.e., $H_1G^\top = 0$ and $H_2G^\top = 0$). The corresponding code C is called the Simplex code of length 7, and the dual code is the Hamming code of length 7.
2) For an example of an MDS code, consider a systematic generator matrix

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{1.11}$$

over $\mathbb{F}_2$. The corresponding code C with this generator matrix G has length 3 and dimension 2. The minimum distance of this code is 2, so this code is a (trivial) MDS code. In fact, MDS codes over the binary field are trivial, that is, they are either $[n, k = n, d = 1]_2$ codes or $[n, k = n - 1, d = 2]_2$ codes.

1.4.2 Storage Codes

We will now discuss a commonly used model for a storage code used in a DSS, known as *regenerating codes*, a model that we will use in our further analysis. The maximum number of coded symbols that can be stored in a node is termed the *node capacity* (or sub-packetization level), which we will denote by α. We will assume that, in order to store a data file consisting of m data symbols from some finite field $\mathbb{F}_q$, the DSS first encodes this data file into a *coded file* consisting of $n\alpha$ coded symbols, then partitions this file into n *blocks* of α coded symbol each, and stores each block on a node. Using a small value of α is desirable since this is a measure for the coding rate $R = m/n\alpha$.

The nodes in a DSS are prone to occasional failures. When a node fails, the DSS first introduces a replacement node (often called a *newcomer node*) into the system. To construct the data for this node, the DSS chooses a certain subset of the nodes (called the *repair set*), and then contacts each node in this set (called a *helper node*) in order to download a limited amount of data from that node. The replacement data for the newcomer node is then constructed by the DSS from the data thus obtained. The replacement data on the newcomer node should be such that the DSS maintains the same functionality and performance as before.

We distinguish two variations in the node repair model. We speak of *repair-by-transfer* if, during node repair, a helper node can only deliver data symbols that are actually stored on that node. In a less restricted repair model, we allow a helper node to *compute* the data that it delivers from the data stored on that node. Below, we adhere to this more general form of repair, which lends itself better for a more information-theoretical analysis (the repair-by-transfer model is more combinatorial in nature).

We say that the code has *repair locality* r if every lost node can be repaired with the help of at most r other nodes. A code has *transport capacity* β if during repair a node never provides more than β data symbols. In the part of the thesis about regenerating codes, we will make the assumption that every set of r live nodes can function as a repair set for a lost node; in addition, we assume that a regenerating code *every* node has transport capacity β. In the part of the thesis about update efficient codes we will make no such assumption. We will later discuss repair locality and related notions relevant for update efficient codes more deeply. For regenerating codes, we assume that at any time, the original data file that was stored by the DSS can be reconstructed from any subset of size k of the nodes. During recovery, such a subset will be referred to as a *recovery set*.

The total amount of data that is exchanged between the storage nodes in the repair process is termed the *repair bandwidth*, which we will denote by γ (note that $\gamma = r\beta$). It is desirable to minimize the repair bandwidth because a large value of repair bandwidth consumes valuable resources. A storage code used in this model is referred to as an $\{m, (n, k), (r, \alpha, \beta)\}$ *regenerating storage code*.

It has been shown that assuming the above model, the possible trade-offs between the storage capacity α and the repair bandwidth $\gamma = r\beta$, are guided by the cut-set bound [24]

$$m \leq \sum_{i=0}^{k-1} \min(\alpha, (r-i)\beta), \tag{1.12}$$

where as usual we assume $r \geq k$. Here, if $k > r$, the same bound is obtained when we lower k until $k = r$, so we may assume without much loss of generality that $r \geq k$. The cut-set bound is an inequality involving m, k, r, α, β (and is independent of n). It can be used in (at least) two ways. First, we can consider it a bound on m given the other parameters $k, (r, \alpha, \beta)$; codes attaining this bound on m are called *optimal*. Secondly, we can think of the bound as showing the trade-off possibilities between α and β, for fixed m, k, r. Indeed α and β are more-or-less assumed to be *integers*. In terms of the *normalized* parameters $\overline{\alpha} := \alpha/m$ and $\overline{\beta} := \beta/m$, the cut-set bound takes the form

$$1 \leq \sum_{i=0}^{k-1} \min(\overline{\alpha}, (r-i)\overline{\beta}). \tag{1.13}$$

The attainable $(\overline{\alpha}, \overline{\beta})$-region takes the form of a convex region (for fixed m, k, r), bounded by a piecewise linear function, as shown in Figure 1.3. The cut-set bound is a purely information-theoretical bound, originally obtained

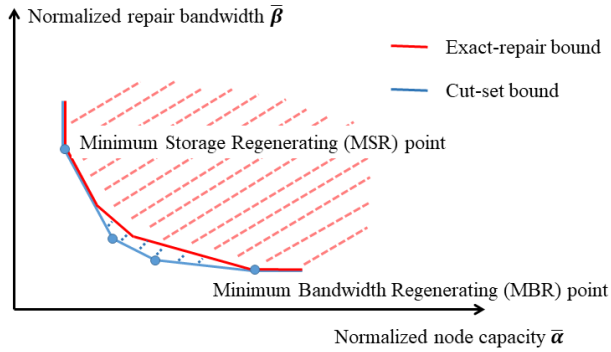

Figure 1.3: A conceptual sketch of the cut-set bound region.

by setting up a network representing the information flow during repairs, and analyzing this flow by using the Ford-Fulkerson algorithm [28, 64]. A code attaining a point on the boundary of the cut-set region is termed *optimal*.

The piecewise linear boundary indicates the possible trade-offs between α and β. Two of the points on the cut-set bound are of special interest. The extreme point where α is minimal is called the Minimum Storage Regenerating (MSR) point, and the extreme point where β is minimal is called the Minimum Bandwidth Regenerating (MBR) point; both are indicated in Figure 1.3. A code at the MSR or MBR point is called an MSR or MBR code, respectively. Note that both an MSR and an MBR code are optimal codes.

To compute the MSR point, we first minimize α to obtain the absolute minimum

$$\alpha = \frac{m}{k},$$

and given this value for α, minimize β to obtain

$$\beta = \frac{\alpha}{r - m + 1} = \frac{m}{K(r - m + 1)}.$$

To compute the MBR point, we first minimize β to obtain the absolute minimum

$$\beta = \frac{2m}{k(2r - k + 1)},$$

and given this value for β, minimize α to obtain

$$\alpha = r\beta = \frac{2mr}{k(2r - k + 1)}.$$

The simplest repair strategy is to let the data constructed on the newcomer node be an *exact* copy of the data on the lost node. This is called

exact repair. Although commonly used, not all attainable points on the cut-set bound can be realized by codes that employ exact repair. A more subtle repair strategy is to employ *functional repair*, where after repair the reconstructed data can be different from the original data, but overall data integrity is still maintained over time. A code that only employs exact repair is referred to as an *exact-repair code*, otherwise, the code is referred to as a *functional-repair* code. Obviously, every exact repair code is also a functional-repair code. A functional-repair code that is not an exact repair code is sometimes referred to as a *strictly* functional-repair code.

It is known that an MSR code is just an MDS code with optimal repair bandwidth [92]. Moreover, it has been shown that only the MSR and MBR points on the curve together with the points on a small segment starting from the MSR point and ending before the first corner point can be attained by exact repair codes [74,84]. As a consequence, a storage code with parameters that attain an extreme point of the cut-set region different from the MSR or MBR points cannot be an exact repair code. Once a storage code is constructed for every extreme point on the cut-set bound, any point on the cut-set bound between two extreme points can be attained by a code obtained from the codes in the two extreme points by *time-sharing* [84]. It is therefore of great importance to construct codes that realize the extreme points other than the MSR and MBR points.

Example 1.4.3 *Consider the code C that stores a vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5) \in \mathbb{F}_2^5$ across four nodes, where*

- *The first node stores x_1, x_2,*
- *The second node stores x_3, x_4,*
- *The third node stores $x_5, x_2 + x_4$,*
- *The fourth node stores $x_1 + x_3, x_1 + x_5$.*

So the code C has codewords $\boldsymbol{c} = ((x_1, x_2), (x_3, x_4), (x_5, x_2+x_4), (x_1+x_3, x_1+x_5))$, considered as a word in $(\mathbb{F}_2^2)^4$. A failed node can be repaired exactly from three other nodes in this storage code. For example, if the contents of the first node are lost, then a replacement node can download from helper nodes (second, third, and fourth) the coded symbols $(x_3, x_4), x_2 + x_4, x_1 + x_3$ and exactly reconstruct the two lost coded symbols x_1 and x_2. In this example, one node should provide two coded symbols during repair, i.e., the second node provides x_3 and x_4. So this exact repair code has repair locality $r = 3$ and transport capacity $\beta = 2$.

Example 1.4.4 (Continued) *If the contents of the second node ((x_3, x_4)) are lost, then a replacement node can download from helper nodes (first, third, and fourth) the coded symbols $(x_1+x_2), (x_5+x_2+x_4), (x_3+x_5)$ (compute repair vectors), respectively. The replacement node then uses the addition operation over $\mathbb{F}_2$ to obtain coded symbols $x_1 + x_4 + x_5$, $x_1 + x_2 + x_3 + x_5$. Here, the contents of the replacement node are different from the contents of the lost node. Now codewords are $((x_1, x_2), (x_5, x_2 + x_4), (x_1 + x_3, x_1 + x_5), (x_1 + x_4 + x_5, x_1 + x_2 + x_3 + x_5))$. It is easy to verify that coded symbols stored in any three nodes can recover the data vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)$. For the reason why the subsequent functional-repair process still works, we refer the reader to [46]. The functional-repair code in this example is a $\{5, (4, 3), (3, 2, 1)\}$ code, which attains a point on the cut-set bound curve, i.e., this code is optimal in terms of the cut-set bound.*

Most storage codes used in practice are *linear*, in the sense that if we represent the data file stored by the DSS by a vector $\boldsymbol{x}$ of length m over a finite field $\mathbb{F}_q$ (often the binary field $\mathbb{F}_2$), then all the computations performed in the DSS are linear, in particular,

(1) at any moment in time, each of the coded symbols stored by a node can be represented by a linear combination of the original data symbols;

(2) the β symbols computed by a helper node during repair are each a linear combination of the coded symbols stored in that node;

(3) the replacement data stored on a newcomer node during repair is a linear combination of the $\gamma = r\beta$ symbols obtained from the helper nodes in the repair set, and

(4) during recovery, the original data symbols are each computed as a linear combination of the coded symbols stored on the nodes of a recovery set.

This leads to the following *vector space view* [41, 57, 59] on linear storage codes. Define the *dot product* $\boldsymbol{x} \cdot \mathbf{y}$ of two vectors $\boldsymbol{x}, \mathbf{y}$ in the m-dimensional data space $\mathbb{F}_q^m$ as

$$\boldsymbol{x} \cdot \mathbf{y} = x_1 y_1 + \cdots + x_n y_n.$$

Given vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_s \in \mathbb{F}_q^m$, let $\langle \boldsymbol{a}_1, \ldots, \boldsymbol{a}_s \rangle$ denote the *span* of these vectors in $\mathbb{F}_q^m$, that is, the subspace consisting of all vectors

$$\lambda_1 \boldsymbol{a}_1 + \ldots + \lambda_s \boldsymbol{a}_s$$

with $\lambda_1, \ldots, \lambda_s \in \mathbb{F}_q$. Then as a consequence of (1), every linear combination of the original data vector $\boldsymbol{x} \in \mathbb{F}_q$ that is stored by the DSS can be represented

by a dot product $\boldsymbol{u} \cdot \boldsymbol{x}$ for some $\boldsymbol{u} \in \mathbb{F}_q^m$. So at any moment in time, we can associate with every node an α-dimensional vector space $U = \langle \boldsymbol{u}_1, \ldots, \boldsymbol{u}_\alpha \rangle$ representing the actual data symbols $\boldsymbol{u}_1 \cdot \boldsymbol{x}, \ldots, \boldsymbol{u}_\alpha \cdot \boldsymbol{x}$ that are stored on that node; we will refer to such a subspace as the *coding space* associated with that node at that moment. Note that the storage code does not depend on *which* basis is chosen in the actual coding space U associated with the node, as long as both the node and the DSS know the actual basis. Now if, at a certain moment, a coding space U_i is associated with node i $(i = 1, \ldots, n)$, then we refer to the (multi-)set $\mathcal{U} = \{U_1, \ldots, U_n\}$ as the actual *state* of the DSS. Now suppose that node j is lost, and suppose that the DSS wants to employ the repair set consisting of the nodes with node numbers in the set $R \subseteq \{1, \ldots, n\}$. First, according to (2), the data computed by node $i \in R$ can be represented by an β-dimensional *repair space* $W_{i,j} \subseteq U_i$. And then, as a consequence of (3), the coding space U associated with the newcomer node can be any α-dimensional subspace inside the span $\langle W_{i,j} \mid i \in R \rangle$ of the repair spaces (we will refer to such a repair process as *β-repair*). According to (4), a collection of nodes with node numbers contained in a subset $K \subseteq \{1, \ldots, n\}$ is a recovery set for the original data file precisely when the span $\langle U_i \mid i \in K \rangle$ of the coding spaces associated with these nodes is equal to the entire data space $\mathbb{F}_q^m$.

According to the above description, a functional-repair storage code consists of the collections of states of the DSS, and to be a proper functional-repair code, these states must have the property that if any coding space in a state is lost, then every subset of size r of the remaining coding spaces must allow the construction by β-repair of a new coding space U such that U together with the $n - 1$ remaining coding spaces again forms a coding state of the code. A (multi-)set consisting of $n - 1$ remaining coding states when one coding space is lost will be referred to as a *proto-state* of the code. Note that an exact repair storage code will have a *single* coding state: every node is associated with a *single* coding space, which has to be reconstructed *exactly* during repair if that node is lost.

It is precisely the above precise mathematical description of linear functional-repair storage codes that greatly helps to understand and to construct such codes, and that allows us to actually *prove* that a constructed object is indeed such a code.

In practice, a DSS with n storage nodes will be used to store not just *one* data file, but will store a *large number* of data files *simultaneously*. A linear functional-repair storage code as above needs to be aware at any time of its associated coding space, where we assume that each coding space comes with a particular basis. As a consequence, it is sufficient that the node also stores an *identifier* (think a number) of its current coding space.

If the DSS stores a *large* number of files simultaneously, and if the number of coding spaces involved in the storage codes is relatively *small*, then this extra overhead is just a *very small* fraction of all the data on the node. In addition, the complexity of the repair process also depends directly on the number of distinct coding spaces involved in the storage code, thus making this number an important parameter of a linear storage code. Note that this number equals the number of nodes for an exact repair code.

Some nonexplicit functional-repair codes have been proposed for all extreme points [75], [91]. However, these codes require a large field size and do not have an efficient repair algorithm. Currently, we know of only one explicit example of a strictly functional-repair code with a small field size and efficient repair that attains an extreme point on the cut-set bound different from the MSR and MBR points [41]. This storage code is based on a certain vector space partition of $\mathbb{F}_2^5$. Here, a *vector space partition* of a vector space V is a collection of subspaces $V_1, ..., V_m$, not necessarily all of the same dimension, such that every nonzero vector in V belongs to exactly one of these subspaces. In the partition described in [41], eight 2-dimensional vector subspaces and one 3-dimensional subspace together form a vector space partition of a 5-dimensional vector space, and the eight 2-dimensional spaces can be used as coding spaces for a small functional-repair code.

Now we introduce some works related to optimal codes. Many of the code constructions in MSR and MBR points proposed in the literature are derived from well-studied geometric objects, providing a framework for generalizations and new constructions that offer more flexibility in trade-offs between desirable properties [59]. There is an explicit construction of a high-rate MSR code that allow for repairing any failed node [75]. This method uses part of the information in a node as an exact component and other parts as an auxiliary component. Upon node failure, the exact component is repaired exactly, while the auxiliary component may change, making the repair functional. Probabilistic functional-repair codes were proposed in [42], where the data for replacing a failed node is randomly generated according to specific computing rules.

The repair process discussed above addresses one failure at a time, allowing multiple failures to be handled by performing sequential repairs. Recent work has focused on developing methods to repair multiple failures simultaneously [53]. In these works, a trade-off between node capacity and repair bandwidth for multiple failures has been explored [48]. The repair process for multiple failures uses data from surviving nodes, and also other newcomer nodes that are being repaired [77]. This process involves a collaborative phase, during which nodes exchange data, and also a downloading phase similar to that employed during a single failure. For multiple failures, the

repair bandwidth takes into account both the collaborative and downloading phases. For more details and computations, refer to [78]. By leveraging the trade-off between node capacity and repair bandwidth, a scheme was proposed for repairing multiple failed nodes to achieve optimal functional repair in [57]. They show that the data can be recovered if three specific geometric conditions on the coding spaces are met by the storage and repair scheme.

1.4.3 Update Performance

Next we discuss another relevant issue in a DSS, which is update performance. In various applications, user data is frequently updated, therefore the data stored across nodes in a DSS also needs frequent updates. Since stored coded symbols depend on a number of user data symbols, an update of a data symbol requires a number of updates of coded symbols. Performing an update of coded symbols after every update of a data symbol may lead to the waste of management resources across nodes. In practice, it is common to integrate multiple updates of data symbols and perform an update for coded symbols at fixed time intervals.

Example 1.4.5 *Suppose that eight coded symbols* $(x_1, x_2+x_3, x_2, x_3+x_4, x_3, x_4+x_1, x_4, x_1+x_2)$ *are stored in eight nodes. An update (change) of data symbol* x_1 *triggers updates of three nodes, namely the nodes that store the three coded symbols* x_1, x_4+x_1, x_1+x_2 *that involve* x_1.

Let the row vector $\boldsymbol{x} = (x_1, ..., x_k)$ represent the data symbols and let $\boldsymbol{c} = (c_1, ..., c_n) = \boldsymbol{x}G$ be the coded symbols obtained by using a generator matrix G. The update efficiency of a data symbol is defined to be the number of coded symbols that need to be updated when updating the data symbol. In fact, the *update efficiency* u_i of a data symbol x_i is the weight of the i-th row $G(i)$ of G. The update efficiency u of the code is defined to be the maximum value of u_i, for $i = 1, .., n$. Update efficiency was first investigated in connection with error correction [4], then later an update efficient code ensemble was proposed in [66]. In general, an update efficient code requires a sparse generator matrix [66].

When updating the i-th data symbol, the user of the DSS knows the difference of the data $\Delta_i = x_i' - x_i$, where x_i is the old value of the i-th data symbol and x_i' is the new value. We refer to Δ_i as the *data delta* and such update strategies as *data delta updates*. In actual practice, the user can collect the data deltas in a certain time slot and provide their sum to the DSS. Write each data delta in a given time slot as $\Delta_i^{(1)}, \Delta_i^{(2)}, ...$, and the user can provide the sum $\Delta_i = \sum_j \Delta_i^{(j)}$ to the DSS, then the DSS updates its coded symbols according to the Δ_i.

Example 1.4.6 *The codeword* $\boldsymbol{c} = (x_1, x_2+x_3, x_2, x_3+x_4, x_3, x_4+x_1, x_4, x_1+x_2)$ *is stored across eight nodes in a DSS, where each node stores one coded symbol. An update of data symbol* x_1 (x_2) *triggers updates of the three nodes that store coded symbols* x_1, x_4+x_1, x_1+x_2 (x_2+x_3, x_2, x_1+x_2)*. If the DSS performs the update of both* x_1 *and* x_2 *simultaneously in a given time interval, then the five nodes that store coded symbols* $x_1, x_4+x_1, x_2+x_3, x_2, x_1+x_2$ *need to be updated. Notably, the node that stores* x_1+x_2 *needs to be updated twice. Assume* x_1 *is updated to* x_1' *and* x_2 *is updated to* x_2'*. The DSS first learns* $\Delta_1 = x_1' - x_1$ *and* $\Delta_2 = x_2' - x_2$ *and then applies these changes to each affected node. Consequently, the five nodes storing the coded symbols* $x_1, x_4+x_1, x_2+x_3, x_2, x_1+x_2$ *are updated to contain* $x_1+\Delta_1, x_4+x_1+\Delta_1, x_2+x_3+\Delta_2, x_2+\Delta_2, x_1+x_2+\Delta_1+\Delta_2$*, respectively.*

In this thesis, we construct the first family of update efficient codes having efficient local repair with availability and a short description.

