Tel Aviv University
Raymond and Beverly Sackler Faculty of Exact Sciences
Blavatnik School of Computer Science

Explicit Constructions and Lower Bounds in Coding Theory

A thesis submitted for the degree of
Doctor of Philosophy

By

Roni Con

Thesis supervisors: Prof. Amir Shpilka and Prof. Itzhak Tamo

Submitted to the Senate of Tel Aviv University
September 2023

Contents

Abstract **3**

Acknowledgements **5**

1 Introduction **6**

2 Nonlinear Repair of Reed–Solomon Codes **11**
2.1 Introduction . . . 11
2.1.1 Coding for distributed storage systems . . . 11
2.1.2 The setup . . . 12
2.1.3 Repair of RS codes . . . 12
2.1.4 Linear and nonlinear repair schemes . . . 13
2.1.5 Our contribution . . . 14
2.1.6 Organization of this chapter . . . 14
2.2 A General framework for node repair . . . 15
2.3 Asymptotically MSR RS codes over $\mathbb{F}_p$ . . . 17
2.3.1 Existence of asymptotically $[n, 2, d]$ MSR RS codes over $\mathbb{F}_p$ . . . 18
2.3.2 Outperforming linear repair schemes over field extensions . . . 19
2.4 Explicit constructions of RS codes . . . 20
2.4.1 A toy example . . . 21
2.4.2 An RS code construction with $k < d \leq n/2$ . . . 22
2.4.3 Repairing by any d helper nodes . . . 24
2.5 An improved lower bound on the bandwidth . . . 25
2.6 Concluding remarks and open problems . . . 27
2.7 Appendix . . . 28
2.7.1 Proof of Lemma 2.4.1 . . . 28
2.7.2 Repairing α_i for $i \in \{1, 2, 3\}$ . . . 28

3 Linear codes correcting insertions and deletions **30**
3.1 Introduction . . . 30
3.1.1 Insertion and Deletions . . . 30
3.1.2 Linear codes . . . 31
3.1.3 Basic definitions and notation . . . 31
3.1.4 Previous results . . . 31
3.1.5 Our results . . . 32
3.1.6 Proof idea . . . 32
3.1.7 Organization of the chapter . . . 33
3.2 Linear Insdel Codes over Finite Alphabet via Synchronization Strings . . . 33
3.2.1 Half-linear insdel codes . . . 34
3.2.2 Full linear insdel codes . . . 35
3.3 Binary Linear Codes . . . 37

3.3.1 The inner code . . . 37
3.3.2 Construction of our code . . . 39
3.3.3 Analysis . . . 39
3.3.4 Proof of Theorem 3.1.10 . . . 44
3.4 Open questions . . . 45

4 Reed–Solomon codes correcting insertions and deletions 46
4.1 Introduction . . . 46
4.1.1 Previous results . . . 46
4.1.2 Our results . . . 47
4.1.3 Proof idea . . . 47
4.1.4 Organization . . . 48
4.2 Optimal Reed-Solomon codes exist . . . 48
4.2.1 An algebraic condition . . . 48
4.2.2 Existence using Schwarz-Zippel-DeMillo-Lipton lemma . . . 49
4.2.3 Existence using the Lovász-local-lemma . . . 51
4.3 Deterministic construction for any k . . . 53
4.4 Explicit construction for $k = 2$ with quartic field size . . . 55
4.4.1 A lower bound on the field size . . . 56
4.5 Summary and open questions . . . 57

5 Constructions of coding schemes for the binary deletion channel and the Poisson repeated channel 58
5.1 Introduction . . . 58
5.1.1 Lower bounds on the capacity of the BDC . . . 59
5.1.2 Upper bounds on the capacity of the BDC . . . 59
5.1.3 Efficient constructions for the BDC . . . 59
5.1.4 The Poisson repeat channel . . . 59
5.1.5 Our Results . . . 60
5.1.6 Construction and Proof Overview . . . 60
5.1.7 Organization . . . 63
5.2 Preliminaries . . . 63
5.2.1 Facts from Probability . . . 63
5.2.2 The Code of Haeupler and Shahrasbi [HS17] . . . 65
5.3 The Inner Code . . . 65
5.4 Construction . . . 71
5.5 Correctness and Analysis . . . 72
5.5.1 Correctness of Decoding Algorithm . . . 72
5.5.2 Proof of Theorem 5.1.1 . . . 84
5.6 Rates For Fixed Values of Deletion Probabilities . . . 85
5.7 Poisson Repeat Channel . . . 86
5.7.1 Construction . . . 87
5.7.2 Correctness of Decoding Algorithm . . . 88