1.5 Projective Planes

This thesis uses projective planes to design new codes and explain the repair process discussed in Section 1.4. A projective plane is a geometric structure that adds points at infinity to a usual plane, which is defined as follows [37]:

Definition 1.5.1 (Projective plane) *A projective plane is a 2-tuple* $(\mathcal{P}, \mathcal{L})$*, where* $\mathcal{P}$ *is a set whose elements are called* points *and where* $\mathcal{L}$ *is a collection of subsets of* $\mathcal{P}$ *called* lines*, such that the following properties hold.*
A1. *Any two distinct points are contained in exactly one line.*
A2. *Any two distinct lines intersect in exactly one point.*
A3. *There exist four points, such that no three of them are on a line.*

Axiom A1 states that every two points determine a unique line, axiom A2 states that every two lines pass through a unique point, and axiom A3 excludes some degenerate cases.

It can be shown in a $finite$ projective plane, that there exists a number q, called the *order* of the plane, with the property that every line contains $q+1$ points, and every point is contained in $q+1$ lines. Let $P_1 = (\mathcal{P}_1, \mathcal{L}_1)$ and $P_2 = (\mathcal{P}_2, \mathcal{L}_2)$ be two projective planes. An *isomorphism* from P_1 to P_2 is an ordered pair of bijections (π, σ), where $\pi : \mathcal{P}_1 \to \mathcal{P}_2$ and $\sigma : \mathcal{L}_1 \to \mathcal{L}_2$, such that whenever a point p_1 is contained in line l_1, the point $\pi(p_1)$ is contained in the line $\sigma(l_1)$. The projective plane P_1 is said to be isomorphic to the projective plane P_2 if there exists an isomorphism (π, σ) such that π maps $\mathcal{P}_1$ to $\mathcal{P}_2$ and σ maps $\mathcal{L}_1$ to $\mathcal{L}_2$.

Let $V(n,q)$ be an n-dimensional vector space over $\mathbb{F}_q$. We simply use V instead of $V(n,q)$ if the field $\mathbb{F}_q$ and the dimension n are clear from the context. The points in V are vectors $(x_1, x_2, ..., x_n)$ with $x_i \in \mathbb{F}_q$. Let $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ and $\boldsymbol{y} = (y_1, y_2, ..., y_n)$ be nonzero vectors from $V(n,q)$. We say that two nonzero vectors $\boldsymbol{x}, \boldsymbol{y} \in V(n,q)$ are *equivalent*, denoted by $\boldsymbol{x} \sim \boldsymbol{y}$ if there exists a nonzero $\lambda \in \mathbb{F}_q$ such that $\boldsymbol{y} = \lambda \boldsymbol{x}$. The relation $\sim$ is in fact an equivalence relation. The equivalence class to which $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ belongs is denoted by $(x_1 : x_2 : ... : x_n)$. The *projective space* $\mathrm{PG}(n-1,q)$ corresponding to the vector space $V(n,q)$ has as its points the equivalence classes of nonzero vectors of $V(n,q)$. The projective space inherits the projective subspaces and their containment relations from the vector space and its vector subspaces. We call $\mathrm{PG}(2,q)$ the *Desarguesian projective plane over* $\mathbb{F}_q$. It is in fact a projective plane. A *line* in $\mathrm{PG}(2,q)$ is the set of equivalence classes of points from a 2-dimensional subspace of $V(3,q)$. In other words, a line in $\mathrm{PG}(2,q)$ is a set of points $X = (x_0 : x_1 : x_2)$ in $\mathrm{PG}(2,q)$ satisfying an equation of the form $a_0x_0 + a_1x_1 + a_2x_2 = 0$ for some projective point $A = (a_0 : a_1 : a_2)$ in $\mathrm{PG}(2,q)$. It is easily verified that the projective plane $\mathrm{PG}(2,q)$ has $q^2 + q + 1$ points and $q^2 + q + 1$ lines.

The existence of projective planes of order q where q is not a prime power is an open problem [22]. The only general restriction known on the order is the Bruck–Ryser–Chowla theorem stating that if the order q is congruent to 1 or 2 mod 4, then q must be the sum of two squares; this rules out $q = 6$; the existence of a plane of order $q = 10$ was a longstanding open problem until in 1991 Lam showed that no such plane can exist [50]. The existence of a projective plane of order 12 (the next open case) is unknown.

Example 1.5.1 *The projective plane* $\mathrm{PG}(2,2)$*, also called the Fano plane, is shown in Figure 1.4. The Fano plane contains* $2^2 + 2 + 1 = 7$ *points, they are* $(0:0:1), (0:1:0), (1:0:0), (0:1:1), (1:0:1), (1:1:0), (1:1:1)$*. Every line has three points, every point is contained in three lines, and every pair of points determines a unique line that contains these points. For example,* $(0:1:1)$ *is on the line* $\{(1:0:0), (0:1:1), (1:1:1)\}$*, and the points* $\{(1:1:1), (0:0:1)\}$ *determine the line* $\{(1:1:1), (0:0:1), (1:1:0)\}$*.*

In recent years, codes derived from projective geometry have gained attention because geometric objects in projective geometry can help to understand and, sometimes, help to determine many parameters and bounds of these codes [5]. We first introduce some commonly used geometric objects [83] in $\mathrm{PG}(2,q)$. An *arc* in $\mathrm{PG}(2,q)$ is a set of points, no three on a line. An arc with size $q+1$ in $\mathrm{PG}(2,q)$ is called an *oval*. If q is odd, then an arc has at most $q+1$

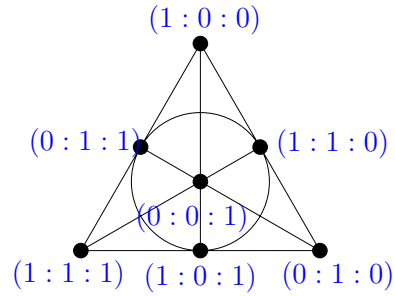

Figure 1.4: The Fano plane PG(2, 2).

points. When q is even, the situation is different. In that case, an arc has at most $q + 2$ points; an arc of the maximal size $q + 2$ is called a *hyperoval*. Furthermore, every oval is contained in a unique hyperoval. A line that meets an oval at exactly one point is called a *tangent*. When q is even, it can be shown that all tangents of an oval intersect at one point, and this point is called the *nucleus* of the oval [51]. The oval together with its nucleus forms the unique hyperoval containing the oval. A *conic* in $\mathrm{PG}(2, q)$ is the set of points X satisfying a quadratic equation of the form

$$a_{00}x_0^2 + a_{11}x_1^2 + a_{22}x_2^2 + a_{01}x_0x_1 + a_{02}x_0x_2 + a_{12}x_1x_2 = 0 \qquad (1.14)$$

for some projective point $A = (a_{00} : a_{11} : a_{22} : a_{01} : a_{02} : a_{12})$ in $\mathrm{PG}(5, q)$. If q is odd, then every oval in a $\mathrm{PG}(2, q)$ is a conic [72].

Given an arc K, the matrix $G(K)$ that has as columns the coordinate vectors of the points of the arc generates a linear MDS code $C(K)$. Conversely, the columns of a generator matrix of a linear MDS code form the set of points on an arc [8].

An incidence matrix M of a projective plane $\mathrm{PG}(2, q)$ is a $(q^2 + q + 1) \times (q^2 + q + 1)$ matrix, where rows of the matrix correspond to lines of $\mathrm{PG}(2, q)$, columns of the matrix correspond to points of $\mathrm{PG}(2, q)$, and the entries of the matrix tell us which points are on a line and which lines contain a point. The exact definition is as follows. Let $M = (m_{ij}), (i, j = 1, ..., q^2 + q + 1)$ where $m_{ij} = 1$ if point j on line i, and $m_{ij} = 0$ otherwise. Since an incidence matrix represents a projective plane, every row and every column has $q + 1$ ones. Two distinct rows have a common one in exactly one column, and two distinct columns have a common one in exactly one row. We can express these properties by the matrix equations

$$MJ = (q + 1)J, \qquad M^\top M = qI + J,$$

where I and J denotes the identity matrix of size (q^2+q+1) and the all-one matrix of size $(q^2+q+1)\times(q^2+q+1)$, respectively.

The rank of an incidence matrix of $\mathrm{PG}(2,q)$ over $\mathbb{F}_p$ where $q=p^h$ for prime p is $\frac{q^2+q}{2}+1$ [58]. Note that the rank of a matrix is invariant under a reordering of the rows and columns. Hamada computed the ranks of the incidence matrices of finite geometric designs (see [5]) over the underlying finite field. He conjectured that, among all designs with the same parameters, geometric designs achieve the minimum rank [38]. In recent works, designs with the same parameters and rank as geometric designs have been discovered. However, no design has been found that achieves a rank lower than that of geometric designs [19].

Example 1.5.2 *Two incidence matrices M_1, M_2 of the Fano plane are*

$$M_1=\begin{bmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\\1&0&0&0&1&1&0\\0&1&0&0&0&1&1\\1&0&1&0&0&0&1\end{bmatrix},\quad M_2=\begin{bmatrix}0&1&1&0&0&0&1\\0&1&0&1&1&0&0\\0&0&1&1&0&1&0\\1&0&1&0&1&0&0\\0&0&0&0&1&1&1\\1&1&0&0&0&1&0\\1&0&0&1&0&0&1\end{bmatrix}. \tag{1.15}$$

By interchanging the first column and the seventh column of M_1, the third column and the fourth column of M_1, one can obtain M_2. The rank of both M_1 and M_2 is four.

Let $\mathrm{PG}(2,q)$ denote the projective plane over the finite field $\mathbb{F}_q$, where $q=p^h$, p prime, $h\geq 1$, and let $V(3,q)$ denote the underlying vector space. Let $M=(m_{ij})$ be the incidence matrix of points and lines in the $\mathrm{PG}(2,q)$. The code associated with $\mathrm{PG}(2,q)$ is the p-ary linear code C that is the $\mathbb{F}_p$-span of the rows of the incidence matrix M. Such codes are called *incidence-matrix (projective) codes*. The dual code $C^{\perp}$ is the set of all vectors orthogonal to all codewords of C [6]. Such dual codes are called (projective) *geometric codes*.

A lot is known about the incidence-matrix codes associated with projective planes. It can be shown that the scalar multiples of the incidence vectors of the lines provide the codewords of minimal weight $q+1$ in C. Furthermore, there are no codewords with weights between $q+2$ and $2q-1$, in fact, the second smallest weight is $2q$. These codewords of weight $2q$ are actually the scalar multiples of the incidence vectors of the symmetric differences of the incidence vectors corresponding to two different lines. For a proof of these claims, see [18].

Although not directly connected to our work, we want to mention some interesting developments. We need the following definitions.

Definition 1.5.2 (Big-O notation) *Suppose $f(x)$ and $g(x)$ are two functions defined on real numbers, we write $f(x) = O(g(x))$ for $x \to \infty$ if and only if there exist constants C and D such that $|f(x)| \leq C|g(x)|$ for all $x > D$.*

Intuitively, $f(x) = O(g(x))$ means that $f(x)$ does not grow faster than $g(x)$ when $x \to \infty$. Let H be a $k \times n$ parity-check matrix where each row of H has weight $O(\sqrt{n})$. The code defined by H is called a *Moderate Density Parity-Check* (MDPC) code. For a parity-check matrix H where every row has a constant weight w_1 and every column has a constant weight w_2, the MDPC code defined by this matrix H is said to be of *type* (w_1, w_2). Therefore, the p-ary linear codes C generated by incidence matrices of $\mathrm{PG}(2, q)$ are MDPC codes of type $(q + 1, q + 1)$ [11]. MDPC codes were developed as a more secure alternative for use in McEliece cryptosystems [56]. While MDPC codes have a lower error-correction performance compared to LDPC codes, they are sufficient to provide efficient decoding algorithms [10].

In most applications, the codes that are used are binary. Let M be an incidence matrix of a projective plane $\mathrm{PG}(2, q)$, and let the code C be the $\mathbb{F}_2$-span of the rows of an incidence matrix M. If q is odd, the dual code $C^\perp$ has dimension 1 and minimum distance $q^2 + q + 1$, so these codes are not interesting when q is odd. If q is even, specifically $q = 2^h$ with $h \geq 1$, the code $C^\perp$ has dimension $2^{2h} - 3^h + 2^h$ and minimum distance $2^h + 2$ [10]. The plane uses $\mathbb{F}_q$ with q the power of a prime p, and the code uses the 0-1 incidence matrix over a field $\mathbb{F}_{p'}$ to define a p'-ary code, which is only interesting if $p' = p$. To overcome this limitation, new MDPC codes were presented [11] using projective bundles from projective planes. A projective bundle is a collection of $q^2 + q + 1$ ovals of $\mathrm{PG}(2, q)$, mutually intersecting at a common point [33]. Another incidence matrix M' can be created based on the points and ovals in the projective bundle. By combining the incidence matrices M and M', we can obtain a matrix $H = (M \mid M')$. The minimum distance of the code C' with parity-check matrix H is $q + 2$. If q is odd, the dimension of the code C' is $q^2 + q + 2$. If $q = 2^h, h \geq 1$, the dimension of code C' is $2^{2h+1} + 2^{h+1} - 2(3^h) + 1$ [10].

Next, we introduce the concept of a perfect difference set. In this thesis, we utilize perfect difference sets to describe the generator matrix of the geometric code. The use of perfect difference sets allows for a short description of the generator matrix. Let $\mathbb{Z}_v$ be the numbers $\{0, 1, ..., v-1\}$ with addition and multiplication modulo v.

Definition 1.5.3 (Difference Set) *A (v, k, λ) difference set is a subset $D \subseteq \mathbb{Z}_v$ of size k such that every element $x \in \mathbb{Z}_v \backslash \{0\}$ can be expressed by $x \equiv d_1 - d_2 (\mod v)$ for exactly λ pairs $(d_1, d_2) \in D^2$.*

Assume D is a set of k positive integers. If every nonzero integer (mod n) has a unique representation as the difference of two elements of D, then D is termed a *perfect difference set* [36]. So a perfect difference set is a difference set with $\lambda = 1$. For a difference set to exist, a necessary condition is that $k(k-1) = (v-1)\lambda$. So a necessary condition for a perfect difference set to exist is $v = k^2 - k + 1$. Note that taking $k = q + 1$, the necessary condition is $v = q^2 + q + 1$. A sufficient condition for a perfect difference set to exist is that q be a prime power [35].

Example 1.5.3 *1) A perfect difference set is $D = \{0, 1, 3\}$ in $\mathbb{Z}_7$. We can observe that if we construct a vector of length seven, assigning 1 to entries at coordinates $\{1, 2, 4\}$ and 0 to all other coordinates, we obtain a vector $(1, 1, 0, 1, 0, 0, 0)$, This vector aligns with the first row of M_1 from the previous example.*
2) Another example is perfect difference set $D = \{1, 2, 5, 7\}$ in $\mathbb{Z}_{13}$.

Difference sets are closely connected to block designs. A block design is a pair $(\mathcal{P}, \mathcal{B})$, consisting of a finite set $\mathcal{P}$ of *points* and a collection $\mathcal{B}$ of distinct subsets of $\mathcal{P}$ called *blocks*, satisfying certain regularity properties. The pair $(\mathcal{P}, \mathcal{B})$ is a $t - (v, k, \lambda)$-*design* if $|\mathcal{P}| = v$, if every block has size $k \leq v - 1$, and every set T of t points is contained in precisely λ blocks. So a difference set is a $2 - (v, k, \lambda)$-design, and a perfect difference set is a *symmetric* $2 - (v, k, 1)$-design, so with $|\mathcal{P}| = |\mathcal{B}| = v$.

1.6 Repair Locality and Availability

In this section, we will introduce two key metrics for evaluating the repair efficiency of a code: repair locality and availability.

Given a $k \times n$ generator matrix G over $\mathbb{F}_q$, a data vector $\boldsymbol{x} = (x_1, ..., x_k)$ is encoded into the codeword $\boldsymbol{c} = (c_1, ..., c_n) = \boldsymbol{x}G$. Given a codeword $\boldsymbol{h}$ in the dual code such that $h_i \neq 0$, let $R = R(\boldsymbol{h}, i) := \{j \in \{1, \ldots, n\} \setminus \{i\} \mid h_j \neq 0\}$. We refer to such a set R as a *repair set for the coded symbol in position i*. The reason for this name is that if $\boldsymbol{c}$ is a codeword, then

$$\sum_j h_j c_j = 0, \tag{1.16}$$

hence an erased codeword symbol c_i can be recovered from the collection $(c_r)_{r\in R}$ of codeword symbols in the positions of R as

$$c_i = -\sum_{j\in R} h_j c_j / h_i. \tag{1.17}$$

Let s_i be the smallest size of a repair set for the coded symbol at position i. Then the *repair locality* of the code is defined to be the maximum s of the numbers s_i. The families of codes with good local repair properties are often studied under the name of *Locally Repairable Codes* or LRC's.

LRC's have been widely studied since their proposal [39, 43]. It has been shown that the minimum distance of an LRC with repair locality r can be at most $n-k-\lceil \frac{k}{r} \rceil +2$, where n is the length of the code and k is the dimension of the code [34, 62]. The above minimum distance bound is a generalization of the Singleton bound $d \leq n-k+1$ (which is the value of this new bound when we take $r = k$). An LRC that attains this bound with equality is called an *optimal* LRC. It is possible to construct an optimal LRC from a so-called 2-layered MDS code [82]. Another construction of optimal LRC's is by using a two-level construction based on Gabidulin codes and a single parity check code [80].

In addition to having small repair sets, it is also important for a code to have many disjoint repair sets per coded symbol, since that enables simultaneous repair of copies of coded symbols [81]. Given a collection of repair sets $\mathcal{R}$, the *repair availability* of the coded symbol in position i is the maximum number t_i of pairwise disjoint repair sets for the coded symbol in position i ($i = 1, ..., n$). The *repair availability* of the code is the minimum t of the numbers t_i ($i = 1, ..., n$). A larger repair availability provides more options to repair an erasure of a coded symbol, therefore providing enhanced reliability for a DSS. Moreover, in a DSS, it is common for users to access certain parts of the data (i.e., more popular data) more often than other parts. We will discuss this issue in more detail later.

Example 1.6.1 *Let*

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{1.18}$$

be a matrix over $\mathbb{F}_2$. Using G as a generator matrix for a code C, a vector $\boldsymbol{x} = (x_1, x_2, x_3)$ is encoded as $\boldsymbol{c} = (c_1, ..., c_7) = (x_1, x_2, x_3, x_2 + x_3, x_1 + x_3, x_1 + x_2, x_1 + x_2 + x_3)$. The resulting code is systematic because every data symbol appears in the codeword. The minimal repair sets for coded symbol c_1

are $\{2,6\}$, $\{3,5\}$, $\{4,7\}$, $\{5,7\}$, $\{6,7\}$. *So the repair locality for coded symbol* c_1 *is two. It is easy to see that every coded symbol has a repair locality of at most* 2*, so the repair locality of the code is* 2*. It can be verified that the repair availability for each coded symbol* c_i *(*$i = 1, ..., 7$*) is* 3*. For example, for* c_1*, the disjoint repair sets are* $\{2,6\}$, $\{3,5\}$, $\{4,7\}$. *Therefore, the repair availability of the code is* 3*.*

Earlier Hadoop Distributed File System [12] (HDFS) employed Reed-Solomon codes, while more recent versions employ LRCs, which are theoretically optimal in terms of repair locality [70].

A notion somewhat related to LRC's is Private Information Retrieval (PIR), where the focus is on recovering a *data symbol* rather than repairing a coded symbol. A cryptographic PIR scheme allows a user to retrieve a specific data symbol from a DSS without revealing to any single node which data symbol the user is interested in [26]. Originally, PIR codes were developed to reduce the storage overhead as a coding layer on top of an information-theoretically private PIR scheme on multiple non-colluding servers. Nowadays, PIR codes are also used for example in network switches to improve the throughput rate. A t-PIR code enables a user to retrieve a specific data symbol from the information on each of t disjoint sets of nodes (these sets are called *recovery sets* for that data symbol). So here the role played by the parameter t is similar to the role played by the repair availability for an LRC. In cryptography, much research focuses on enhancing the privacy of users and nodes [20], while in coding theory, the focus is on decreasing locality and increasing availability [93]. A t-functional PIR code allows a user to retrieve a *linear combination* of data symbols from each of t disjoint sets of nodes. In coding theory, a significant amount of attention has been devoted to *batch codes* [68]. A t-batch code enables t users to retrieve t data symbols from the data on each of t disjoint sets of nodes, and a t-functional batch code enables t users to retrieve t linear combinations of data symbols from the data on each of t disjoint sets of nodes [45]. It has been shown that as the number of nodes used to store the data approaches infinity, serving PIR requests becomes as hard as serving batch requests [52].

In the context of a DSS, repair problems are not the only concern, other factors also play an important role. One significant issue is managing popular data, that is, handling a situation where certain information is frequently accessed by multiple processes in parallel. A systematic code with repair availability t can serve $t + 1$ simultaneous requests for copies of any fixed data symbol. There is a trade-off between the code rate, the minimum Hamming distance, and the availability parameters. Concerning t-PIR codes, it is possible to construct codes that support an increasing number of parallel

reads while maintaining a large code rate [65]. Another way to address hot data is to balance the popularity (that is, the number of requests) of the data when designing a DSS. If too many requests go to one node, it may cause the service to slow down. The MaxMin model spreads out the most popular items evenly across servers to avoid overloads and thereby controls the discrepancy of access requests [23]. Additionally, a method named Balanced Trades can redistribute data items without changing the total number of specific items on each node [61].

The service rate region of a DSS shows the possible data requests that the system can handle. The service rate region is crucial to understanding DSS efficiency [3]. The number of data requests that the system can support can be increased by employing coding techniques, thus improving the service rate region. It has been shown that by using a generalized version of a specific type of batch codes, known as primitive multi-set batch codes, the integer service rate region can be maximized [2]. For a given code, a DSS can select nodes and split requests to be served, a process known as a request-splitting scheme. It is possible to find the optimal request-splitting scheme to maximize the achievable service rate region for certain classes of codes, such as MDS codes. However, finding the optimal request-splitting scheme for other code families is still an open problem [1].

1.7 Group Actions

The construction of the linear FR storage code in [41] is based on the idea that an invertible linear map on the data space effectively "transfers" the available (linear) repair procedures for a potential proto-state to repair procedures for its image under this map. Now suppose that we can find linear maps that map the initial potential proto-state to each of the proto-states resulting from a repair and a subsequent node loss. Then by letting the group generated by these linear maps act on the initial proto-state, we produce a collection of proto-states, each of which automatically comes with the necessary repair procedures transferred from the initial proto-state. As a consequence, the orbit of the initial proto-state under the group in fact forms the collection of proto-states of a linear FR storage code, with the group acting as the group of linear automorphisms of the code. In this thesis, we will use a similar idea to describe the automorphisms of another functional-repair code. To describe these ideas in detail later in this thesis, we need some preparation.

Definition 1.7.1 (Group Action [25]) *A (right) group action of a group G on a set S is a map $(s, g) \mapsto s \cdot g$ from $S \times G$ to S satisfying the following:*
(1): $(s \cdot g_1) \cdot g_2 = s \cdot (g_1 \cdot g_2)$, for all $s \in S, g_1, g_2 \in G$,

(2): $s \cdot 1 = s$, for all $s \in S$.

A left group action is defined similarly. We write sg for the expression $s \cdot g$ if no confusion can arise. The action of G on S partitions S into *orbits*, where $s_1, s_2 \in S$ are in the same orbit if $s_1 \cdot g = s_2$ for some $g \in G$. We denote the orbit of an element s of S as $s \cdot G$. Formally, the relation $\sim$ defined by $s_1 \sim s_2$ if and only if $s_1 g = s_2$ for some $g \in G$ defines an equivalence relation, and the equivalence classes are the orbits. Given $s \in S$, the subset $G_s = \{g \in G \mid s \cdot g = s\}$ is called the *stabilizer* of $s \in S$. The stabilizer G_s of $s \in S$ is a subgroup of G. If G is a finite group, the orbit-stabilizer theorem states that the number of elements in the orbit $s \cdot G$ is equal to the number of distinct left cosets of G_s in G, that is, $|s \cdot G| = |G|/|G_s|$. The action is said to be *transitive* if there is only one orbit, necessarily the whole of S.

Example 1.7.1 *Consider the collection*

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.19}$$

of six 2×2 invertible matrices with non-zero determinants over $\mathbb{F}_2$. These matrices actually form a group under the operation of ordinary matrix multiplication called the general linear group *of order 2 over $\mathbb{F}_2$, denoted by* $\mathrm{GL}(2,2)$. *Let*

$$g_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad g_2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{1.20}$$

and consider the group $G_1 := \langle g_1, g_2 \rangle$ generated by g_1, g_2. Let the set S consist of all the non-zero vectors of vector space $V(2,2)$, and consider the right group action of G_1 on S by matrix multiplication. So

$$S = \{[1,0], [0,1], [1,1]\}. \tag{1.21}$$

In this group action, g_1 is a stabilizer of $[1,0], [1,1]$ and $[0,1]$, g_2 is a stabilizer of $[0,1]$. The orbits of this group action are $S_1 = \{[1,0], [1,1]\}, S_2 = \{[0,1]\}$, so this action is not transitive.

Concerning our new code, we use a group of order 677376 of linear automorphisms of a 9-dimensional vector space, i.e. the data space, acting transitively on a collection of 72 subspaces, to construct the code and to describe its automorphisms. Here, these 72 subspaces constitute the coding spaces of the code.

1.8 Author's Contributions

In this thesis, we mainly focus on the construction of update and repair codes for DSS. Our contributions can be summarized as follows.

- We present a family of storage codes with a rate close to 1/2. These codes achieve good update efficiency, have small locality and large availability, and have a short description (see below). To the best of our knowledge, these are the first storage codes that achieve all these aims simultaneously.
- We provide a pair of sparse matrices G and H for the above update efficient codes. Our codes have some freedom in the choice of G, thus enabling unequal update frequencies for different data symbols. The codes have a short description, that is, the matrix H can be described in $O(\sqrt{n}\log n)$ bits, and the matrix G can be described in $O(n\log n)$ bits.
- We probabilistically analyze the update and repair performance of the distributed storage system based on our codes. Moreover, we experimentally determine the critical probabilities for permanent server failure for small values of q.

The above contributions are based on a paper published at the 2022 IEEE International Symposium on Information Theory [47].

- We construct a new small optimal binary linear functional-repair regenerating code, with parameters $\{m,(n,k),(r,\alpha,\beta)\}=\{9,(5,4),(4,3,1)\}$. This is only the second known example of a regenerating code with both small field size and efficient repair, and with parameters in a corner point on the cut-set bound different from the MSR and MBR points. Unlike the codes constructed by Mital et.al. [57], our codes *guarantee* that repair remains possible indefinitely, not just with high probability. In addition, we have revisited the first example of such a code, a storage code with $\{m,(n,k),(r,\alpha,\beta)\}=\{5,(4,3),(3,2,1)\}$ first described in [41].
- The new storage code has a relatively simple description involving only 72 distinct coding spaces. These 72 vector spaces together with one additional space form a vector space partition of $\mathrm{V}(9,2)$ into 3-dimensional subspaces. There is a one-to-one correspondence between the vector spaces in this partition and the points in the projective plane

PG(2, 8). This correspondence provides a rather unusual, namely *non-linear* representation of PG(2, 8). This partition, when considered projectively, is actually a *plane spread*. It is left invariant by a group of order 677376 that is isomorphic to the subgroup of PGL(3, 8) fixing a point (in its action on PG(2, 8)). The efficient repair algorithm of the new storage code can be conveniently described in terms of conics in the underlying geometry.

- We provide geometric and algebraic descriptions of both the old and the new codes, together with efficient repair methods. We show that the automorphism group of the code is isomorphic to the subgroup of the full automorphism group of PG(2, 8) that fixes a point.

The above contributions combine content from the paper published at the 2024 IEEE International Symposium on Information Theory [46] and a paper submitted to the international journal Designs, Codes and Cryptography.

All authors listed in these published or submitted papers have contributed equally.

Bibliography

[1] Mehmet Aktaş, Sarah E. Anderson, Ann Johnston, Gauri Joshi, Swanand Kadhe, Gretchen L. Matthews, Carolyn Mayer, and Emina Soljanin. On the service capacity region of accessing erasure coded content. In *Proceedings 55th Annual Allerton Conference on Communication, Control, and Computing*, pages 17–24. IEEE, 2017.

[2] Mehmet Aktaş, Gauri Joshi, Swanand Kadhe, Fatemeh Kazemi, and Emina Soljanin. Service rate region: a new aspect of coded distributed system design. *IEEE Transactions on Information Theory*, 67(12):7940–7963, 2021.

[3] Gianira N. Alfarano, Altan B. Kilic, Alberto Ravagnani, and Emina Soljanin. The service rate region polytope. *arXiv preprint arXiv:2303.04021*, 2023.

[4] N.P̃rasanth Anthapadmanabhan, Emina Soljanin, and Sriram Vishwanath. Update-efficient codes for erasure correction. In *Proceedings 48th Annual Allerton Conference on Communication, Control, and Computing*, pages 376–382. IEEE, 2010.

[5] Edward F. Assmus and Jennifer D. Key. *Designs and their Codes*, volume 103. Cambridge University Press, 1992.

[6] Bhaskar Bagchi and S. P. Inamdar. Projective geometric codes. *Journal of Combinatorial Theory, Series A*, 99(1):128–142, 2002.

[7] Simeon Ball. A proof of the MDS conjecture over prime fields. In *Proceedings 3rd International Castle Meeting on Coding Theory and Applications*, volume 5, page 41. Servicio de Publicaciones= Servei de Publicacions, 2011.

[8] Simeon Ball and Michel Lavrauw. Arcs in finite projective spaces. *EMS Surveys in Mathematical Sciences*, 6(1):133–172, 2020.

[9] Carter Bancroft, Timothy Bowler, Brian Bloom, and Catherine Taylor Clelland. Long-term storage of information in DNA. *Science*, 293(5536):1763–1765, 2001.

[10] Jessica Bariffi. A finite geometry construction for MDPC-codes. Master's thesis, University of Zurich, 2020.

[11] Jessica Bariffi, Sam Mattheus, Alessandro Neri, and Joachim Rosenthal. Moderate-density parity-check codes from projective bundles. *Designs, Codes and Cryptography*, 90(12):2943–2966, 2022.

[12] Dhruba Borthakur et al. HDFS architecture guide. *Hadoop Apache Project*, 53(1-13):2, 2008.

[13] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

[14] Martin Bossert. *Channel coding for telecommunications*. John Wiley & Sons, Inc., 1999.

[15] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows Azure storage: a highly available cloud storage service with strong consistency. In *Proceedings ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[16] Paulo Cappelletti, Carla Golla, Piero Olivo, and Enrico Zanoni. *Flash memories*. Springer Science & Business Media, 2013.

[17] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2):1–26, 2008.

[18] Kevin Lee Chouinard. *Weight distributions of codes from finite planes*. University of Virginia, 1998.

[19] David Clark, Dieter Jungnickel, and Vladimir D. Tonchev. Affine geometry designs, polarities, and Hamada's conjecture. *Journal of Combinatorial Theory, Series A*, 118(1):231–239, 2011.

[20] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. Authenticated private information retrieval. In *Proceedings 32nd USENIX Security Symposium*, pages 3835–3851, 2023.

[21] Daniel J. Costello and G. David Forney. Channel coding: the road to channel capacity. *Proceedings of the IEEE*, 95(6):1150–1177, 2007.

[22] Harold Scott Macdonald Coxeter. *The real projective plane*. Springer Science & Business Media, 1992.

[23] Hoang Dau and Olgica Milenkovic. Maxminsum Steiner systems for access balancing in distributed storage. *SIAM Journal on Discrete Mathematics*, 32(3):1644–1671, 2018.

[24] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010.

[25] David Steven Dummit and Richard M. Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[26] Arman Fazeli, Alexander Vardy, and Eitan Yaakobi. Codes for distributed PIR with low storage overhead. In *Proceedings IEEE International Symposium on Information Theory*, pages 2852–2856. IEEE, 2015.

[27] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. In *Proceedings 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[28] Lester Randolph Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[29] G. David Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[30] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[31] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[32] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient

neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[33] David G. Glynn. *Finite projective planes and related combinatorial systems*. PhD thesis, University of Adelaide, 1978.

[34] Parikshit Gopalan, Cheng Huang, Huseyin Simitci, and Sergey Yekhanin. On the locality of codeword symbols. *IEEE Transactions on Information theory*, 58(11):6925–6934, 2012.

[35] R. K. Guy. Modular difference sets and error correcting codes. *C10 in Unsolved Problems in Number Theory*, pages 118–121, 1994.

[36] H. Halberstam and R. R. Laxton. Perfect difference sets. *Glasgow Mathematical Journal*, 6(4):177–184, 1964.

[37] Marshall Hall. Projective planes. *Transactions of the American Mathematical Society*, 54(2):229–277, 1943.

[38] Noboru Hamada. A characterization of some $[n, k, d; q]$-codes meeting the Griesmer bound using a minihyper in a finite projective geometry. *Discrete Mathematics*, 116(1-3):229–268, 1993.

[39] Junsheng Han and Luis Alfonso Lastras-Montano. Reliable memories with subline accesses. In *Proceedings IEEE International Symposium on Information Theory*, pages 2531–2535. IEEE, 2007.

[40] Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.

[41] Henk D. L. Hollmann and Wencin Poh. Characterizations and construction methods for linear functional-repair storage codes. In *Proceedings IEEE International Symposium on Information Theory*, pages 336–340. IEEE, 2013.

[42] Yuchong Hu, Henry C. H. Chen, Patrick P. C. Lee, and Yang Tang. Nccloud: applying network coding for the storage repair in a cloud-of-clouds. In *Proceedings USENIX Conference on File and Storage Technologies*, volume 21, 2012.

[43] Cheng Huang, Minghua Chen, and Jin Li. Pyramid codes: flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Transactions on Storage*, 9(1):1–28, 2013.

[44] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in Windows Azure storage. In *Proceedings USENIX Annual Technical Conference*, pages 15–26, 2012.

[45] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings ACM symposium on Theory of computing*, pages 262–271, 2004.

[46] Junming Ke, Henk D. L. Hollmann, and Ago-Erik Riet. A binary linear functional-repair regenerating code on 72 coding spaces related to $\mathrm{PG}(2,8)$. In *Proceedings IEEE International Symposium on Information Theory*, pages 2335–2340. IEEE, 2024.

[47] Junming Ke and Ago-Erik Riet. Update and repair efficient storage codes with availability via finite projective planes. In *Proceedings IEEE International Symposium on Information Theory*, pages 3268–3273, 2022.

[48] Anne-Marie Kermarrec, Nicolas Le Scouarnec, and Gilles Straub. Repairing multiple failures with coordinated and adaptive regenerating codes. In *Proceedings International Symposium on Networking Coding*, pages 1–6. IEEE, 2011.

[49] Ralf Koetter and Alexander Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 49(11):2809–2825, 2003.

[50] Clement W. H. Lam. The search for a finite projective plane of order 10. *The American Mathematical Monthly*, 98(4):305–318, 1991.

[51] Michel Lavrauw, Leo Storme, and Geertrui Van de Voorde. *Linear codes from projective spaces*, chapter Linear codes from projective spaces, pages 185–202. AMS Contemporary Mathematics book series. American Mathematical Society, 2010.

[52] Jüri Lember and Ago-Erik Riet. Equal requests are asymptotically hardest for data recovery. In *Proceedings IEEE International Symposium on Information Theory*, pages 3678–3683. IEEE, 2024.

[53] Shiqiu Liu and Frédérique Oggier. An overview of coding for distributed storage systems. *Network Coding and Subspace Designs*, pages 363–383, 2018.

[54] David J. C. MacKay and Radford M. Neal. Near Shannon limit performance of Low Density Parity Check Codes. *Electronics Letters*, 33(6):457–458, 1997.

[55] Robert J. McEliece. The algebraic theory of convolutional codes. *Handbook of Coding Theory*, 1998.

[56] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: new McEliece variants from moderate density parity-check codes. In *Proceedings IEEE International Symposium on Information Theory*, pages 2069–2073. IEEE, 2013.

[57] Nitish Mital, Katina Kralevska, Cong Ling, and Deniz Gündüz. Functional broadcast repair of multiple partial failures in wireless distributed storage systems. *IEEE Journal on Selected Areas in Information Theory*, 2(4):1093–1107, 2021.

[58] G. Eric Moorhouse. Bruck nets, codes, and characters of loops. *Designs, Codes and Cryptography*, 1:7–29, 1991.

[59] Siaw-Lynn Ng and Maura B. Paterson. Functional repair codes: a view from projective geometry. *Designs, Codes and Cryptography*, 87:2701–2722, 2019.

[60] J. Omura. On the Viterbi decoding algorithm. *IEEE Transactions on Information Theory*, 15(1):177–179, 1969.

[61] Chao Pan, Ryan Gabrys, Xujun Liu, Charles Colbourn, and Olgica Milenkovic. Balanced and swap-robust trades for dynamical distributed storage. In *Proceedings IEEE International Symposium on Information Theory*, pages 2385–2390. IEEE, 2022.

[62] Dimitris S. Papailiopoulos and Alexandros G. Dimakis. Locally repairable codes. *IEEE Transactions on Information Theory*, 60(10):5843–5855, 2014.

[63] Vera Pless. *Introduction to the theory of error-correcting codes*. John Wiley & Sons, 2011.

[64] Vinayak Ramkumar, S. B. Balaji, Birenjith Sasidharan, Myna Vajha, M. Nikhil Krishnan, and P. Vijay Kumar. Codes for distributed storage. *Foundations and Trends in Communications and Information Theory*, 19(4):547–813, 2022.

[65] Ankit Singh Rawat, Dimitris S. Papailiopoulos, Alexandros G. Dimakis, and Sriram Vishwanath. Locality and availability in distributed storage. *IEEE Transactions on Information Theory*, 62(8):4481–4493, 2016.

[66] Ankit Singh Rawat, Sriram Vishwanath, Abhishek Bhowmick, and Emina Soljanin. Update efficient codes for distributed storage. In *Proceedings IEEE International Symposium on Information Theory*, pages 1457–1461. IEEE, 2011.

[67] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[68] Ago-Erik Riet, Vitaly Skachek, and Eldho K. Thomas. Batch codes for asynchronous recovery of data. *IEEE Transactions on Information Theory*, 68(3):1545–1559, 2021.