Abstract

At the core of coding theory lies a fundamental challenge. We have two parties named Alice and Bob. Alice intends to send messages to Bob, but there is an intermediary that corrupts these messages. The question is: Can Alice still transmit messages to Bob in a way he can understand them? Error-correcting codes, henceforth referred to as 'codes,' represent clever methods for encoding data so that one can recover the original information even if parts of it are corrupted.

Codes are employed not only in communication, but also in disks, storage systems, and in general in physical media that might have errors. Moreover, coding theory's reach extends into theoretical computer science, impacting areas like complexity and cryptography.

Given a storage medium or a communication channel prone to errors, two primary challenges arise in coding theory. Firstly, it is essential to characterize the limits of performance given the error model. In other words, what is the best performance one can hope for? The second challenge involves designing efficient codes capable of robust error recovery, ideally achieving these limits. The results presented in this thesis are of these types where we focus on addressing innovative challenges that are both modern and driven by practical applications.

The first part of this thesis addresses a fundamental issue in the field of coding for distributed storage systems. In modern distributed storage systems, codes are used to safeguard data against server failures. When a server fails, a new replacement server is installed, and then it triggers a data recovery process utilizing the remaining operational servers. The task is to do this process with as little bandwidth (that is, the amount of bits transferred between the functional servers to the new replacement server) as possible. Given that a notable number of distributed storage systems deploy Reed-Solomon (RS) codes, the repair problem for RS codes received a lot of attention recently. By now, there are examples of RS codes that have efficient repair schemes. However, these schemes fall short in several aspects; they require a considerable field extension degree. They do not provide any nontrivial repair scheme over prime fields. Lastly, they are all linear repairs, i.e., the computed functions are linear over the base field. Motivated by these, we study the problem of nonlinear repair for RS codes. Our main results are the first nonlinear repair schemes of RS codes with asymptotically optimal repair bandwidth. Importantly, these are the only known nonlinear repair schemes across all codes, that outperform all linear schemes. Lastly, we derive an improved lower bound for the repair bandwidth of RS codes over prime fields.

In the second part of this thesis, we focus on linear codes that can recover from insertions and deletions. Efficient codes that can handle these errors (called also synchronization errors) hold immense value in DNA-based storage—a novel and captivating storage medium that has recently attracted significant attention. In this part, we construct linear codes over small alphabets that can recover efficiently from adversarial insertions and deletions. Notably, our constructed codes offer better rate-distance tradeoffs than those found in earlier works. We also show that some RS codes reach the optimal rate-distance tradeoff for a linear code, and also provide an explicit construction over a large field. Additionally, for the special case of two-dimensional RS codes, which have been extensively studied in the literature, we provide a construction for an optimal RS code (one with maximal error correction capability) over a field that is exponentially smaller than in previous constructions.

In the third part of this thesis, we study the binary deletion channel, which is the most basic channel that models deletions. This channel models the situation where bits of a transmitted message are deleted (i.e. removed) from the message randomly and independently with probability p. In this part, we construct

efficient codes for this channel with rate $(1-p)/16$.

The first part of this thesis is based on a joint work with Itzhak Tamo [CT22]. The second part is based on joint works with Amir Shpilka and Itzhak Tamo [CST22, CST23]. The third part is based on a joint work with Amir Shpilka [CS22].