[69] Ron M. Roth. Introduction to coding theory. *IET Communications*, 47(18-19):4, 2006.

[70] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: novel erasure codes for big data. *arXiv preprint arXiv:1301.3791*, 2013.

[71] Beniamino Segre. Curve razionali normali e k-archi negli spazi finiti. *Annali di Matematica Pura ed Applicata*, 39:357–379, 1955.

[72] Beniamino Segre. Ovals in a finite projective plane. *Canadian Journal of Mathematics*, 7:414–416, 1955.

[73] Gadiel Seroussi and Ron M. Roth. On MDS extensions of generalized Reed-Solomon codes. *IEEE Transactions on Information Theory*, 32(3):349–354, 1986.

[74] Nihar B. Shah, K. Vinayak Rashmi, P. Vijay Kumar, and Kannan Ramchandran. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff. *IEEE Transactions on Information Theory*, 58(3):1837–1852, 2011.

[75] Nihar B. Shah, K. Vinayak Rashmi, P. Vijay Kumar, and Kannan Ramchandran. Interference alignment in regenerating codes for distributed storage: necessity and code constructions. *IEEE Transactions on Information Theory*, 58(4):2134–2158, 2011.

[76] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[77] Kenneth W. Shum. Cooperative regenerating codes for distributed storage systems. In *Proceedings IEEE International Conference on Communications*, pages 1–5. IEEE, 2011.

[78] Kenneth W. Shum and Yuchong Hu. Cooperative regenerating codes. *IEEE Transactions on Information Theory*, 59(11):7229–7258, 2013.

[79] Aisha Siddiqa, Ahmad Karim, and Abdullah Gani. Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering*, 18:1040–1070, 2017.

[80] Natalia Silberstein, Ankit Singh Rawat, O. Ozan Koyluoglu, and Sriram Vishwanath. Optimal locally repairable codes via rank-metric codes. In *Proceedings IEEE International Symposium on Information Theory*, pages 1819–1823. IEEE, 2013.

[81] Itzhak Tamo and Alexander Barg. A family of optimal locally recoverable codes. *IEEE Transactions on Information Theory*, 60(8):4661–4676, 2014.

[82] Itzhak Tamo, Dimitris S. Papailiopoulos, and Alexandros G. Dimakis. Optimal locally repairable codes and connections to matroid theory. *IEEE Transactions on Information Theory*, 62(12):6661–6671, 2016.

[83] Joseph A. Thas. Projective geometry over a finite field. In *Handbook of incidence geometry*, pages 295–347. Elsevier, 1995.

[84] Chao Tian. Characterizing the rate region of the (4, 3, 3) exact-repair regenerating codes. *IEEE Journal on Selected Areas in Communications*, 32(5):967–975, 2014.

[85] J. H. van Lint. *Introduction to coding theory*, volume 86. Springer Science & Business Media, 1998.

[86] Branka Vucetic and Jinhong Yuan. *Turbo codes: principles and applications*, volume 559. Springer Science & Business Media, 2012.

[87] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: a quantitative comparison. In *Proceedings International Workshop on Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[88] Michael E. Whitman, Herbert J. Mattord, et al. *Principles of information security*. Thomson Course Technology Boston, MA, 2009.

[89] Jacob Wolfowitz. *Coding theorems of information theory*, volume 2nd ed. New York: Springer, 1964.

[90] Adam Wright. Worldwide global datasphere structured and unstructured data forecast, 2024–2028. Technical report, IDC Report, September 2024.

[91] Yunnan Wu. Existence and construction of capacity-achieving network codes for distributed storage. *IEEE Journal on Selected Areas in Communications*, 28(2):277–288, 2010.

[92] Min Ye and Alexander Barg. Explicit constructions of high-rate MDS array codes with optimal repair bandwidth. *IEEE Transactions on Information Theory*, 63(4):2001–2014, 2017.

[93] Yiwei Zhang, Tuvi Etzion, and Eitan Yaakobi. Bounds on the length of functional PIR and batch codes. *IEEE Transactions on Information Theory*, 66(8):4917–4934, 2020.

Chapter 2

Update and Repair Efficient Storage Codes with Availability via Finite Projective Planes

Abstract

Update performance is a common concern in modern distributed storage systems. In this work, we construct explicit update-efficient codes via finite projective planes, also having efficient local repair with availability, and a short description. We compare to other existing solutions, including block codes from convolutional codes.

We analyze the repair behavior of the codes via analogy with decoding of LDPC codes over an erasure channel, and the performance of the distributed storage system based on the codes involving updates and repairs.

Key words: erasure codes for distributed storage, update efficiency, locally repairable codes with availability, decoding of LDPC codes, erasure channel, projective geometry codes.

2.1 Introduction

For a prime q, the q-ary linear code given by the row span of the 0-1 line-point incidence matrix of the (standard, Desarguesian) finite projective plane $\mathrm{PG}(2,q)$ is denoted by $\mathrm{C}(2,q)$ (here 0 and 1 are viewed and computations are performed in the finite field $\mathbb{F}_q$; for definitions, see below).

This definition can more generally be made for prime powers q, and for q a power of 2, the dual binary code $\mathrm{C}^{\perp}(2,q)$ of the binary code $\mathrm{C}(2,q)$ has been proposed as an LDPC code [13] with geometric aspects discussed in [35]. We will instead let q be a prime, and propose to use the code $\mathrm{C}^{\perp}(2,q)$ for storage. Later LDPC codes from finite geometry over other finite fields have also been considered [36].

Note that there are constructions of quantum LDPC codes [1] based on finite geometries, where (row space of) the parity-check matrix is self-orthogonal over $\mathbb{F}_2$, whereas for our codes the generator matrix is self-orthogonal over $\mathbb{F}_q$.

Codes for storage or erasure codes considered in the literature based on finite geometry, optimized for different purposes, are often assumed to be binary, e.g. [22], [24]. Our choice of a linear code over the field $\mathbb{F}_q$ with q prime has two reasons: we can provide a sparse pair of generator and parity-check matrices G and H only for prime q for this code $\mathrm{C}^{\perp}(2,q)$; the code $\mathrm{C}^{\perp}(2,q)$ is non-trivial only as a q-ary code [14].

In fact the matrices G and H will have "moderate" density like in moderate-density parity-check codes [5], i.e. row weight of order square-root of row length. The sparseness of G provides for good update performance, while the sparseness of H together with the structure of $\mathrm{PG}(2,q)$ provides for

good locality and availability of local repair.

Storage codes for functional repair from projective geometry, optimized for repair and data recovery, were considered in [21]. Codes with locality and availability from projective geometry were considered e.g. in [23], [10], [27] and [31], based on equidistant constant dimension subspace codes in [27], and binary codes based on partial geometries in [23].

For us availability refers to multiple repair alternatives for local repair. While codes with good updation and locality have been considered in the literature [28], [17], our codes have the advantage of being entirely explicit (they do not exist for large lengths only) and achieving a good repair availability.

Block codes from convolutional codes recently proposed for distributed storage [7], [15] also have sparse matrices and a good locality, but fail to provide a good repair availability, see section 2.4.2 for some more comparison details.

Additionally, $\mathrm{PG}(2,q)$ has an explicit incidence matrix in circulant form, from which our matrix H, obtained by deleting some rows, and via H our matrix G, obtained by taking some differences of pairs of rows, will both have a short description.

Perhaps a disadvatage of our codes is that they are necessarily q-ary when derived from $\mathrm{PG}(2,q)$, for q prime, otherwise they are trivial [14]. But in a prime field, computations are conceptually easy and permit optimizations for, say, Fermat primes or primes close to powers of 2.

Perhaps another disadvantage is that data recovery from the distributed storage system is not clearly seen to be cheap for an individual data symbol, i.e. it is not clear how to recover a single data symbol from a small number of coded symbols. This is left as further work. But this assumption is not necessary for efficient updates.

We also analyze probabilistic models of updates and random (non-adversarial) failures of servers, and a scheme for the respective repair analogous to decoding of LDPC codes. We experimentally find the critical probability of permanent server failure under the repair scheme for small primes q.

2.1.1 Contribution

Our contribution is providing a family of explicit storage codes, of rate close to $1/2$, specifically $\frac{q^2+q}{2(q^2+q+1)}$, that achieve at the same time good update-efficiency, local repair with both locality and availability, and have a short description. To our knowledge, this is the first construction of an explicit family of codes for storage having good update efficiency, repair locality and repair availability at the same time.

Both update, based on data delta update, and repair only involve additions and subtractions in the field $\mathbb{F}_q$ for a prime number q. Our proposed simplified repair scheme is completely analogous to iterative decoding of LDPC codes over an erasure channel, so the respective algorithms can be used, and the relevant analyses apply. Our family of codes is a known family of q-ary linear codes of length $q^2 + q + 1$ for any prime number q, and we provide a pair of sparse matrices G and H for the codes. We have freedom in the choice of G, admitting support for unequal update frequencies for different data symbols. Achieving all these aims at the same time is novel to the best of our knowledge.

Furthermore, we probabilistically analyze the update and repair performance of the distributed storage system based on our codes, and experimentally find the critical probabilities for permanent server failure for small q.

2.2 Preliminaries

2.2.1 Distributed storage system, update efficiency, repair locality and repair availability

A distributed storage system (DSS) will be given as a linear code with a given k-by-n generator matrix G over $\mathbb{F}_q$. Given a row vector a consisting of *data symbols* $a_1, \ldots, a_k \in \mathbb{F}_q$ we store a row vector $b = aG$ of *coded symbols* $b_1, \ldots, b_n \in \mathbb{F}_q$. A coded symbol corresponds to a server, and as is standard, we can recognize when the server is down, i.e. when the coded symbol is erased.

The *support* of a vector is its set of positions with non-zero entries, and its *(Hamming) weight* is the size of its support.

The *update efficiency* u_i of a data symbol a_i is the weight of the i-th row of G, i.e. the number of coded symbols that need to be updated when updating a_i. The *update efficiency* u of the code given by G is $u = \max_i u_i$.

The *(repair) locality* of a coded symbol b_j is the minimum r_j such that b_j is a function of some other r_j coded symbols $b_{i_1}, \ldots, b_{i_r} \in \{b_1, \ldots, b_n\} \backslash \{b_j\}$. Then $\{b_{i_1}, \ldots, b_{i_r}\}$ is a *repair group* for b_j. The *repair locality* r of the code is $r = \max_j r_j$. A coded symbol b_j has (r_j, δ_j)-*locality* if there is a set of some other $r_j + \delta_j - 1$ coded symbols such that any of its subsets of size r_j is a repair group for b_j. The code has (r, δ)-*locality* if all of its coded symbols have (r, δ)-locality.

For a DSS given by a generator matrix G, a repair group $\{b_{i_1}, \ldots, b_{i_r}\}$ for coded symbol b_j is containment-wise minimal if and only if $\{j, i_1, \ldots, i_r\}$

is the support of a dual codeword (to the row space of G) that properly contains no other non-empty support of a dual codeword.

The *(repair) availability* of a coded symbol b_j is its maximum number t_j of pairwise disjoint repair groups; the *repair availability* of the code given by G is $t = \min_j t_j$.

2.2.2 Finite projective plane $\mathbf{PG}(2, q)$

The pair $(X, \mathcal{L})$, where X is a finite set and $\mathcal{L}$ is a system of its subsets, is *(an axiomatic) finite projective plane* [16] if 1) there exists a 4-element set $F \subseteq X$ with $|L \cap F| \leq 2$ for each $L \in \mathcal{L}$, 2) $|L_1 \cap L_2| = 1$ for any two distinct $L_1, L_2 \in \mathcal{L}$, 3) for any two distinct $x_1, x_2 \in X$ there is exactly one $L \in \mathcal{L}$ with $x_1, x_2 \in L$. The elements of X are called *points* and the elements of $\mathcal{L}$ are called *lines*, where a point P *lies on* a line L if $P \in L$. Then there exists q, the *order* of the projective plane, such that 1) $|L| = q + 1$ for each $L \in \mathcal{L}$, 2) $|\{L \in \mathcal{L} : P \in L\}| = q + 1$ for each $P \in X$, 3) $|X| = q^2 + q + 1$, 4) $|\mathcal{L}| = q^2 + q + 1$.

Projective planes exist for all orders q that are powers of a prime number, and are conjectured to not exist for other q.

For the *Desarguesian finite projective plane* $PG(2, q)$ of order q, points are given by 1-dimensional subspaces of the vector space $\mathbb{F}_q^3$, lines are given by 2-dimensional subspaces, and a point lies on a line if one of the subspaces is contained in the other. Formally, it has $X = \{v \in \mathbb{F}_q^3 \backslash \{0\}\}/\sim$ where $v \sim w$ if and only if $v = \lambda w$ for some $\lambda \in \mathbb{F}_q \backslash \{0\}$, and $\mathcal{L} = \{(\langle P, Q \rangle \backslash \{0\})/\sim : \bar{P}, \bar{Q} \in X, \bar{P} \neq \bar{Q}\}$ where a point $\bar{P}$ lies on a line $\bar{L}$ if there are $P \in \bar{P}$, $L \in \bar{L}$ with $P \in L$.

2.3 Linear codes from the finite projective plane $\mathbf{PG}(2, q)$

In this section, we describe the q-ary dual code $\mathrm{C}^{\perp}(2, q)$ of the q-ary linear code $\mathrm{C}(2, q)$ corresponding to the standard (Desarguesian) projective plane $\mathrm{PG}(2, q)$ for a prime number q, and note that it has both an explicit sparse generator matrix and an explicit sparse parity-check matrix. For an overview of linear codes from projective geometries, see for example [14].

2.3.1 Incidence matrix A_q

From a given projective plane of order q we can construct the transpose A_q of its $(q^2 + q + 1)$-by-$(q^2 + q + 1)$ incidence matrix as follows. The rows of

A_q are indexed by lines of the projective plane, and columns by points. An entry in the incidence matrix is 1 if the respective point lies on the respective line, and 0 otherwise.

The weight of each row and each column of A_q is $q+1$. The supports of any two rows, or any two columns, intersect in exactly one element.

Example: An incidence matrix A_3 of PG$(2,3)$ is

$$A_3 = \begin{pmatrix} 1&1&0&0&1&0&1&0&0&0&0&0&0 \\ 0&1&1&0&0&1&0&1&0&0&0&0&0 \\ 0&0&1&1&0&0&1&0&1&0&0&0&0 \\ 0&0&0&1&1&0&0&1&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0&1&0&1&0&0 \\ 0&0&0&0&0&1&1&0&0&1&0&1&0 \\ 0&0&0&0&0&0&1&1&0&0&1&0&1 \\ 1&0&0&0&0&0&0&1&1&0&0&1&0 \\ 0&1&0&0&0&0&0&0&1&1&0&0&1 \\ 1&0&1&0&0&0&0&0&0&1&1&0&0 \\ 0&1&0&1&0&0&0&0&0&0&1&1&0 \\ 0&0&1&0&1&0&0&0&0&0&0&1&1 \\ 1&0&0&1&0&1&0&0&0&0&0&0&1 \end{pmatrix}. \tag{2.1}$$

2.3.2 Circulant structure of A_q

Note that in this example the matrix has a circulant structure. For any prime power q such a circulant transposed incidence matrix exists, corresponding to PG$(2,q)$, coming from a *(Singer planar) perfect difference set* [32].

A perfect difference set is a set of residues $S = \{a_1, \ldots, a_{q+1}\} \pmod n$ such that every non-zero residue mod n can be uniquely expressed as $a_i - a_j$ for distinct $a_i, a_j \in S$. Then $n = q(q+1)+1$.

From Fourier-theoretic arguments, no row of the circulant matrix A_q can have long blocks of consecutive zeros [11], so A_q will have non-zero entries far from the main diagonal.

We describe a construction [25] of a perfect difference set modulo $q^2 + q + 1$ equivalent to Singer's original construction [32]. Let $\mathbb{F}^*_{q^3} = \langle\alpha\rangle$ be the multiplicative group of the finite field $\mathbb{F}_{q^3}$. Then $\mathbb{F}_q$ is a subfield of $\mathbb{F}_{q^3}$ with $\mathbb{F}^*_q = \langle\alpha^{q^2+q+1}\rangle$. The group $\mathcal{G} = \mathbb{F}^*_{q^3}/\mathbb{F}^*_q$ is cyclic of order q^2+q+1, corresponding to integers modulo q^2+q+1. The cosets of the $A \in \mathbb{F}^*_{q^3}$, for which the *trace* $\mathrm{Tr}(A) = A + A^q + A^{q^2}$ equals zero, form our desired perfect difference set. From this we obtain an incidence matrix A_q for PG$(2,q)$: a row of A_q will contain ones in positions indexed by an element of the perfect difference set and zeros elsewhere, and other rows of A_q will be its cyclic shifts, so A_q will be circulant.

From now on, let q be a prime, not just a prime power.

2.3.3 Parity-check matrix H for the linear code $\mathrm{C}^{\perp}(2,q)$ for q prime

The q-ary linear code $\mathrm{C}(2,q)$ is the row span of A_q over $\mathbb{F}_q$. A full-rank parity-check matrix H for its dual code $\mathrm{C}^{\perp}(2,q)$ is obtained by deleting some rows of A_q.

Moorhouse [19] proved that we get a basis for $\mathrm{C}(2,q)$ as follows, this is referred to as a Moorhouse basis; the proof is quite lengthy. Fix an arbitrary row R of A_q, and let $\{r_1,\ldots,r_{q+1}\}$ be an arbitrary ordering of the positions with a 1 in row R. Of the rows other than R, for each $i=1,\ldots,q+1$, keep arbitrary $i-1$ rows with a 1 in position r_i. Moorhouse proved this yields a full-rank $((q^2+q)/2+1)$-by-(q^2+q+1) matrix H whose rows generate the row span of A_q. We will use such an H to later construct a generator matrix G.

We will in fact use the parity-check matrix A_q (with redundant rows) for local repair.

Example: First select the first row of A_3 as the basic row R. Then delete 3 rows of A_3 which have 1 in position 1, delete 2 rows of A_3 which have 1 in position 2, delete 1 row of A_3 which has 1 in position 5. Finally we obtain a full-rank parity-check matrix H_3 from the incidence matrix A_3 of $PG(2,3)$. (Here we kept the first few rows of A_q, but this is not always possible.)

$$H_3=\begin{pmatrix}h_1\\h_2\\h_3\\h_4\\h_5\\h_6\\h_7\end{pmatrix}=\begin{pmatrix}1&1&0&0&1&0&1&0&0&0&0&0&0\\0&1&1&0&0&1&0&1&0&0&0&0&0\\0&0&1&1&0&0&1&0&1&0&0&0&0\\0&0&0&1&1&0&0&1&0&1&0&0&0\\0&0&0&0&1&1&0&0&1&0&1&0&0\\0&0&0&0&0&1&1&0&0&1&0&1&0\\0&0&0&0&0&0&1&1&0&0&1&0&1\end{pmatrix}. \tag{2.2}$$

2.3.4 Generator matrices G for the linear code $\mathrm{C}^{\perp}(2,q)$ for q prime

The code $\mathrm{C}^{\perp}(2,q)$ is generated by the differences $v-w$ (computed over $\mathbb{F}_q$) where v,w are the 0-1 incidence vectors of lines of $\mathrm{PG}(2,q)$ or the rows of A_q [2].

Thus a full-rank $((q^2+q)/2)$-by-(q^2+q+1) generator matrix G for $\mathrm{C}^{\perp}(2,q)$ can be obtained as follows. Let V be the set of the rows of any previously obtained matrix H, or more generally any full-rank matrix obtained by keeping $(q^2+q)/2+1$ rows of A_q that generates the row space of A_q. Fix any (arbitrary) tree on vertex set V, i.e. a simple undirected connected graph (V,E) without cycles. Then rows of G will be indexed by the edge set E of the tree, and a row is $v-w$ (computed over $\mathbb{F}_q$, with v and w taken in any order) where $\{v,w\}\in E$ is the respective edge.

Indeed, all pairwise differences of rows of H can be expressed via paths between vertices, so the codimension of the row span of G in $\mathrm{C}(2,q)$ is at most one (add a row of H). On the other hand, $\mathrm{C}^{\perp}(2,q)$ is generated by all pairwise differences of rows of A_q, is contained in $\mathrm{C}(2,q)$, and has codimension one in it [2].

Choosing a graph on V, we necessarily have to choose a tree. The graph needs to be connected to express all pairwise differences $u-v$. The graph needs to have no cycles for the matrix G to be full-rank: a cycle would provide a dependency of the rows corresponding to the edges of the cycle. We will use that rows of G are sparse since rows of H are sparse.

Example: We give two generator matrices $G_3^{(1)}, G_3^{(2)}$ from the parity-check matrix H_3 of $PG(2,3)$. Number the rows of H_3 from 1 to 7, so the vertex set is $V=\{1,2,3,4,5,6,7\}$.

Matrix $G_3^{(1)}$ corresponds to the balanced binary tree with edges $1-6,\ 1-7,\ 6-2,\ 6-3,\ 7-4,\ 7-5$:

$$G_3^{(1)} = \begin{pmatrix} h_1-h_6 \\ h_1-h_7 \\ h_2-h_6 \\ h_3-h_6 \\ h_4-h_7 \\ h_5-h_7 \end{pmatrix} = \begin{pmatrix} 1&1&0&0&1&-1&0&0&0&-1&0&-1&0 \\ 1&1&0&0&1&0&0&-1&0&0&-1&0&-1 \\ 0&1&1&0&0&0&-1&1&0&-1&0&-1&0 \\ 0&0&1&1&0&-1&0&0&1&-1&0&-1&0 \\ 0&0&0&1&1&0&-1&0&0&1&-1&0&-1 \\ 0&0&0&0&1&1&-1&-1&1&0&0&0&-1 \end{pmatrix}. \tag{2.3}$$

Matrix $G_3^{(2)}$ corresponds to the balanced binary tree with edges $1-6,\ 1-7,\ 6-2,\ 6-4,\ 7-3,\ 7-5$:

$$G_3^{(2)} = \begin{pmatrix} h_1-h_6 \\ h_1-h_7 \\ h_2-h_6 \\ h_4-h_6 \\ h_3-h_7 \\ h_5-h_7 \end{pmatrix} = \begin{pmatrix} 1&1&0&0&1&-1&0&0&0&-1&0&-1&0 \\ 1&1&0&0&1&0&0&-1&0&0&-1&0&-1 \\ 0&1&1&0&0&0&-1&1&0&-1&0&-1&0 \\ 0&0&1&1&0&0&0&-1&1&0&-1&0&-1 \\ 0&0&0&1&1&-1&-1&1&0&0&0&-1&0 \\ 0&0&0&0&1&1&-1&-1&1&0&0&0&-1 \end{pmatrix}. \tag{2.4}$$

The trees are different in the edges $6-3,\ 7-4$ versus $6-4,\ 7-3$, and $G_3^{(1)}$ and $G_3^{(2)}$ are different in rows 4 and 5.

2.3.5 Short description

Matrix A_q is circulant [32], so to determine it, it suffices to list the positions of ones in the first row. To determine the matrix H, we additionally need to remember the numbers of rows of A_q that we keep. For matrix G, we also need to remember the tree. So we have a short description for G.

Example: Matrix A_3 is determined by the support $\{1,2,5,7\}$ of its first row. Matrix H_3 is determined by the set $\{1,2,3,4,5,6,7\}$ of the rows of A_3 that we keep. Matrix $G_3^{(1)}$ is determined by the rows of H_3 and the tree with edges $1-6,\ 1-7,\ 6-2,\ 6-3,\ 7-4,\ 7-5$ (and the sign of its rows $+$ or $-$), analogously for matrix $G_3^{(2)}$.

2.3.6 Choice for G: support for unequal update frequencies

By the choice of a different tree for forming G, having different vertex degrees, it is possible to support unequal update frequencies of different data symbols. If the support of a column of G is bigger, roughly corresponding to a bigger vertex degree, this support could correspond to a set of less frequently updated data symbols. This is because a coded symbol corresponding to a column of G is updated precisely when updating a data symbol corresponding to a row index in the support of the column.

It is possible to start using a new tree during system operation, involving first a full repair, then recovering the full data vector a and then computing the coded symbol vector $b = aG'$ for the new matrix G' corresponding to the new tree.

Example: Suppose the data symbols $a_1, \ldots, a_6$ are updated respectively a) 2, 2, 2, 2, 2, 2; b) 1, 4, 1, 4, 1, 1; c) 4, 1, 1, 4, 1, 1 times per time unit, i.e. on average twice. Then, using matrix $G_3^{(1)}$, the maximum of update frequencies of coded symbols is a) 8; b) 7; c) 10, and for matrix $G_3^{(2)}$ it is a) 10; b) 11; c) 8 times per time unit. So in cases a) and b) matrix $G_3^{(1)}$ is preferable, while in case c) matrix $G_3^{(2)}$ is preferable.

2.4 Minimum distance of $\mathbf{C}(2,q)$ and $\mathbf{C}^{\perp}(2,q)$ for q prime, efficient update and local repair with availability, operation of the DSS

For q prime, the minimum distance of $\mathrm{C}(2,q)$ and $\mathrm{C}^{\perp}(2,q)$ is known to be respectively $q+1$ and $2q$.

Thus update efficiency $u \geq 2q$. The matrix G is already optimal for update efficiency u among generator matrices of $\mathrm{C}^{\perp}(2,q)$, since its every row has weight $2q$, so $u = 2q$.

Similarly, locality $r \geq q = q+1-1$, since repair can only be performed via the support of a codeword of $\mathrm{C}(2,q)$, i.e. one coded symbol in the support can be repaired knowing all the other coded symbols of the support; but $r = q$ because each projective line is a support of a codeword of $\mathrm{C}(2,q)$. That is, the coded symbol corresponding to a point of a line can be repaired from the coded symbols corresponding to all the other points of the line: it is repaired to minus the sum of the other coded symbols of the line (over $\mathbb{F}_q$).

Furthermore, our code achieves the maximum possible availability $t = q+1 = ((q^2+q+1)-1)/q$ for codes of this locality $r = q$ and this length

$n = q^2+q+1$: note that the pencil of all projective lines through a projective point partitions the set of the other points of the projective plane into $q+1$ sets of size q (the lines). That is, each coded symbol has $t = q+1$ pairwise disjoint alternative repair groups of size $r = q$, jointly covering all other coded symbols.

Example: From rows 1, 8, 10 and 13 of A_3, we see that the 4 alternative disjoint repair groups of size 3 for coded symbol b_1, which corresponds to the first column of A_3, are $\{b_2.b_5.b_7\}$, $\{b_8, b_9, b_{12}\}$, $\{b_3, b_{10}, b_{11}\}$, $\{b_4, b_6, b_{13}\}$.

2.4.1 Data delta updates

Updates are performed so that, when updating the i-the data symbol, the user lets the system know the "data delta" $\Delta_i = a'_i - a_i$ with the old value of the i-th data symbol a_i and the new value a'_i. Since all non-zero entries in the sparse matrix G are 1 or -1, this involves adding or subtracting Δ_i for $2q$ coded symbols of the DSS (over $\mathbb{F}_q$).

The concept of "data delta updates" assumes the user can provide the delta, i.e. the old data symbol was not lost.

Data deltas can be aggregated as $\Delta_i^{(1)}, \Delta_i^{(2)}, \ldots$ before providing their sum $\Delta_i = \sum_j \Delta_i^{(j)}$ to the DSS. In practice, the data delta may be very sparse compared to the size of the data symbol, allowing for source coding techniques.

The order of implementing the updates is not important, as addition in $\mathbb{F}_q$ is commutative and associative.

We remark that recovering a data symbol from the storage system is a separate task, and to facilitate this other properties of the code may be desirable, such as its batch-code-like properties [33], or as studied in the service rates of codes problem [12]. An analysis of the feasibility of this facilitation via optimizing the choice of G is deferred to further work.

Example: for the DSS given by the generator matrix $G_3^{(1)}$, when adding Δ_1 to data symbol a_1, we see from the first row of $G_3^{(1)}$ that we have to add Δ_1 to coded symbols b_1, b_2, b_5 and subtract Δ_1 from coded symbols b_6, b_{10}, b_{12}.

2.4.2 Repair performance

Comparison with convolutional codes and (r, δ)-locality

The block codes from convolutional codes [7], [15] obtain an (r, δ)-locality for $\delta > 1$. On the other hand, unlike our codes, the availability obtained in [7] is very poor, only 2, at the cost of an increased locality.

We can trivially obtain (r, δ)-locality (iq, i) by just taking the union of i lines/repair groups through the point that we repair; this does not capture the full power of availability.

Repair scheme

We define the following scheme of repair, with similarities with LDPC decoding:

While possible, do:

Find a projective line with exactly one erased point or coded symbol (corresponding to a server that is down), and repair this coded symbol or server from the other points (coded symbols or servers) of the line.

Specifically, the coded symbol is repaired to minus the sum of the other coded symbols of the line (over $\mathbb{F}_q$). □

We call the scheme a *success* if all servers get repaired and a *failure* otherwise.

This corresponds to sequential repair similarly to the exact locally repairable codes, e.g. [34]. However, here we do not fix the maximum number of erasures that can be sequentially repaired beforehand. As we will see, this number will be $s_q - 1$, one less than the stopping distance.

Example: In the DSS (given by $G_3^{(1)}$) with parity-check matrix A_3, suppose the set of erased coded symbols is $\{b_1, b_2, b_3, b_4, b_8\}$. At first, Algorithm A cannot repair b_1 or b_4, as each line through b_1 or b_4 (each repair group) has an erased symbol. But after repairing b_2, b_3, b_8 respectively from, say, repair groups $\{b_9, b_{10}, b_{13}\}$, $\{b_5, b_{12}, b_{13}\}$, $\{b_7, b_{11}, b_{13}\}$ (here putting total load 3 on server b_{13}), symbols b_1, b_4 can then be repaired from, say, repair groups $\{b_2, b_5, b_7\}$, $\{b_3, b_8, b_{10}\}$.

LDPC decoding

This repair scheme is exactly analogous to message-passing decoding of LDPC codes over an erasure channel for the code $\mathrm{C}^{\perp}(2, q)$, for the parity-check matrix A_q (with redundant rows compared to H), as in [13], specifically it is the Edge Removal algorithm [26]. A symmetric erasure channel is analogous to a "static" probabilistic model of repair, i.e. first each server fails independently with probability p, and later we start a repair process to repair them. So all relevant algorithms and analyses of LDPC codes apply.

Sets without tangents and stopping sets

We will subsequently compare with notions about stopping sets for LDPC codes as in the survey [26], and explain their significance.

The *stopping sets* here are precisely the *sets without tangents* in geometry, that is, sets of projective points intersecting no line in exactly one point [35], see also [8] and [13].

Since sets without tangents are closed under unions, like stopping sets, there is a unique largest set without tangents S that is a subset of a given set T of failed servers. Given a fixed set T of failed servers, the set S is precisely the set of unrepaired servers as a result of Algorithm A, analogously to stopping sets. So the result of Algorithm A is a success if and only if the set of failed servers contains no non-empty set without tangents (no non-empty stopping set).

Since any codeword of $\mathrm{C}^{\perp}(2,q)$ has dot product over $\mathbb{F}_q$ equal to zero with the incidence vector of any line, the support of any codeword of $\mathrm{C}^{\perp}(2,q)$ is a set without tangents. But for odd $q > 5$, the minimum distance of $\mathrm{C}^{\perp}(2,q)$ is $2q$, whereas there exist sets without tangents of size $2q-2$ [35].

We may ask for the smallest size s_q of a set without tangents or stopping set (*stopping distance* or *stopping number* [30], [26] of the incidence matrix A_q, see also [37]). We have $s_q \geq q+2$ since any point of a set without tangents has another point on every line through it (for q a power of two this is sharp because of the existence of hyperovals). We have $s_q \leq 2q$ as the union of two lines without the intersection point is a set without tangents. Van de Voorde [35] provides s_q for small odd q as $s_3 = 6$, $s_5 = 10$, $s_7 = 12$, $s_9 = 15$, $s_{11} = 18$. For odd $q > 5$, we have $q + \frac{1}{4}\sqrt{2q} + 2 \leq s_q \leq 2q-2$ [6]. For odd $q \geq 5$, it was recently proved that $s_q \geq q+5$ [20].

Example: In the DSS (given by $G_3^{(1)}$) with parity-check matrix A_3, suppose the set of erased coded symbols contains the set $\{b_2, b_3, b_5, b_6, b_7, b_8\}$. This is a stopping set (as it is the union of two lines through b_1, without b_1), so any line meeting this set meets it in at least 2 points. Algorithm A never manages to repair any of those servers/coded symbols.

2.4.3 Operation of the DSS: server load, probabilistic models of repair and update, and experimental results

Server load

Note that we will have to assume that a server can handle load at least 2 or 3, i.e. 2 or 3 updates or repairs at the same time. This is because, most of

the time, repair sets or the sets of updated servers will intersect in a small number of elements: the supports of two rows of A_q intersect in one element, and thus most pairs of row supports of G_q will also intersect.

"Static" probabilistic model of repair, and error floor

In the "static" probabilistic model of repair, first all coded symbols are erased or servers break independently with probability p, and then Algorithm A is started for repair.

The dominating term of the failure probability of Algorithm A can be approximated by the smallest size of a stopping set [8]. For the size of the smallest set without tangents or stopping set (stopping distance) s_q we have $\max\{q + \frac{1}{4}\sqrt{2q} + 2,\ q + 5\} \leq s_q \leq 2q - 2$ for odd $q \geq 5$, so it is slightly above the square-root of the length $q^2 + q + 1$, providing the "error floor" as in LDPC codes [29].

We should expect a "phase transition", or a "sharp threshold" at the "critical probability" as in [9], [3], and [4], that is: for p just slightly above some critical probability p_c (on the "error floor" of the "waterfall curve"), Algorithm A will fail almost surely, and for p just slightly below p_c, Algorithm A will succeed almost surely.

Experimental determination of critical probability

It is interesting to understand if this relationship between the dominating term of the failure probability of Algorithm A and the stopping distance is already apparent for small q.

For $q = 3, 5, 7, 11$, and for each failure probability p from 0% to 50% (with step 1%), we ran 500 simulations of the "static" model. We let each server fail independently with probability p, and ran Algorithm A for repair. This took about one hour on a desktop computer. We recorded the size of the largest stopping set S contained in the set of failed servers T, and whether S was empty, i.e. if the repair succeeded.

For $q = 3$, 5, 7, the set S sometimes had size $s_3 = 6$, $s_5 = 10$, $s_7 = 12$ respectively. For $q = 11$, the set S always had size 27 or larger, whereas $s_{11} = 18$. This shows that these q are too small to show this relationship, or that the number of experiments was too small.

The system failed 10% or more times, i.e. at 50 or more experiments out of the 500, starting from server failure probability about $p = 38\%$ for $q = 3$; $p = 38\%$ for $q = 5$; $p = 34\%$ for $q = 7$; $p = 29\%$ for $q = 11$. For $q = 3$, 5, 7, 11, the value $p' = s_q/(q^2 + q + 1)$ is respectively about 46%, 32%, 21%, 14% which is lower than p, as expected for such small q

(here $q = 3$ is an outlier, explainable by the large standard deviation of the number of failed servers, and the abundance of small stopping sets).

"Dynamic" probabilistic model of repair and update

It may seem desirable to simulate the DSS where servers break at random times and repairs and updates take a varying amount of time. For batch codes, server load was simulated in [18].

But we may instead rely on our previous analysis: on updating, we ignore erased servers and update working servers, separating the internal operation of the DSS (repairs; use equally updated versions or propagate updates to server being repaired) from the external operation of the DSS (updates).

2.5 Conclusions and future work

We proposed to use the q-ary linear codes $\mathrm{C}^{\perp}(2, q)$ coming from the Desarguesian projective planes $\mathrm{PG}(2, q)$, for primes q, as storage codes. We showed that they obtain a good update and repair performance. It would be desirable to understand if we can at the same time obtain a good data recovery performance for individual data symbols.

Bibliography

[1] Salah A. Aly. A class of quantum ldpc codes constructed from finite geometries. In *Proceedings IEEE Global Telecommunications Conference*, pages 1–5. IEEE, 2008.

[2] Edward F. Assmus and Jennifer D. Key. *Designs and their Codes*. Cambridge University Press, 1994.

[3] József Balogh, Béla Bollobás, Hugo Duminil-Copin, and Robert Morris. The sharp threshold for bootstrap percolation in all dimensions. *Transactions of the American Mathematical Society*, 364(5):2667–2701, 2012.

[4] Alexander Barg and Gilles Zémor. High-rate storage codes on triangle-free graphs. *arXiv preprint arXiv:2110.02378*, 2021.

[5] Jessica Bariffi, Sam Mattheus, Alessandro Neri, and Joachim Rosenthal. Moderate density parity-check codes from projective bundles. *arXiv preprint arXiv:2103.09722*, 2021.

[6] Aart Blokhuis, A. Seress, and H. A. Wilbrink. On sets of points in PG (2, q) without tangents. *Mitt. Math. Sem. Giessen*, 201:39–44, 1991.

[7] Zitan Chen and Alexander Barg. Cyclic and convolutional codes with locality. *IEEE Transactions on Information Theory*, 67(2):755–769, 2020.

[8] Changyan Di, David Proietti, I. Emre Telatar, Thomas J. Richardson, and Rüdiger L. Urbanke. Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Transactions on Information theory*, 48(6):1570–1579, 2002.

[9] Venkatesan Guruswami, Jonathan Mosheiff, Nicolas Resch, Shashwat Silas, and Mary Wootters. Sharp threshold rates for random codes. *arXiv preprint arXiv:2009.04553*, 2020.

[10] Henk D. L. Hollmann. Storage codes—coding rate and repair locality. In *Proceedings International Conference on Computing, Networking and Communications (ICNC)*, pages 830–834. IEEE, 2013.

[11] https://mathoverflow.net/users/38624/lucia. Large gaps in singer planar difference sets? MathOverflow. URL:https://mathoverflow.net/q/262064 (version: 2017-02-13).

[12] Fatemeh Kazemi, Esmaeil Karimi, Emina Soljanin, and Alex Sprintson. A combinatorial view of the service rates of codes problem, its equivalence to fractional matching and its connection with batch codes. In *Proceedings IEEE International Symposium on Information Theory*, pages 646–651. IEEE, 2020.

[13] Yu Kou, Shu Lin, and Marc P. C. Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Transactions on Information theory*, 47(7):2711–2736, 2001.

[14] Michel Lavrauw, Leo Storme, and Geertrui Van de Voorde. *Linear codes from projective spaces*, chapter Linear codes from projective spaces, pages 185–202. AMS Contemporary Mathematics book series. American Mathematical Society, 2010.

[15] Umberto Martínez-Peñas and Diego Napp. Locally repairable convolutional codes with sliding window repair. *IEEE Transactions on Information Theory*, 66(8):4935–4947, 2020.

[16] Jiří Matoušek and Jaroslav Nešetřil. *Invitation to discrete mathematics*. OUP Oxford, 2008.

[17] Arya Mazumdar, Venkat Chandar, and Gregory W. Wornell. Update-efficiency and local repairability limits for capacity approaching codes. *IEEE Journal on Selected Areas in Communications*, 32(5):976–988, 2014.