Acknowledgements

Chapter 1

Introduction

The purpose of coding theory is to study techniques that enable reliable and efficient communication over channels or mediums that are prone to errors or noise. There are two main error models - a worst-case model and an average-case model. The first model, which is very common in the theory of computation and has found many applications there, is called the Hamming model [Ham50]. This is a worst-case setting in which a transmitted message is subjected to an adversarial corruption of a fraction p of its entries and we must recover the original message regardless of the location of the errors. Thus, if the adversary is allowed to corrupt a fraction p of the entries of a transmitted message, then a code that allows perfect recovery is a subset of the messages such that any two codewords (i.e. elements of the code) have normalized hamming distance larger than $2p$.

The second error model was first considered by Shannon in his pioneering work [Sha48]. This is an average-case model in which a transmitted message is subjected to a random corruption such as bit flips, bit erasures, bit deletions, etc., where each bit is corrupted independently at random according to some distribution. A channel is basically determined by the probability distribution of the corruptions. Since the corruption is random, it can be the case that the whole word is corrupted. In particular, in this setting, the most we can expect from the decoder is to decode the original word with high probability (over the randomness of the corruptions).

To explain the questions that we study we recall some basic notions from coding theory. Let Σ denote an alphabet set. An error-correcting code can be described either as an encoding map $C : \Sigma^k \to \Sigma^n$ or, abusing notation, as the image of such a map C. A code C is said to be an $[n, k]_q$ linear code if $C \subseteq \mathbb{F}_q^n$ is a linear subspace of dimension k. The rate of such a code C is $\mathrm{Rate}(C) = k/n$, which intuitively captures the amount of information encoded in every bit of a codeword. Naturally, we would like the rate to be as large as possible, but there is a tension between the rate of the code and the amount of errors/noise it can tolerate.

The major problems in coding theory are on one hand to determine the largest rate for which one can uniquely recover the original message from a corrupted codeword and on the other hand, to design explicit codes with efficient encoding and decoding algorithms that achieve this rate. This of course depends on the nature of corruption.

Notations

For positive integers $a < b$ let $[a, b] = \{a, a + 1, \ldots, b\}$ and $[a] = \{1, 2, \ldots, a\}$. Throughout this thesis, $\log(x)$ refers to the base-2 logarithm and $h(x)$ refers to the binary entropy function, that is, $h(x) = -x \log(x) - (1 - x) \log(1 - x)$. For a prime power q, we denote with $\mathbb{F}_q$ the field of size q and for a prime number p, $\mathbb{F}_p$ denotes the prime field with p elements.

Scope of this thesis

Part 1 - Nonlinear Repair of Reed–Solomon Codes

In this part, we address a coding-theoretic problem that has its origins in the industry. This problem, referred to as the 'repair problem,' holds significant importance within the domain of distributed storage systems. In such systems, a large file is encoded using an erasure-correcting code and then distributed over many nodes. More specifically, the file is cut into k data fragments that form the input to an erasure-correcting code which generates n data fragments. Then, the system stores them on n distinct storage devices in order to increase data reliability. When a node fails, we would like to be able to set up a replacement node efficiently using information from the remaining nodes. Since an erasure-correcting code is used, repairing the lost data amounts to decoding a single erasure in the codeword. The primary parameter of focus is the *repair bandwidth* which is the total amount of information downloaded from the functional nodes to the replacement node. The problem of recovering the failed node with low repair bandwidth was first considered in the seminal paper of Dimakis et al. [DGW$^+$10] and has since been the topic of much research.

Today, many distributed storage systems utilize Reed-Solomon (RS) codes in practice, such as the specific $[14, 10]$-RS code used in the Facebook storage system [SAP$^+$13]. Therefore, it comes as no surprise that the repair problem in RS codes received attention from the community [SPDC14, GW17b, TYB17, DDKM18, BBD$^+$22]. We recall the definition of RS codes

Definition 1.0.1. *Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct points of the finite field $\mathbb{F}_q$ of order q. For $k < n$ the $[n, k]_q$* RS code *defined by the evaluation set $\{\alpha_1, \ldots, \alpha_n\}$ is the set of codewords*

$$\{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \deg f < k\} \ .$$

When $n = q$, the resulting code is called a full-length RS code*.*

In the distributed storage setting, each evaluation point, $f(\alpha_i), i \in [n]$ is distributed to the ith node. Thus, when we say the ith node, we refer to the code symbol $f(\alpha_i)$. Assuming that the nth node has failed, i.e., the value of $f(\alpha_n)$ is lost, the question boils down to how much information is needed from the remaining nodes storing $f(\alpha_1), \ldots, f(\alpha_{n-1})$, in order to determine $f(\alpha_n)$. It is clear that any k values of $f(\alpha_i)$ suffice to recover the polynomial f, and in particular recover $f(\alpha_n)$. This repair is termed as the "trivial repair" and the amount of bandwidth needed for this repair is $k \cdot \log_2 q$. However, if one contacts d nodes where $d > k$, then it is known that one can repair a failed node with much less bandwidth [SPDC14, GW17b, TYB17]. We note that in distributed storage, d, which is often called the number of *helper nodes*, is the number of nodes that participate in the repair (and not the distance of the codes).

Up until our work, all repair schemes for distributed storage systems had been linear, meaning that the information transmitted from each node to the replacement node was a linear function (over the base field) of the data it stored. As a result, all RS codes used in previous work to address the repair problem had to be defined over extension fields. An open question that appeared in [GW17b] asked "How much better can one do with non-linear repair schemes?"

In Chapter 2, we design nonlinear repair schemes for RS codes over prime fields that asymptotically achieve the optimal bandwidth. Our schemes are the first nonlinear repair schemes in the literature and we show that they outperform all linear repair schemes for RS codes over prime fields.

Our main results are

1. For any $2 < d < n$, we show that $1 - o(1)$ fraction of all the $[n, 2]_p$ RS codes are *asymptotically MSR* codes, where the term $o(1)$ tends to zero as p tends to infinity. Namely, any code symbol admits an asymptotically optimal repair by any set of d helper nodes.

2. For any $k < d \leq n/2$ we present an explicit construction of an $[n, k]_p$ RS code such that its symbols can be partitioned into two sets of equal size, such that each symbol admits an asymptotically optimal repair using any d helper nodes from the other set.

3. Unlike the problem of repairing RS codes over field extensions, we show that one can not achieve the cut-set bound with equality, and over prime fields, one can obtain a tighter lower bound on the total incurred bandwidth. Concretely, we improve the cut-set bound by showing that every node must transmit another additive factor of $\Omega(\log(k)/(d-k+1))$ bits.

This part of the thesis is based on the work done in [CT22].

Part 2 - Linear codes correcting insertions and deletions

In this part, we study how well linear codes perform against adversarial insertions and deletions (insdel for short). An insertion error is when a new symbol is inserted between two symbols of the transmitted word. A deletion is when a symbol is removed from the transmitted word. For example, over the binary alphabet, when 100110 is transmitted, we may receive the word 1101100, which is obtained from two insertions (1 at the beginning and 0 at the end) and one deletion (one of the 0's at the beginning of the transmitted word). We note that in this model of insertions and deletions, the length of the output can be different than the length of the originally transmitted message. Generally, insdel errors cause the sending and receiving parties to become "out of sync" which makes them inherently more difficult to deal with.

Mitzenamacher wrote in his excellent survey [Mit09] that "Channels with synchronization errors, including both insertions and deletions as well as more general timing errors, are simply not adequately understood by current theory. Given the near-complete knowledge we have for channels with erasures and errors... our lack of understanding about channels with synchronization errors is truly remarkable."

Insdel errors appear in diverse settings such as optical recording, semiconductor devices, integrated circuits, and synchronous digital communication networks. Recently, codes for correcting insdel errors have attracted a lot of attention due to their possible application in correcting errors in DNA storage. Unlike classical optical, magnetic, and flash storage technologies, DNA-based storage does not require an electrical supply to maintain data integrity. Furthermore, DNA-based storage system allows very high data densities over classical storage technologies. Given the trends in cost decreases of DNA synthesis and sequencing, it is now acknowledged that DNA storage may become a highly competitive archiving technology [CCS+18]. It is thus natural that designing codes for DNA storage and studying the limitations of this model received a lot of attention recently [HSRD17, LSWZY19, HMG19, BLOS+21, SH+22].