[18] Sander Mikelsaar. Empirical study of asynchronous batch codes. Master's thesis, University of Tartu, https://dspace.ut.ee/handle/10062/66455, 2019.

[19] G. Eric Moorhouse. Bruck nets, codes, and characters of loops. *Designs, Codes and Cryptography*, 1:7–29, 1991.

[20] Vito Napolitano. Small stopping sets in finite projective planes of order q. *The Art of Discrete and Applied Mathematics*, 4(3):P3–05, 2021.

[21] Siaw-Lynn Ng and Maura B. Paterson. Functional repair codes: a view from projective geometry. *Designs, Codes and Cryptography*, 87:2701–2722, 2019.

[22] Frederique Oggier and Anwitaman Datta. Self-repairing codes for distributed storage—a projective geometric construction. In *Proceedings IEEE Information Theory Workshop*, pages 30–34. IEEE, 2011.

[23] Lluis Pamies-Juarez, Henk D. L. Hollmann, and Frédérique Oggier. Locally repairable codes with multiple repair alternatives. In *Proceedings IEEE International Symposium on Information Theory*, pages 892–896. IEEE, 2013.

[24] Rina Polyanskaya, Nikita Polyanskii, and Ilya Vorobyev. Binary batch codes with improved redundancy. *IEEE Transactions on Information Theory*, 66(12):7360–7370, 2020.

[25] Alexander Pott. *Finite geometry and character theory*. Springer, 2006.

[26] Aiden Price and Joanne Hall. A survey on trapping sets and stopping sets. *arXiv preprint arXiv:1705.05996*, 2017.

[27] Netanel Raviv and Tuvi Etzion. Distributed storage systems based on intersecting subspace codes. In *Proceedings IEEE International Symposium on Information Theory*, pages 1462–1466. IEEE, 2015.

[28] Ankit Singh Rawat, Sriram Vishwanath, Abhishek Bhowmick, and Emina Soljanin. Update efficient codes for distributed storage. In *Proceedings IEEE International Symposium on Information Theory*, pages 1457–1461. IEEE, 2011.

[29] Tom Richardson. Error floors of ldpc codes. In *Proceedings Allerton conference on communication control and computing*, volume 41, pages 1426–1435. University of Illinois at Urbana-Champaign, 2003.

[30] Moshe Schwartz and Alexander Vardy. On the stopping distance and the stopping redundancy of codes. *IEEE Transactions on Information Theory*, 52(3):922–932, 2006.

[31] Natalia Silberstein, Tuvi Etzion, and Moshe Schwartz. Locality and availability of array codes constructed from subspaces. *IEEE Transactions on Information Theory*, 65(5):2648–2660, 2018.

[32] James Singer. A theorem in finite projective geometry and some applications to number theory. *Transactions of the American Mathematical Society*, 43(3):377–385, 1938.

[33] Vitaly Skachek. Batch and PIR codes and their connections to locally repairable codes. In *Network Coding and Subspace Designs*, pages 427–442. Springer, 2018.

[34] Wentu Song and Chau Yuen. Binary locally repairable codes-sequential repair for multiple erasures. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[35] Geertrui Van de Voorde. On sets without tangents and exterior sets of a conic. *Discrete Mathematics*, 311(20):2253–2258, 2011.

[36] Peter Vandendriessche. *Intertwined results on linear codes and Galois geometries*. PhD thesis, Ghent University, 2014.

[37] Yauhen Yakimenka, Vitaly Skachek, Irina E. Bocharova, and Boris D. Kudryashov. Stopping redundancy hierarchy beyond the minimum distance. *IEEE Transactions on Information Theory*, 65(6):3724–3737, 2018.

Chapter 3

A Binary Linear Functional-repair Regenerating Code on 72 Coding Spaces Related to $\mathrm{PG}(2,8)$

Abstract

Only a single example is known of a regenerating code with both small field size and efficient repair, and with parameters in a corner point on the cutset bound different from the MSR and MBR points. Here we present another such code, based on a vector space partition of a 9-dimensional binary space into 73 subspaces of dimension 3 that is strongly related to the projective plane PG$(2, 8)$; the coding spaces of the code consist of 72 of the subspaces in the partition. The new storage code comes with an efficient repair algorithm that can be described in terms of the underlying geometry.

3.1 Introduction

The global datasphere, the collection of all the data that we have generated up to now, is estimated to grow to 125ZB by the end of 2024 (here 1ZB=10^9TB and 1TB=10^3GB) see the IDC White Paper [18]. It is therefore of great importance to devise methods to store these huge amounts of data in an efficient, reliable, and economically feasible way. A distributed storage system (DSS) is designed to store information reliably over time by storing the encoded data on a distributed collection of potentially unreliable storage nodes [4]. A DSS stores data with the aid of a so-called *(distributed) storage code*.

3.1.1 Storage codes

In an often-studied model (see, e.g., [1, 4, 14, 17]), the storage code stores a file consisting of m data symbols from some field $\mathbb{F}_q$ in encoded form across n *storage nodes*, where each node contains α coded symbols from $\mathbb{F}_q$. If a node is lost, the DSS replaces this node by a *newcomer node;* subsequently, the DSS is allowed to choose *any* set of r *helper nodes* among the remaining nodes; each helper node then computes β data symbols and send these to the DSS, which uses this information to compute α new data symbols that are stored on the newcomer node. It is further required that over time, the DSS can recover the original file from the information contained in *any* set of k nodes. We will refer to such a code as a (q-ary) *regenerative storage code* with *parameters* $\{m, (n, k), (r, \alpha, \beta)\}$. In the literature, the symbols m and r are sometimes replaced by $\mathcal{M}$ or B and by d, respectively. Some models [15, 24, 26] consider the repair of multiple erasures, but in this paper we consider only the case of a single erasure.

3.1.2 Exact and functional repair

The type of repair of lost storage nodes comes in two varieties. We speak of *exact repair* if the content of a replacement node is an *exact* copy of the content of the lost storage node [19]. A storage code that exclusively employs exact repair to recover lost nodes is called an exact-repair (ER) storage code. So an ER storage code is just a single-erasure-correcting code that allows for a localized, more efficient correction method. A more subtle, and much more difficult-to-understand repair method is *functional repair*. Here, the content of the replacement node need not be an exact copy of the content of the lost node, but still the repair and recovery properties of the code are preserved over time. A storage code that does not always employ exact repair to recover lost nodes is called a *functional-repair* (FR) storage code. A FR storage code can be thought of as a *family* of codes, where an erasure in a code word from a code in the family is corrected into a code word from possibly another code in the family. We discuss *linear* FR storage codes in detail in Section 4.2, and describe an example in Example 4.2.7. For a formal definition of FR storage codes, we refer to [17, Section 3.1.1].

3.1.3 The cutset bound and regenerating codes

There are various ways to measure the efficiency of a storage code. Firstly, it is desirable to limit the *node capacity* (or *sub-packetization level*) α, the amount of data stored in each node, since this is a measure for the *coding rate* $R := m/n\alpha$ of the code. Secondly, we would also like to limit the *transport capacity* β, since this is a measure for the *repair bandwidth* $\gamma := r\beta$, the total amount of data that needs to be communicated during repair. In [4] it is shown that there is a tradeoff between those two goals, described by the *cutset bound*, which, under the usual assumption that $k \leq r$, takes the form

$$m \leq \sum_{i=0}^{k-1} \min(\alpha, (r-i)\beta). \tag{3.1}$$

For fixed m, k, r, the equality case takes the form of a piece-wise linear curve that represents the possible trade-off between the storage α and the repair bandwidth $\gamma = r\beta$. Note that we necessarily have $\alpha \geq m/k$ and $r\beta \geq \alpha$; the points on the curve where $\alpha = m/k$ with β minimal (so $\beta = \alpha/(r-k+1)$) and $\beta = \alpha/r$ with α minimal (so $\alpha = rm/(rk-(k^2-k)/2)$) are referred to as the Minimum Storage Regenerating (MSR) and Minimum Bandwidth Regenerating (MBR) points.

A *regenerating (RG)* code is an *optimal* regenerative code, so with parameters that meet the cutset bound (4.1) with *equality*, between the MSR and

MBR points [4]. It is known [22, Theorem 7] that no optimal ER RG codes can exist except in the MBR point (examples exist for any triple (n, k, r) [20]) and in a small segment starting in the MSR point; in particular, no ER RG codes exist in the corner points on the tradeoff curve other than the MSR and MBR points.

There have been some constructions of functional-repair RG codes [13,22] at the MSR point. In addition, some nonexplicit FR RG codes have been proposed [23], [25] for all corner points, which require large field size and have high-complexity repair. To the best of our knowledge, the only functional-repair RG code with parameters in corner points other than the MSR and MBR points and with efficient repair is the binary linear storage code that is constructed in [9]. Therefore, it is of great interest to find more functional-repair RG codes allowing for efficient repair.

3.1.4 Our contribution

In this paper, we are especially interested in *linear* storage codes, since such codes can be described relatively easily in terms of vector spaces. So in this paper, we adhere to the vector space view ([9, 15, 21, 26]) on linear storage codes, which we discuss in Section 4.2. Here at any moment in time, the state of a storage node in a q-ary linear code is represented by an α-dimensional vector space over the finite field $\mathbb{F}_q$ of size q; the complexity of the code is strongly related both to the size of q and to the number of different vector spaces that are required to describe the possible node states. Informally, we will speak of a *small* linear code if both the field size and the number of vector spaces involved is small. The binary linear FR RG code with parameters $\{5, (4, 3), (3, 2, 1)\}$ described in [9, Section VI] is small, as it can be described with only 8 vector spaces.

The main result of this paper is the construction of another small binary linear FR regenerating (so optimal) code, with parameters $\{m, (n, k), (r, \alpha, \beta)\} = \{9, (5, 4), (4, 3, 1)\}$. The new code has an efficient repair algorithm, and has a relatively simple description involving only 72 distinct vector spaces. These 72 *coding spaces* together with one additional space form a vector space partition [6] of $\mathbb{F}_2^9$ with 3-dimensional subspaces; in addition there is a one-to-one correspondence between the vector spaces in the partition and the points in the projective plane $\mathrm{PG}(2, 8)$. In fact, these vector spaces form what could be termed a *non-linear representation* [3] of $\mathrm{PG}(2, 8)$, as we will explain in [12]. The repair algorithm for the code can be described in terms of conics in $\mathrm{PG}(2, 8)$. The automorphism group of the code is in fact isomorphic to the subgroup of the full automorphism group $\mathrm{P\Gamma L}(3, 8)$ of $\mathrm{PG}(2, 8)$ that fixes a point. The construction itself is described in Sec-

tion 4.5; the automorphism group of the code and the proof that the code is indeed an FR RG code are the subject of Section 4.7.

In the long version [12] of this paper, we provide more details on the construction and properties of this storage code and its relation with PG(2, 8), and we also describe an efficient repair algorithm for the code.

3.2 Linear storage codes

For a formal definition of a linear exact-repair or functional-repair storage code we refer to [17]. Below, we present a more intuitive derivation of the model that will be used in this paper. See also [8, 9, 15, 16, 26] for additional background.

Informally speaking, a storage code is called *linear* if the four processes of data storage, data recovery, the generation of repair data from the helper nodes, and the generation of the replacement data from the repair data, are all linear (over some finite field). We consider the data file to be stored as a vector $\boldsymbol{x}$ in an m-dimensional vector space $V = \mathbb{F}_q^m$ over the finite field $\mathbb{F}_q$ with q elements, the *ambient space* (or *message space*) of the code. For vector spaces $A_1, \ldots, A_t \subseteq V$, we write $A_1 + \cdots + A_t$ or $\langle A_1, \ldots, A_t \rangle$ to denote their *span* or *sum* $\{a_1 + \cdots + a_t \mid a_i \in A_i, i = 1, \ldots, t\}$, and we write $\langle \boldsymbol{u}_1, \ldots, \boldsymbol{u}_t \rangle$ to denote the span of the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t$.

(a) (*Storage*) If a storage node stores the α linear combinations $c_i := u_{i1}x_1 + \cdots + u_{im}x_m$ $(i = 1, \ldots, \alpha)$, then we associate with that node the vector space $U := \langle \boldsymbol{u}_1, \ldots, \boldsymbol{u}_\alpha \rangle$ over $\mathbb{F}_q$ spanned by the α vectors $\boldsymbol{u}_i := (u_{i1}, \ldots, u_{im})$ in V, so that c_i equals the dot product $\boldsymbol{u}_i \cdot \boldsymbol{x}$ for $i = 1, \ldots, n$. We refer to U as the *storage space* associated with the node. Note that, as a consequence, any linear combination $y = \boldsymbol{u} \cdot \boldsymbol{x} := u_1x_1 + \cdots u_mx_m$ with $\boldsymbol{u} = \lambda_1\boldsymbol{u}_1 + \cdots + \lambda_\alpha\boldsymbol{u}_\alpha \in U$ can be obtained as a linear combination $\lambda_1c_1 + \cdots \lambda_\alpha c_\alpha$ of the data stored in that node.

(b) (*Recovery*) We claim that the stored data can be recovered from k storage nodes with associated vector spaces $U_1, \ldots, U_k$ precisely when the sum $U_1 + \cdots + U_k$ of these vector spaces equals V. Indeed, if there are vectors $\boldsymbol{u}_i \in U_i$ $(i = 1, \ldots, k)$ such that $\boldsymbol{u}_1 + \cdots \boldsymbol{u}_k$ equals the j-th unit vector $\boldsymbol{e}_j$ in V, then the j-th data symbol x_j can be recovered as $x_j = (\boldsymbol{u}_1 \cdot \boldsymbol{x}) + \cdots + (\boldsymbol{u}_k \cdot \boldsymbol{x})$ where the dot product $\boldsymbol{u}_i \cdot \boldsymbol{x}$ is available from the node storing U_i as explained above.

(c) (*Repair data*) A helper node with associated vector space U can generate β *helper data* symbols $h_i = \boldsymbol{r}_i \cdot \boldsymbol{x}$ $(i = 1, \ldots, \beta)$ with *repair vectors* $\boldsymbol{r}_i \in U$ for all i. We refer to the vector space $R := \langle \boldsymbol{r}_1, \ldots, \boldsymbol{r}_\beta \rangle$ spanned by the repair vectors as the *repair space* generated by the helper node.

(d) (*Replacement data*) If the replacement node receives helper data from r helper nodes $U_1, \ldots, U_r$, where the node associated with U_i contributes repair space R_i $(i = 1, \ldots, r)$, then the new node can be associated with a vector space U, that is, it can construct all the data $\boldsymbol{u} \cdot \boldsymbol{x}$ with $\boldsymbol{u} \in U$, provided that $U \subseteq R_1 + \cdots + R_r$.

The above motivates and explains the following definitions.

Definition 3.2.1 *Let $U_1, \ldots, U_k$ be subspaces of some vector space V over $\mathbb{F}_q$. We say that the collection $\{U_1, \ldots, U_k\}$ is a* recovery set *for V if the span $U_1 + \cdots + U_k$ equals V.*

Definition 3.2.2 *Let $U_1, \ldots, U_r$ and U be subspaces of some ambient vector space V over $\mathbb{F}_q$. We say that U can be obtained from the collection $\pi := \{U_1, \ldots, U_r\}$ by* β-repair*, written as $\pi \xrightarrow{\beta} U$, if we can find repair spaces $R_i \subseteq U_i$ with $\dim R_i = \beta$ for $i = 1, \ldots, r$ such that $U \subseteq R_1 + \cdots + R_r$.*

Definition 3.2.3 *A* q-ary linear exact-repair (ER) regenerative storage code *with parameters $\{m; (n, k); (r, \alpha, \beta)\}$ is a collection $U_1, \ldots, U_n$ of α-dimensional subspaces of an m-dimensional vector space V over $\mathbb{F}_q$ (referred to as* coding spaces*) such that (i) every k of the coding spaces form a recovery set for V and (ii) for every $j \in \{1, \ldots, n\}$ and for every subset $I \subseteq \{1, \ldots, n\} \setminus \{j\}$ with $|I| = r$, the space U_j can be obtained from $\{U_i \mid i \in I\}$ by β-repair.*

Example 3.2.4 Consider the code C that stores a vector $\boldsymbol{x} = (x_0, x_1, x_2, x_3) \in \mathbb{F}_2^4$ across four nodes, where node i stores the pair $x_i, x_{i+2} + x_{i+3}$ (indices considered modulo 4). So the code C has code words $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{x}) = ((x_0, x_2 + x_3), (x_1, x_3 + x_0), (x_2, x_0 + x_1), (x_3, x_1 + x_2))$, considered as a word in $(\mathbb{F}_2^2)^4$. If the contents of node 0 are lost, then a replacement node can download from helper nodes 1, 2, and 3 the data symbols $x_3 + x_0$, x_2, and x_3, respectively, which allows to *exactly* reconstruct the two lost data symbols x_0 and $x_2 + x_3$. So by the circular symmetry of the code, this is a binary linear exact-repair storage code with parameters $\{m, (n, k), (r, \alpha, \beta)\} = \{4, (4, 3), (3, 2, 1)\}$.

In the vector space description of this code, we have an ambient space $V = \mathbb{F}_2^4 = \langle \boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3 \rangle$, and with node i we associate the subspace $U_i = \langle \boldsymbol{e}_i, \boldsymbol{e}_{i+2} + \boldsymbol{e}_{i+3} \rangle$ (again with all indices considered modulo 4). If node 0 is lost, then a replacement node can download from nodes 1, 2, and 3 the repair vectors $\boldsymbol{r}_2 := \boldsymbol{e}_3 + \boldsymbol{e}_0$, $\boldsymbol{r}_3 := \boldsymbol{e}_2$, and $\boldsymbol{r}_3 := \boldsymbol{e}_3$, respectively (so the repair spaces are $R_i = \langle \boldsymbol{r}_i \rangle$ $(i = 1, 2, 3)$). Then node 0 can be repaired exactly since $U_0 \subseteq \langle \boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3 \rangle$, or, equivalently, since $U_0 \subseteq R_1 + R_2 + R_3$. □

A *functional-repair* storage code can be thought of as a *family* of codes, all with the same parameters; now if a symbol from a code word in one of the

codes is lost, we require that this symbol can be β-repaired so that the new code word again belongs to one of the codes in the family. And just as a linear exact-repair storage code can be though of as a collection of subspaces in an ambient space, we can think of a linear functional-repair code as a *family* of collections of subspaces. This leads to the following definition.

Definition 3.2.5 *A* q-ary linear functional-repair (FR) regenerative storage code *with parameters* $\{m, (n,k), (r,\alpha,\beta)\}$ *is a family* $\mathcal{S}$ *of collections* $\{U_1, \ldots, U_n\}$ *of* α*-dimensional subspaces of an* m*-dimensional vector space* V *over* $\mathbb{F}_q$*, referred to as* (coding) states*, such that (i) every* k *of the subspaces in a collection in* $\mathcal{S}$ *form a recovery set for* V *and (ii) for every state* $\{U_1, \ldots, U_n\} \in \mathcal{S}$*, for every* $j \in \{1, \ldots, n\}$ *and for every subset* $I \subseteq \{1, \ldots, n\} \setminus \{j\}$ *with* $|I| = r$*, there exists a space* U'_j *that can be obtained from* $\{U_i \mid i \in I\}$ *by* β*-repair such that* $\{U_1, \ldots, U_{j-1}, U'_j, U_{j+1}, \ldots, U_n\}$ *is again a coding state in* $\mathcal{S}$*.*

A proto-state *of the FR storage code* $\mathcal{S}$ *is a collection of the form* $\sigma \setminus \{U_i\}$ *where* $\sigma = \{U_1, \ldots, U_n\}$ *is a coding state in* $\mathcal{S}$*.*

We will refer to a subspace in a coding state as a coding space *of the storage code.*

Note that ER storage code is just a FR storage code with a *single* state. Note also that according to this definition, a regererating code is an optimal regenerative storage code.