Linear codes over small alphabets. Due to the importance of the insdel model and our lack of understanding of some basic problems concerning it, the model has attracted many researchers in recent years [HS17, BGZ17, GW17a, CJLW18, Hae19, CGHL21, GH21] (see also the recent excellent survey [HS21]). However, even the basic question of whether there exist good *linear* codes, over small alphabets, for the insdel model was unknown until the recent work of Cheng, Guruswami, Haeupler, and Li [CGHL21] where it was shown that there are linear codes over $\mathbb{F}_q$ that can correct δ fraction of insertions and deletions and have rate $(1-\delta)/2 - h(\delta)/\log_2(q)$. On the other hand, [CGHL21] showed that linear codes over $\mathbb{F}_q$ that can decode from a δ fraction of insdel errors cannot have rate larger than $(1-\frac{q}{q-1}\delta)/2 + o(1)$. This bound is called the "half-Plotkin bound" and by removing the dependence on the field size, they formulated the "half-Singleton bound", namely, $R \leq (1-\delta)/2 + o(1)$. They also constructed explicit codes with (extremely small) rates bounded away from zero that can efficiently correct up to a $\delta < 1/400$ fraction of insdel errors.

In Chapter 3 we significantly improve their construction. We construct efficient linear codes that get much closer to the rate-distance tradeoff upper bounds. Specifically, our results are

1. We construct linear codes over $\mathbb{F}_q$, for $q = \text{poly}(1/\varepsilon)$, that can efficiently decode from a δ fraction of insdel errors and have rate $(1-4\delta)/8 - \varepsilon$.

2. We construct binary linear codes that can efficiently correct up to $\delta < 1/54$ fraction of deletions and have rate $R = (1 - 54 \cdot \delta)/1216$.

Reed Solomon codes. Reed-Solomon codes (RS-codes for short) are the most widely used family of codes in theory and practice. Among their applications are QR codes [Soo08], secret sharing schemes [MS81],

space transmission [WB99], encoding data on CDs [WB99] and more. The ubiquity of these codes can be attributed to their simplicity as well as to their efficient encoding and decoding algorithms. As such, it is an important problem to understand whether they can also decode from insdel errors. This question received a lot of attention recently [SNW02, WMSN04, TSN07, DLTX19, LT21, CZ21, LX21], but besides very few constructions (i.e., evaluation points for RS-codes), not much was known before our work.

In Chapter 4, we show that there are RS codes that achieve the best rate-distance tradeoff that a linear code can achieve against insdel errors, namely, the achieve the half-Singleton bound. We also give explicit constructions (i.e. sets of evaluation points) that define RS-codes that achieve this bound. As the field sizes that we get grow very fast, our construction runs in polynomial time only for very small values of k. Additionally, for the special case of two-dimensional RS codes, we provide a construction for an optimal RS code (one with maximal error correction capability) over a field that is exponentially smaller than in previous constructions. Specifically, our results are

1. For $q = O(n^{4k-2})$ there exists an $[n, k]_q$ RS-code defined by n distinct evaluation points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$, that can decode from $n - 2k + 1$ adversarial insdel errors.

2. There is a deterministic construction of an $[n, k]_q$ RS-code that can correct from $n-2k+1$ insdel errors where $q = O\left(n^{k^{2\cdot((2k)!)^2}}\right)$. The construction runs in polynomial time for $k = O(\log(n)/\log\log(n))$.

3. For any $n \geq 4$, there exists an explicit $[n, 2]_q$ RS-code that can correct from $n - 3$ insdel errors, where $q = O(n^4)$.

This part of the thesis relies on the work done in [CST22, CST23].

Part 3 - The binary deletion channel

In this part, we consider the binary deletion channel (BDC) with parameter p. This channel models the situation where bits of a transmitted message are deleted (i.e. removed) from the message randomly and independently with probability p. One of the most fundamental questions when studying a channel is to determine its capacity, i.e., the maximum achievable transmission rate over the channel that still allows recovering from the errors introduced by the channel, with high probability. In spite of many efforts (see the excellent surveys [Mit09, CR20]), the capacity of the BDC_p is still not known and it is an outstanding open challenge to determine it. Yet, in the regime where $p \to 1$, we know that the capacity is at least $(1 - p)/9$ [MD06] and at most $0.4143(1 - p)$ [FD10, Dal11]. The lower bound given by [MD06] is achieved via a probabilistic construction and does not yield an explicit and efficient construction with this rate. Our focus in this part is to construct explicit and efficient codes with high rates for the BDC.

Guruswami and Li [GL18] present a deterministic and efficient code construction for the BDC_p with rate $(1-p)/120$ for all values of p. This rate is smaller than Mitzenmacher's bound, but it is the first construction with rate that scales proportionally to $(1-p)$ for $p \to 1$. In Chapter 5, we improve the construction of [GL18] and provide an efficient construction of codes that have rate $(1 - p)/16$.

We also show that the same techniques give efficient codes for the Poisson repeat channel. In the PRC with parameter λ, each bit of the message is (randomly and independently) replaced with a discrete number of copies of that bit, distributed according to the Poisson distribution with parameter $0 < \lambda$. This channel was first introduced in [MD06] and has tight connections to BDC as explored also in [Che18].

The results in this part are

1. Let $p \in (0, 1)$. There exists a family of binary error correcting codes $\{C_i\}_{i=1}^{\infty}$ for the BDC_p where the block length of C_i goes to infinity as $i \to \infty$ and C_i can be constructed in polynomial time, encoded in linear time, decoded in quadratic time, and has rate $(1 - p)/16$.

2. Let $\lambda \leq 0.5$. There exists a family of binary error correcting codes $\{C_i\}_{i=1}^{\infty}$ for PRC_λ where the block length of C_i goes to infinity as $i \to \infty$ and C_i can be constructed in polynomial time, encoded in linear time, decoded in quadratic time, and has rate at least $\lambda/17$.

After our work was published, the works [Rub22, PLW22] showed how to convert any code to the BDC (not necessarily explicit) into an explicit code that has efficient encoding and decoding algorithms with a negligible decrease in the rate. Therefore, the codes of [MD06] with rate $(1-p)/9$ can be converted to explicit and efficient codes.

This part is based on the work done in [CS22].

Chapter 2

Nonlinear Repair of Reed–Solomon Codes

2.1 Introduction

2.1.1 Coding for distributed storage systems

The purpose of distributed storage systems is to store data reliably over long periods of time using a distributed collection of storage nodes which may be individually unreliable. Ensuring reliability in such systems requires some redundancy. Thus, a natural solution is to use an error-correcting code: a file that needs to be stored in the system is cut into k data fragments that form the input to an erasure correcting code. The code then generates n data fragments out of the k input fragments, and the system stores them on n distinct storage devices in order to increase data reliability.

One of the most prevalent scenarios that needs to be addressed in such a system, is the failure of a single storage node and the efficiency of its repair [RSG+14, Section 6.6] (measured in terms of system resources). When this happens we would like to rapidly repair the data that was lost and store it on a replacement node. Since an erasure-correcting code is used, repairing the lost data amounts to decoding a single erasure in the codeword. This problem, called the *(exact) repair problem* was first considered in the seminal paper of Dimakis et al. [DGW+10] and has since witnessed an explosive amount of research. However, despite considerable progress, there are still many challenges to overcome. To deal with the repair problem, one has to design a *repairing scheme*, i.e., an algorithm that uses the information stored in the remaining functional nodes and recovers the information that was stored on the failed node. A first and naive solution is to use a linear $[n, k]_q$-maximum distance separable (MDS) code which has the property that the original file can be recovered from the content stored on any k out of n nodes. $[n, k]_q$ MDS codes are widely used in storage due to their optimal resiliency for a given amount of redundancy, i.e., they can recover from any $n - k$ erasures by the information stored on the remaining k nodes. In the scenario of the repair problem, if a node fails, we can download k symbol from any k nodes and recover the entire file and in particular, the data that was stored on the failed node. We call this repair, the *trivial repair*.