We will informally refer to a linear FR storage code as "small" if both the field-size q and set of all the coding spaces involved in the code are (relatively) small.

We illustrate the above with an example.

Example 3.2.6 (See also [9].) Let $V = \mathbb{F}_2^5$, and let $\mathcal{P}$ consist of all collections $\pi = \{U_1, U_2, U_3\}$ of 2-dimensional subspaces of V for which $U_i \cap U_j = \{\mathbf{0}\}$ for $i \neq j$ and $U_1 + U_2 + U_3 = V$; in [9], such collections were termed $(3,1)$-*good*. It is easily seen that π is $(3,1)$-good if and only if there are $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3 \in V$ such that $V = \langle \boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{a}_1, \boldsymbol{a}_2 \rangle$, $\boldsymbol{a}_1 + \boldsymbol{a}_2 + \boldsymbol{a}_3 = \mathbf{0}$, and $U_i = \langle \boldsymbol{r}_i, \boldsymbol{a}_i \rangle$ for $i = 1, 2, 3$ (here, the vectors $\boldsymbol{a}_i$ are uniquely determined, and each $\boldsymbol{r}_i$ can be replaced by $\boldsymbol{r}_i + \boldsymbol{a}_i$). In that case, define $U_4 := \langle \boldsymbol{r}_1 + \boldsymbol{r}_2, \boldsymbol{r}_1 + \boldsymbol{r}_3 \rangle$. Obviously $\pi \xrightarrow{\beta} U_4$, and it is easily seen that $\pi \setminus \{U_i\} \cup \{U_4\} \in \mathcal{P}$ for $i = 1, 2, 3$. As a consequence, $\mathcal{P}$ is the collection of proto-states of a binary storage code with parameters $\{5, (4,3), (3,2,1)\}$; the states of the code are the collections $\sigma = \{U_1, U_2, U_3, U_4\}$ for which $\sigma \setminus \{U_i\} \in \mathcal{P}$ for all $i = 1, 2, 3, 4$. This code is optimal, with parameters in a corner point different from the MSR and MBR point, so is a FR RG code.

Obviously this is not a *small* code since *every* 2-dimensional vector space is a member of a coding state. In [9] a *small* code with the same parameters was constructed (which is actually a sub-code of this code) that involves only 8 of the 2-dimensional subspaces of V. In [12] we will discuss that code in detail. □

3.3 A new optimal FR storage code

In what follows we describe the construction of a new binary linear FR regenerating (so optimal) code, which is small since it involves only 72 coding spaces of dimension 3 in a 9-dimensional vector space over $\mathbb{F}_2$. As the code from [9], it is a member of the (r, s)-good code family with parameters satisfying $n - 1 = k = r$, $\alpha = s + 1$, $\beta = 1$, with $m = (r - s)(s + 1) + s + (s - 1) + \cdots + 1 = (s + 1)(2r - s)/2$, where r, s are integers with $r - 1 \geq s \geq 0$, see [9], [10], [11]. These codes are characterized by the property that if $\sigma = \{U_1, \ldots, U_{r+1}\}$ is a state, then for any $I \subseteq \{1, \ldots, r + 1\}$, the dimension $d(I)$ of the vector space $\sum_{i \in I} U_i$ equals $|I|(s + 1)$ if $|I| \leq r - s$, and equals $(r - s)(s + 1) + s + \cdots + (r + 1 - |I|)$ if $|I| \geq r - s$. Such codes are MSR if $s = 0$ and MBR if $r = s + 1$; in all other cases these codes are on corner points other than MSR and MBR and as a consequence their parameters cannot be realized by ER storage codes. In [11], such codes are constructed for all (r, s) with $r - 1 \geq s \geq 0$, over any field in the cases where $s = 0$ or $r - s \leq 2$, and over $\mathbb{F}_q$ when $q \geq r - 1$. The code from [9] is a small $(3, 1)$-good code, and the new code is $(4, 2)$-good, so with parameters $\{m, (n, k), (r, \alpha, \beta)\} = \{9, (5, 4), (4, 3, 1)\}$.

Our original construction of the code employed the following result, first mentioned without proof in [9].

Theorem 3.3.1 *Let $U_1, \ldots, U_n$ be α-dimensional subspaces in an ambient vector space V over a finite field $\mathbb{F}_q$ such that $V = \langle U_i \mid i \in I \rangle$ whenever $I \subseteq \{1, \ldots, n - 1\}$ with $|I| \geq k$ (*recovery*) and*

$$\pi := \{U_1, \ldots, U_{n-1}\} \xrightarrow{\beta} U_n.$$

Suppose $L_1, \ldots, L_{n-1}$ are invertible linear maps on V such that

$$L_i : \pi \longrightarrow \pi \cup \{U_n\} \setminus \{U_i\}.$$

Then with $G := \langle L_1, \ldots, L_{n-1} \rangle$, the group generated by $L_1, \ldots, L_{n-1}$, the collection $G(\pi) := \{\{g(U_1), \ldots, g(U_{n-1})\} \mid g \in G\}$ is the collection of proto-states of an $\{m, (n, k), (r, \alpha, \beta)\}$ linear FR regenerative storage code with $r = n - 1$ and $m = \dim V$.

Proof. First, if $V = \langle U_i \mid i \in I \rangle$ and $L \in G$, then

$$V = L(V) = L(\langle U_i \mid i \in I \rangle) = \langle L(U_i) \mid i \in I \rangle$$

(*transfer of recovery*).

Next, for all $L \in G$, since $\pi \xrightarrow{\beta} U_n$, by linearity of the repair process, we also have $L(\pi) \xrightarrow{\beta} L(U_n)$ (*transfer of repair*).

And finally, $L(\pi) \cup \{L(U_n)\}$ is a coding state for all $L \in G$, since for all i, we have

$$L(\pi) \cup \{L(U_n)\} \setminus \{L(U_i)\} = L(\pi \cup \{U_n\} \setminus \{U_i\}),$$

and

$$L(\pi \cup \{U_n\} \setminus \{U_i\}) = L \circ L_i(\pi) \in G(\pi).$$

□

In [12], we will show that a similar method can also be used to construct other types of storage codes.

In this section, we define the 9-dimensional ambient space V and the 72 coding spaces of the new storage code, and (briefly) its relation with PG$(2, 8)$; in the next section we describe its coding states, discuss its automorphism group, and use Theorem 3.3.1 to prove that the code is indeed a FR regenerative storage code.

Consider the finite field $\mathbb{F}_{64}$ as a 6-dimensional vector space Y over $\mathbb{F}_2$, and consider the subfield $\mathbb{F}_8$ of $\mathbb{F}_{64}$ as a 3-dimensional vector space W; then let $V := W \oplus Y$. Let α be a primitive element in $\mathbb{F}_{64}$ for which $\alpha^6 + \alpha^4 + \alpha^3 + \alpha + 1 = 0$; put $\gamma := \alpha^9$ and $\beta := \alpha^7$. Note that β has order 9 and that γ is primitive in $\mathbb{F}_8$; it is easily verified that $\gamma^3 + \gamma + 1 = 0$. The 72 coding spaces are

$$U_{b,d} := \{(w^4 + dw, bw) \mid w \in \mathbb{F}_8\} \tag{3.2}$$

for $b \in \langle \beta \rangle$ and $d \in \mathbb{F}_8$. For later use, we also define the 6-dimensional vector spaces V_b for $b \in \langle \beta \rangle$ by

$$V_b := \mathbb{F}_8 \oplus b\mathbb{F}_8. \tag{3.3}$$

We identify W with $W \oplus \{\mathbf{0}\}$ and Y with $\{\mathbf{0}\} \oplus Y$. Observe that W together with the 72 coding spaces $U_{b,d}$ form a *vector space partition*: every nonzero vector in V is contained in precisely one of these spaces. And observe also that for every $b \in \langle \beta \rangle$, we have $W \subseteq V_b$ and $U_{b,d} \subseteq V_b$ for all d, and every vector in $V_b \setminus W$ is contained in $U_{b,d}$ for exactly one d. This vector space partition is strongly related to the projective plane PG$(2, 8)$. Indeed, for

$d \in \mathbb{F}_8$ and for $b \in \langle\beta\rangle$ with $b = b_0 + b_1\alpha$ with $b_0, b_1 \in \mathbb{F}_8$, define a point $P_{b,d} \in \mathrm{PG}(2,8)$ by letting

$$P_{b,d} := (b_0 : b_1 : d); \tag{3.4}$$

with the "left-over" space W, we associate the point $P_W := (0:0:1)$. First we have the following.

Theorem 3.3.2 *Let $a, b, c \in \langle\beta\rangle$ be distinct and let $d, e, f \in \mathbb{F}_8$. Then the three points $P_{a,d}, P_{b,e}, P_{c,f}$ are collinear in* $\mathrm{PG}(2,8)$ *if and only if* $\dim U_{a,d} + U_{b,e} + U_{c,f} = 9$ *(i.e., if $U_{a,d} + U_{b,e} + U_{c,f} = V$).*

Proof. See [12]. □

As a consequence, the coding spaces form a representation of $\mathrm{PG}(2,8)$. Let $\mathcal{P}$ be the collection of the 72 coding spaces together with the "left-over" space W. We define a collection $\mathcal{L}$ of lines as follows. With each of the 9 spaces V_b ($b \in \langle\beta\rangle$), we associate a line containing W together with the 8 coding spaces $U_{b,d}$ with $d \in \mathbb{F}_8$. Then, as a consequence of Theorem 4.6.1, there are collections of 9 coding spaces, no two from the same space V_b, with the property that any three span the message space V; these collections form the lines in $\mathcal{L}$ *not* containing W. In this way, we obtain a projective plane $(\mathcal{P}, \mathcal{L})$, which is isomorphic to $\mathrm{PG}(2,8)$ by Theorem 4.6.1, with the isomorphism given by the map $U_{b,d} \mapsto P_{b,d}$ and $W \mapsto P_W$. Interestingly, this representation is *non-linear*, that is, *not* every line is induced by a subspace [3]. In [12], we describe a family of non-linear representations for projective spaces $\mathrm{PG}(m, Q)$ with Q even that generalizes this example.

3.4 The automorphism group of the code

Let $\mathrm{Tr} : \mathbb{F}_{64} \to \mathbb{F}_8$ be the trace from $\mathbb{F}_{64}$ to $\mathbb{F}_8$, defined by $\mathrm{Tr}(x) = x + x^8$. Below, we need the following result.

Lemma 3.4.1 *Let $r, s \in \mathbb{F}_{64}$ with $r^9 + s^9 \neq 0$. Then the map*

$$f_{r,s} : b \mapsto (r^2 b + s^2 b^{-1})/(r^9 + s^9 + \mathrm{Tr}(r s^8 b))$$

is a permutation on $\langle\beta\rangle$.

Proof. (Sketch) Let $b \in \langle\beta\rangle$. If $rb + sb^{-1} = 0$, then $r = s = 0$ or $s/r = b^2$ has order 9, that is, $r^9 = s^9$. So the conditions on r, s imply that $rb + sb^{-1}$ is nonzero. Now if $b \in \langle\beta\rangle$, then $b^8 = b^{-1}$, so $(rb + sb^{-1})^8 = r^8 b^{-1} + s^8 b$, hence

$$(rb + sb^{-1})(rb + sb^{-1})^8 = r^9 + s^9 + \mathrm{Tr}(r s^8 b^2);$$

we conclude that $f(b^2) = 1/(rb + sb^{-1})^7 \in \langle\beta\rangle$.

Next, if $x, y \in \langle\beta\rangle$ with $f(x^2) = f(y^2)$, then $(rx + sx^{-1})^7 = (ry + sy^{-1})^7$. After multiplication by $(rx + sx^{-1})(ry + sy^{-1})$, we obtain

$$\begin{aligned} 0 &= (r^8x^{-1} + s^8x)(ry + sy^{-1}) + (r^8y^{-1} + s^8y)(rx + sx^{-1}) \\ &= (r^9 + s^9)(x^{-1}y + xy^{-1}), \end{aligned}$$

and since $r^9 + s^9 \neq 0$, we conclude that $x/y = y/x$, that is, $x^2 = y^2$, so that $x = y$.

For further details and generalizations, we refer to [12]. □

For all $r, s, t \in \mathbb{F}_{64}$ with $r^9 + s^9 \neq 0$, let $g_{r,s,t}$ be the linear map on $V = W \oplus Y$ that maps a pair $(w, y) \in W \oplus Y$ to the pair

$$((r^9 + s^9)^4 w + \mathrm{Tr}((rs^8y)^4) + \mathrm{Tr}(ty), r^2y + s^2y^8) \tag{3.5}$$

and let the map Γ be defined by

$$\Gamma : (w, y) \mapsto (w^2, y^2). \tag{3.6}$$

Theorem 3.4.2 *The collection G of maps $g_{r,s,t}$ with $r, s, t \in \mathbb{F}_{64}$ with $r^9 + s^9 \neq 0$ is a group under composition, with order 225792. The group G fixes W and fixes the collection $U_{b,d}$ of coding spaces, in fact $g_{r,s,t} : U_{b,d} \mapsto U_{b',d'}$, where*

$$d' = ((r^9 + s^9)^4 d + \mathrm{Tr}(tb))/(r^9 + s^9 + \mathrm{Tr}(rs^8b))$$

and $b' = (r^2b + s^2b^{-1})/(r^9 + s^9 + \mathrm{Tr}(rs^8b))$.

Proof. (Sketch) We begin by noting that $r^9, s^9 \in \mathbb{F}_8$, so g is indeed a transformation on $W \oplus Y$. It is not difficult to verify that $g_{d,e,f} \circ g_{a,b,c} = g_{r,s,t}$, where

$$r = ad + b^8e, s = bd + a^8e, t = (d^9 + e^9)^4c + a^2f + b^{16}f^8;$$

note that $r^9 + s^9 = (a^9 + b^9)(d^9 + e^9)$. The number of pairs $(r, s) \in \mathbb{F}_{64}^2$ with $r^9 + s^9 \neq 0$ is $63 + 63 + 63 \cdot (63 - 9) = 2^3 \cdot 3^2 \cdot 7^2$, so $|G| = 2^9 \cdot 3^2 \cdot 7^2 = 225792$. The last claim in the theorem is also easily verified; the claim that $b' \in \langle\beta\rangle$ follows from Lemma 4.7.1. For further details, we refer to [12]. □

Now consider the group $\mathcal{G} = \langle G, \Gamma\rangle$ generated by G and Γ. Since $\Gamma, \Gamma^2 \notin G$ and $\Gamma^3 = g_{0,1,0} \in G$, we have $|\mathcal{G}| = 3|G|$. It is easily seen that Γ fixes W and maps a coding space $U_{b,d}$ to U_{b^2,d^2}. As a consequence of Theorem 4.6.1, the action of $\mathcal{G}$ on the coding spaces induces an action of $\mathcal{G}$ on $\mathrm{PG}(2, 8)$

where each element of $\mathcal{G}$ acts as an automorphism of $\mathrm{PG}(2,8)$ that fixes P_W. Now by the fundamental theorem of projective geometry, the full group of automorphisms of $\mathrm{PG}(n,q)$ is $\mathrm{P\Gamma L}(n,q)$ [7]. So in fact $\mathcal{G} \cong \mathrm{P\Gamma L}(3,8)_0$, the subgroup of $\mathrm{P\Gamma L}(3,8)$ fixing a point, as can be concluded by a comparison of the orders of these two groups.

We are now in a position where we can prove the following result.

Theorem 3.4.3 *Let $U_1 := U_{1,0}$, $U_2 := U_{\beta^7,0}$, $U_3 := U_{\beta^5,\gamma^4}$, $U_4 := U_{\beta,\gamma^4}$, and let $\pi := \{U_1, U_2, U_3, U_4\}$. Then the collection $g(\pi)$ for $g \in \mathcal{G}$ is the collection of proto-states of a binary linear FR regenerative code involving the 72 vector spaces $U_{b,d}$ ($b \in \langle\beta\rangle$, $d \in \mathbb{F}_8$) defined in (4.32), with parameters $\{m, (n,k), (r,\alpha,\beta)\} = \{9, (5,4), (4,3,1)\}$; these parameters represent an inner corner point on the cutset bound and hence the code is regenerating (i.e., optimal).*

Proof. (Sketch) We noted in Section 4.5 that the parameters correspond to an interior point on the cutset bound. Define $U_0 := U_{\beta^8,1}$. With $\boldsymbol{r}_1 := (\gamma^3, \gamma^6)$, $\boldsymbol{r}_2 := (\gamma^3, \alpha^{40})$, $\boldsymbol{r}_3 := (\gamma^5, \alpha^{35})$, and $\boldsymbol{r}_4 := (1, \alpha^{52})$, we have $\boldsymbol{r}_i \in U_i$ $(i = 1,2,3,4)$ and $U_0 = \langle \boldsymbol{r}_1 + \boldsymbol{r}_2, \boldsymbol{r}_1 + \boldsymbol{r}_3, \boldsymbol{r}_1 + \boldsymbol{r}_4 \rangle \subseteq \langle \boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_4 \rangle$. Hence $\pi \xrightarrow{1} U_0$. Next, let $C := \Gamma \circ g_{\alpha^{61},\alpha^{17},\gamma}$, let $L := \Gamma \circ g_{\alpha^{62},\alpha^{21},0}$, and let $M := \Gamma \circ g_{\alpha^{21},\alpha^{33},\gamma^6}$. It is not difficult to verify that $C : U_1 \mapsto U_2 \mapsto U_3 \mapsto U_1, U_4 \mapsto U_4, U_0 \mapsto U_0$, $L : U_1 \mapsto U_1, U_2 \mapsto U_2, U_3 \mapsto U_3, U_4 \mapsto U_0$, and $M : U_1 \mapsto U_2 \mapsto U_3 \mapsto U_4 \mapsto U_0$. So if we define $L_4 := L$ and $L_{i+1} := C^i M C^{-i}$ for $i = 0,1,2$, then the claim in the theorem follows from Theorem 3.3.1 and Theorem 4.7.2. □

With the notation as in the proof of Theorem 4.7.3, let $U_0^+ := \langle U_0, \boldsymbol{r}_1 \rangle$. Then $U_0^+ \setminus U_0$ consists of the eight points $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_4$ and $\boldsymbol{r}_5 := (\gamma^6, \alpha^{55}) \in U_{\beta^4,\gamma^5}$, $\boldsymbol{r}_6 := (\gamma^6, \alpha^{48}) \in U_{\beta^3,\gamma^5}$, $\boldsymbol{r}_7 := (1, \alpha^{24}) \in U_{\beta^6,\gamma^4}$, and $\boldsymbol{r}_8 := (\gamma^5, \alpha^{14}) \in U_{\beta^2,\gamma^4}$. It turns out that P_W and the eight projective points associated to the eight coding spaces containing the vectors in $U_0^+ \setminus U_0$ are contained in a unique conic in $\mathrm{PG}(2,8)$ with as nucleus the projective point associated with the coding space U_0. In fact, P_W and the four points in $\mathrm{PG}(2,8)$ associated with the 4 coding spaces U_i for $i = 1,2,3,4$ form a 5-arc in $\mathrm{PG}(2,8)$, and a 5-arc is contained in a unique conic [2]. As a consequence of the group action, we have the following.

Theorem 3.4.4 *Every proto-state of the FR storage code in Theorem 4.7.3 repairs to the nucleus space of the conic determined by W and the four coding spaces in the proto-state.*

In [12], we describe an efficient algebraic repair algorithm for this FR storage code.