The main parameter of a repairing scheme that we wish to optimize is the *bandwidth* of the scheme, i.e., the number of bits that are transmitted (during the repair of a failed node) from the functional nodes, in order to recover the data on the failed node. One can easily see that in the trivial repair, the bandwidth is exactly $k \cdot \lceil \log(q) \rceil$ where q is the alphabet size of the code. The main goal is to minimize the repair bandwidth since a large bandwidth can clog the network. Indeed, this was the main subject of many works [DGW+10, ERR10, GERCP13, PDC13, TWB12, WTB16, RSK11], just to name a few.

2.1.2 The setup

Before formally defining and discussing RS codes' repair problem, we begin with basic definitions of linear codes.

An $[n, k]$ *array code* $\mathcal{C}$, with *subpacketization* L over a finite field $\mathbb{F}$ is a linear subspace $\mathcal{C} \subseteq \mathbb{F}^{L \times n}$ over $\mathbb{F}$ and dimension kL. The *length* and the *rate* of the code are n and k/n, respectively, and the elements of $\mathcal{C}$ are called *codewords*. We view each codeword in this code as an $L \times n$ matrix. The ith (code) symbol of a codeword is the ith column of the codeword, which is a vector of length L over $\mathbb{F}$. An $[n, k]$ array code is called MDS if each codeword is uniquely determined by any k of its symbols. Lastly, *scalar codes* (which are the more common mathematical object when one refers to a code) are array codes with $L = 1$. Hence, any scalar code is also an array code. Furthermore, a scalar code $\mathcal{C}$ over a field extension $\mathbb{E}$ with a subfield $\mathbb{F}$, where $[\mathbb{E} : \mathbb{F}] = L$ can be viewed as an array code over $\mathbb{F}$ and subpacketization L, simply by expanding each symbol of the code to a column vector of length L over $\mathbb{F}$, according to some basis of $\mathbb{E}$ over $\mathbb{F}$.

In a distributed storage system that employs an array code of length n, it is assumed that each code symbol resides on a distinct storage device (node) to increase the data reliability in case of a node malfunctioning. As already mentioned, the most common scenario in such systems is a single node's failure, which is a single symbol erasure in the coding theory terminology. Such an event triggers a repair scheme whose goal is to recover the erased symbol by receiving information from a subset of the remaining $n-1$ nodes, called *helper nodes*. We mention that we interchangeably use the term node and a symbol of the code in the sequel and say that node i holds (stores) the ith symbol. The figure of merit considered in this problem is the total incurred bandwidth across the network due to the repair scheme. In other words, how many bits the helper nodes need to transmit to repair the failed node. This quantity is called the *repair bandwidth* whose formal definition is given next.

Definition 2.1.1 (Repair bandwidth)**.** *Let $\mathcal{C}$ be an $[n, k]$ MDS array code. For $i \in [n]$ and a subset $D \subseteq [n] \setminus \{i\}, |D| = d \geq k$ of helper nodes, define $N(i, D)$ as the smallest number of bits that need to be transmitted from the helper nodes D in order to repair the failed node i. The repair bandwidth of the code $\mathcal{C}$ with d helper nodes is defined as*

$$\max_{\substack{i \in [n] \\ D \subseteq [n] \setminus \{i\}, |D| = d}} N(i, D) \ .$$

Note that the transmitted information from each helper node can be any function of the symbol it holds. Next, we shall state the well-known lower bound on the repair bandwidth, called the *cut-set bound*, derived by Dimakis et al. in the seminal work [DGW$^+$10].

Theorem 2.1.2. *[DGW$^+$10] Let $\mathcal{C}$ be an $[n, k]$ MDS array code with subpacketization L over a field $\mathbb{F}$, then for every $i \in [n]$ and every set of d helper nodes $D \subseteq [n] \setminus \{i\}$*

$$N(i, D) \geq \frac{dL \log(|\mathbb{F}|)}{d + 1 - k} \ . \tag{2.1}$$

MDS codes are widely used in practice due to their optimal resiliency to erasures for the given amount of added redundancy. Therefore, MDS codes that also attain the cut-set bound during the repair scheme are highly desirable.