3.5 Conclusions

We have constructed a regenerating (i.e., optimal) binary linear functional-repair storage code with parameters $\{m, (n, k), (r, \alpha, \beta)\} = \{9, (5, 4), (4, 3, 1)\}$; this code does not employ any symbol extension (i.e., $\beta = 1$). These parameters represent a corner point on the cutset bound different from the MSR and MSB points, and according to [22] these parameters cannot be realized by an exact-repair storage code. The code is "small": it is binary and the coding states are made up from a collection of only 72 coding spaces. These coding spaces together with one additional space form a *vector space partition* of the 9-dimensional ambient space $V \cong \mathbb{F}_2^9$ of the code into 73 3-dimensional subspaces, and constitute a *non-linear* representation of the projective plane $\mathrm{PG}(2, 8)$ of order 8 over $\mathbb{F}_8$. The code (and the vector space partition) has an automorphism group $\mathcal{G}$ of order 677376 consisting of invertible linear maps on V that is isomorphic to the subgroup of the full automorphism group $\mathrm{P\Gamma L}(3, 8)$ of $\mathrm{PG}(2, 8)$ that fixes a point. After the small $(3, 1)$-good binary linear code with parameters $\{m, (n, k), (r, \alpha, \beta)\} = \{5, (4, 3), (3, 2, 1)\}$ involving only 8 coding spaces from [9], this is only the second small optimal storage code known with parameters realizing a corner point on the cutset bound different from the MSR and MBR points.

3.6 Acknowledgment

This research was supported by the Estonian Research Council grants PRG49 and PSG114.

Bibliography

[1] S.B. Balaji, M.N. Krishnan, M Vajha, V. Ramkumar, B. Sasidharan, P.V. Kumar, *Erasure coding for distributed storage: an overview*, Sci. China Inf. Sci. 61, 100301 (2018).

[2] J. Bierbrauer, *The maximal size of a 3-arc in* $\mathrm{PG}(2,8)$, J. Comb. Math. and Comb. Comp., vol. 45 (2003), 147–161.

[3] R.H. Bruck and R.C. Bose, *Linear representations of projective planes in projective spaces*, J. of Algebra 4, 1966, 117–172.

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

[5] R. Elman, N. Karpenko, A. Merkurjev, The Algebraic and Geometric Theory of Quadratic Forms, AMS, 2008.

[6] O. Heden, *A survey of the different types of vector space partitions*, Discrete Math., Alg. and Appl., vol. 4, no. 1, 2012.

[7] J.W.P. Hirschfeld, Projective geometries over finite fields, 2nd ed., Oxford University Press, Oxford, 1998.

[8] H.D.L. Hollmann, *Storage codes – coding rate and repair locality*, in *2013 International Conference on Computing, Networking and Communications, Invited Position Papers (ICNC'13 - Invited)*, San Diego, USA, Jan. 2013.

[9] H.D.L. Hollmann, W. Poh, *Characterizations and construction methods for linear functional-repair storage codes*, Proceedings 2013 IEEE International Symposium on Information Theory, ISIT-2013, Istanbul, Turkey, 7–12 July 2013, pp. 336–340.

[10] H.D.L. Hollmann, *On the minimum storage overhead of distributed storage codes with a given repair locality*, Proceedings 2014 IEEE International Symposium on Information Theory, ISIT-2014, Honolulu, Hawai'i, USA, June 29 through July 4 2014, pp. 1041–1045.

[11] H.D.L. Hollmann, *A class of optimal interior-point linear functional-repair storage codes*, submitted to IEEE Trans. on Inform. Theory.

[12] H.D.L. Hollmann, J. Ke, A.-E. Riet, *An optimal binary linear functional-repair storage code with efficient repair related to* $\mathrm{PG}(2, 8)$, to be submitted to IEEE Trans. on Inform. Theory.

[13] Y. Hu, H. C.H. Chen, P.P.C. Lee, and Y. Tang, *NCCloud: Applying network coding for the storage repair in a cloud-ofclouds*, in Proc. 10th USENIX conference on File and Storage Technologies, San Jose, CA, USA, 2012, p. 21, 2012.

[14] S.Q. Liu, F. Oggier, *An overview of coding for distributed storage systems*, in: Network Coding and Subspace Designs. Berlin: Springer, 2018, 363–383.

[15] N. Mital, K. Kralevska, C. Ling and D. Gündüz, *Functional Broadcast Repair of Multiple Partial Failures in Wireless Distributed Storage Systems*, IEEE J. Sel. Areas Inform. Theory, vol. 2, no. 4, pp. 1093–1107, Dec. 2021.

[16] S.-L. Ng, M.B. Paterson, *Functional repair codes: a view from projective geometry*, Des., Codes and Crypt., vol. 87, iss. 11, Nov. 2019. 2701–2722.

[17] V. Ramkumar, S.B. Balaji, B. Sasidharan, M. Vajha, M. Nikhil Krishnan, P. Vijay Kumar, *Codes for Distributed Storage*, Foundations and Trends in Communications and Information Theory, Vol. 19: No. 4, pp. 547–813, 2022.

[18] D. Reinsel, J. Gantz, J. Rydning, *The Digitization of the World From Edge to Core*, An IDC White Paper, november 2018.

[19] K.V. Rashmi, N.B. Shah, P.V. Kumar, K. Ramchandran, *Explicit construction of optimal exact regenerating codes for distributed storage*, 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 1243-1249, September, 2009.

[20] K.V. Rashmi, N.B. Shah, P.V. Kumar, *Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a*

Product-Matrix Construction, IEEE Trans. on Inform. Theory, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[21] N.B. Shah, K.V. Rashmi, P.V. Kumar, K. Ramchandran, *Explicit codes minimizing repair bandwidth for distributed storage*, 2010 IEEE Information Theory Workshop on Information Theory (ITW 2010, Cairo), Cairo, Egypt, 2010, pp. 1–5.

[22] N.B. Shah, K.V. Rashmi, P.V. Kumar, K. Ramchandran, *Distributed Storage Codes with Repair-by-Transfer and Non-achievability of Interior Points on the Storage-Bandwidth Tradeoff*, IEEE Trans. on Inform. Theory, vol. 58, no. 3, March 2012, 1837–1852.

[23] N.B. Shah, K.V. Rashmi, P.V. Kumar, K. Ramchandran, *Interference Alignment in Regenerating Codes for Distributed Storage: Necessity and Code Constructions*, IEEE Trans. on Inform. Theory, vol. 58, no. 4, April 2012, 2134–12158.

[24] K. W. Shum, *Cooperative regenerating codes for distributed storage systems*, in Proc. IEEE Int. Conf. Commun. (ICC), 2011, pp. 1–5.

[25] Y. Wu, *Existence and construction of capacity-achieving network codes for distributed storage*, IEEE J. Sel. Areas Commun., vol. 28, no. 2, 277–288, Feb. 2010.

[26] M. Zorgui, Z. Wang, *Centralized multi-node repair regenerating codes*, IEEE Trans. Inf. Theory, vol. 65, no. 7, pp. 4180–4206, Jul. 2019.

Summary

Codes for Distributed Storage

Vast amounts of data are generated from sources like video, voice, and text, and these amounts are still growing yearly, requiring sustainable storage systems for data integrity and availability. Distributed Storage Systems (DSS's) offer a low-cost, reliable, and stable solution. A DSS stores data across potentially unreliable storage nodes by adding redundancy to ensure system reliability. Efficient update and repair mechanisms are critical for maintaining stability, especially during node failures.

This thesis first constructs update-efficient codes based on finite projective planes, featuring efficient local repair and large repair availability. Sparse generator and parity-check matrices are provided that also allow efficient updates with flexibility in update frequencies. The performance of a DSS using these codes is also evaluated.

Two key concepts in storage codes are node capacity and repair bandwidth, where the possible trade-offs between them are determined by the cut-set bound. Most research focuses on exact repair, where lost node data is recovered exactly. However, functional repair, which does not recover lost node content exactly but still maintains data integrity, allows to reach otherwise unattainable points on the cut-set bound. This thesis constructs a new optimal functional-repair code based on a vector space partition in 9-dimensional binary space, strongly related to the projective plane $\mathrm{PG}(2, 8)$. This storage code also has an efficient repair algorithm which is described both geometrically and algebraically.

Kokkuvõte

Hajustalletuskoodid

Allikatest nagu video, audio ja tekst genereeritakse suuri andmehulki, ja need hulgad endiselt aastast aastasse suurenevad, nii et nende talletamine nõuab andmete terviklikkust ja kättesaadavust garanteerivaid jätkusuutlikke andmetalletussüsteeme. Hajustalletussüsteemid (HTS-d) pakuvad madala hinnaga, töökindlat ja stabiilset lahendust. HTS talletab andmeid potentsiaalselt ebatöökindlates talletussõlmedes liiaga, mis garanteerib süsteemi töökindluse. Efektiivsed andmeuuenduse ja parandusmehhanismid on stabiilsuse hoidmiseks kriitilised, eriti sõlmede tõrgete korral.

See dissertatsioon esiteks konstrueerib uuendusefektiivsed koodid, mis põhinevad lõplikel projektiivsetel tasanditel ja mis võimaldavad efektiivset lokaalset ja suure saadavusega parandust. Konstrueeritakse hõredad genereeriv- ja paarsuskontrollimaatriksid, mis võimaldavad ka efektiivseid ja paindlike sagedustega uuendusi. Samuti hinnatakse neid koode kasutava HTS-i jõudlust.

Kaks talletuskoodide võtmesuurust on sõlmemahtuvus ja paranduse läbilaskevõime, mille omavahelist vastandlikkust kirjeldab lõiketõke. Enamus uurimusi keskendub täpsele parandusele, kus mittetöötava sõlme andmed taastatakse täpselt. Seevastu funktsionaalne parandus, mille korral ei taastata mittetöötava sõlme andmeid täpselt, seejuures siiski tagades andmete terviklikkust, võimaldab saavutada muul viisil mittesaavutatavaid punkte lõiketõkkel. Dissertatsioon konstrueerib uue optimaalse ja funktsionaalse parandusega koodi, mis põhineb vektorruumitükeldusel 9-mõõtmelises binaarses ruumis, ja millel on tugev seos projektiivse tasandiga $\mathrm{PG}(2, 8)$. Sellel talletuskoodil on ka efektiivne parandusalgoritm, mida kirjeldatakse geomeetriliselt ja algebraliselt.

Curriculum Vitae

NAME: Junming Ke
DATE AND PLACE OF BIRTH: 11/07/1994, China
CITIZENSHIP: China
PHONE: +372 56702416
EMAIL: junming.ke@ut.ee
LANGUAGES: Chinese and English
FIELD OF SCIENTIFIC INTEREST: Coding theory

EDUCATION
2012 - 2016 Bachelor of Science in Mathematics, Zhejiang University of Technology, China
2016 - 2019 Master of Science in Computer Science and Technology, Shandong University, China
2020 - 2024 Doctoral studies in Mathematics, University of Tartu, Estonia

WORK EXPERIENCE
2015 - 2016 Research assistant at Zhejiang University
2019 - 2020 Visiting fellow at Singapore University of Technology and Design

AWARDS
2022 The Fund of XRP Ledger Trust Scholarship
2015 Meritorious Winner of the Mathematical Contest in Modeling, COMAP.

Elulookirjeldus

NIMI: Junming Ke
SÜNNIKUUPÄEV JA -KOHT: 11/07/1994, Hiina
KODAKONDSUS: Hiina
TELEFON: +372 56702416
EMAIL: junming.ke@ut.ee
KEELED: hiina ja inglise keel
TEADUSLIK HUVIALA: kodeerimisteooria

HARIDUS
2012 - 2016 bakalaureusekraad matemaatikas, Zhejiangi Tehnikaülikool, Hiina
2016 - 2019 arvutiteaduse ja tehnoloogia magistrikraad, Shandongi Ülikool, Hiina
2020 - 2024 matemaatika doktoriõpe, Tartu Ülikool, Eesti

TÖÖKOGEMUS
2015 - 2016 teadusassistent Zhejiangi Ülikoolis
2019 - 2020 Singapuri Tehnika- ja Disainiülikooli külalisuurija

AUHINNAD
2022 XRP Ledger Trusti stipendium
2015 modelleerimise matemaatikavõistluse COMAP teeneline võitja.

List of original publications

[1] Henk D.L. Hollmann, Junming Ke, and Ago-Erik Riet, An optimal binary linear functional-repair storage code with efficient repair related to PG(2,8). *Submitted to* Designs, Codes and Cryptography.

[2] Junming Ke, Henk D.L. Hollmann, and Ago-Erik Riet, A binary linear functional-repair regenerating code on 72 coding spaces related to PG(2,8). In *Proceedings IEEE International Symposium on Information Theory*, IEEE, 2024.

[3] Junming Ke, and Ago-Erik Riet. Update and repair efficient storage codes with availability via finite projective planes. In *Proceedings IEEE International Symposium on Information Theory*, IEEE, 2022.

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. **Mati Heinloo.** The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991, 23 p.
2. **Boris Komrakov.** Primitive actions and the Sophus Lie problem. Tartu, 1991, 14 p.
3. **Jaak Heinloo.** Phenomenological (continuum) theory of turbulence. Tartu, 1992, 47 p.
4. **Ants Tauts.** Infinite formulae in intuitionistic logic of higher order. Tartu, 1992, 15 p.
5. **Tarmo Soomere.** Kinetic theory of Rossby waves. Tartu, 1992, 32 p.
6. **Jüri Majak.** Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992, 32 p.
7. **Ants Aasma.** Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993, 32 p.
8. **Helle Hein.** Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993, 28 p.
9. **Toomas Kiho.** Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994, 23 p.
10. **Arne Kokk.** Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995, 165 p.
11. **Toomas Lepikult.** Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995, 93 p, (in Russian).
12. **Sander Hannus.** Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995, 74 p, (in Russian).
13. **Sergei Tupailo.** Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996, 134 p.
14. **Enno Saks.** Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996, 96 p.
15. **Valdis Laan.** Pullbacks and flatness properties of acts. Tartu, 1999, 90 p.
16. **Märt Põldvere.** Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999, 74 p.
17. **Jelena Ausekle.** Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999, 72 p.
18. **Krista Fischer.** Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999, 124 p.
19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.

23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. Töö kaitsmata.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.
43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.

49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
68. **Olga Liivapuu.** Graded q-differential algebras and algebraic models in noncommutative geometry. Tartu 2011, 112 p.
69. **Aleksei Lissitsin.** Convex approximation properties of Banach spaces. Tartu 2011, 107 p.
70. **Lauri Tart.** Morita equivalence of partially ordered semigroups. Tartu 2011, 101 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.

72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.
74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
75. **Nadežda Bazunova.** Differential calculus $d^3 = 0$ on binary and ternary associative algebras. Tartu 2011, 99 p.
76. **Natalja Lepik.** Estimation of domains under restrictions built upon generalized regression and synthetic estimators. Tartu 2011, 133 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
80. **Marje Johanson.** $M(r, s)$-ideals of compact operators. Tartu 2012, 103 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
82. **Vitali Retšnoi.** Vector fields and Lie group representations. Tartu 2012, 108 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
85. **Erge Ideon**. Rational spline collocation for boundary value problems. Tartu, 2013, 111 p.
86. **Esta Kägo.** Natural vibrations of elastic stepped plates with cracks. Tartu, 2013, 114 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
88. **Boriss Vlassov.** Optimization of stepped plates in the case of smooth yield surfaces. Tartu, 2013, 104 p.
89. **Elina Safiulina.** Parallel and semiparallel space-like submanifolds of low dimension in pseudo-Euclidean space. Tartu, 2013, 85 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
93. **Kerli Orav-Puurand.** Central Part Interpolation Schemes for Weakly Singular Integral Equations. Tartu, 2014, 109 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.

95. **Kaido Lätt.** Singular fractional differential equations and cordial Volterra integral operators. Tartu, 2015, 93 p.
96. **Oleg Košik.** Categorical equivalence in algebra. Tartu, 2015, 84 p.
97. **Kati Ain.** Compactness and null sequences defined by ℓ_p spaces. Tartu, 2015, 90 p.
98. **Helle Hallik.** Rational spline histopolation. Tartu, 2015, 100 p.
99. **Johann Langemets.** Geometrical structure in diameter 2 Banach spaces. Tartu, 2015, 132 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
105. **Md Raknuzzaman.** Noncommutative Galois Extension Approach to Ternary Grassmann Algebra and Graded q-Differential Algebra. Tartu, 2016, 110 p.
106. **Alexander Liyvapuu.** Natural vibrations of elastic stepped arches with cracks. Tartu, 2016, 110 p.
107. **Julia Polikarpus.** Elastic plastic analysis and optimization of axisymmetric plates. Tartu, 2016, 114 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.
113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
115. **Tiina Kraav.** Stability of elastic stepped beams with cracks. Tartu, 2017, 126 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.

117. **Silja Veidenberg.** Lifting bounded approximation properties from Banach spaces to their dual spaces. Tartu, 2017, 112 p.
118. **Liivika Tee.** Stochastic Chain-Ladder Methods in Non-Life Insurance. Tartu, 2017, 110 p.
119. **Ülo Reimaa.** Non-unital Morita equivalence in a bicategorical setting. Tartu, 2017, 86 p.
120. **Rauni Lillemets.** Generating Systems of Sets and Sequences. Tartu, 2017, 181 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.
123. **Kaur Lumiste.** Improving accuracy of survey estimators by using auxiliary information in data collection and estimation stages. Tartu, 2018, 112 p.
124. **Paul Tammo.** Closed maximal regular one-sided ideals in topological algebras. Tartu, 2018, 112 p.
125. **Mart Kals.** Computational and statistical methods for DNA sequencing data analysis and applications in the Estonian Biobank cohort. Tartu, 2018, 174 p.
126. **Annika Krutto.** Empirical Cumulant Function Based Parameter Estimation in Stable Distributions. Tartu, 2019, 140 p.
127. **Kristi Läll.** Risk scores and their predictive ability for common complex diseases. Tartu, 2019, 118 p.
128. **Gul Wali Shah.** Spline approximations. Tartu, 2019, 85 p.
129. **Mikk Vikerpuur.** Numerical solution of fractional differential equations. Tartu, 2019, 125 p.
130. **Priit Lätt.** Induced 3-Lie superalgebras and their applications in superspace. Tartu, 2020, 114 p.
131. **Sumaira Rehman.** Fast and quasi-fast solvers for weakly singular Fredholm integral equation of the second kind. Tartu, 2020, 105 p.
132. **Rihhard Nadel.** Big slices of the unit ball in Banach spaces. Tartu, 2020, 109 p.
133. **Katriin Pirk.** Diametral diameter two properties, Daugavet-, and Δ-points in Banach spaces. Tartu, 2020, 106 p.
134. **Zahra Alijani**. Fuzzy integral equations of the second kind. Tartu, 2020, 103 1p.
135. **Hina Arif.** Stability analysis of stepped nanobeams with defects. Tartu, 2021, 165 p.
136. **Joonas Sova.** Pairwise Markov Models. Tartu, 2021, 166 p.
137. **Kristo Väljako.** On the Morita equivalence of idempotent rings and monomorphisms of firm bimodules. Tartu, 2022, 139 p.
138. **Andre Ostrak.** Diameter two properties in spaces of Lipschitz functions. Tartu, 2022, 77 p.

139. **Mohammed Mainul Hossain.** Numerical analysis of vibrations of nanobeams. Tartu, 2022, 158 p.
140. **Alvin Lepik.** On Morita equivalence of semigroups. Tartu, 2023, 107 p.
141. **Shahid Mubasshar.** Natural vibrations of curved nanobeams. Tartu, 2023, 116 p.
142. **Stefano Ciaci.** Transfinite geometric properties of the unit ball in Banach spaces. Tartu, 2024, 117 p.