An $[n, k]$ MDS code achieving the cut-set bound (2.1) with equality for every failed node i by every set of d helper nodes is called an $[n, k, d]$ *MSR (minimum storage regenerating)* (array) code.

2.1.3 Repair of RS codes

RS codes are the most known MDS codes, and they have found many applications both in theory and practice (some applications include QR codes [Soo08], Secret sharing schemes [MS81], space transmission [WB99], encoding data on CDs [WB99] and more). The ubiquity of these codes can be attributed to their simplicity and their efficient encoding and decoding algorithms. Thus, it might be surprising that RS codes were not considered a possible solution for the repair problem in distributed storage systems. In fact, many researchers

believed that RS codes do not admit efficient repair schemes except for the trivial scheme. Therefore, several MSR (that are not RS codes) codes were constructed, e.g., [YB17a, YB17b, RSK11, WTB16, GFV17, RSE17]. Yet, the problem of understanding the efficiency of the repair of RS codes remained unresolved, and since many distributed storage systems in fact employ RS codes (e.g., Facebook Hadoop Analytics cluster employs a $[14, 10]$ RS code [SAP+13]), this problem assumed even greater importance.

In [GW17b] it was shown that the repair problem of RS codes could be seen as a new and interesting twist on the standard interpolation problem of polynomials. Thus, studying this problem might have theoretical implications since polynomial interpolation is widely used across all areas of mathematics. Before explaining the new twist on the interpolation problem, we give next a formal definition of RS codes.

Definition 2.1.3. *Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct points of the finite field $\mathbb{F}_q$ of order q. For $k < n$ the $[n, k]_q$* RS code *defined by the evaluation set $\{\alpha_1, \ldots, \alpha_n\}$ is the set of codewords*

$$\{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \deg f < k\} \ .$$

When $n = q$, the resulting code is called a full-length RS code*.*

Thus, a codeword of an $[n, k]$ RS code is the evaluation vector of some polynomial of degree less than k at n distinct points, i.e., the codeword that corresponds to a polynomial f of degree less than k is $(f(\alpha_1), \ldots, f(\alpha_n))$. Assuming that the nth node has failed, i.e., the value of $f(\alpha_n)$ is lost, the question boils down to how much information is needed from the remaining nodes storing $f(\alpha_1), \ldots, f(\alpha_{n-1})$, in order to determine $f(\alpha_n)$. It is clear that any k values of $f(\alpha_i)$ suffice to recover the polynomial f, and in particular recover $f(\alpha_n)$. In the terminology of a repair scheme, this corresponds to $d = k$ helper nodes that transmit their entire symbol. This type of repair scheme is termed as the *trivial repair*, although it also attains the cut-set bound (2.1) with equality.

Hence, the more exciting and challenging question is whether it is possible to recover the polynomial value at a specific location without recovering the original polynomial, thereby possibly requiring less information from the d helper nodes for $d > k$. It turns out that to determine $f(\alpha_n)$, one needs much less information than the amount needed in the trivial scheme that employs polynomial interpolation. Indeed, Shanmugam, Papailiopoulos, Dimakis, and Caire [SPDC14] developed a general framework for repairing scalar MDS codes and, in particular, RS codes, then they exemplified their framework by showing that there are repair schemes for RS codes that are more efficient than the trivial scheme.

Then, Guruswami and Wootters [GW17b] generalized the framework of [SPDC14] and gave a complete characterization of linear repair schemes of scalar MDS codes. They also provided few examples of RS codes with linear repair schemes that outperform the trivial repair scheme. A more recent work by Tamo, Ye, and Barg [TYB17] used the framework of [GW17b] and showed that for every $k < d < n$ there are RS codes that are indeed $[n, k, d]$ MSR codes. The caveat in their work is the large field extension degree (subpacketization) which is $L = \exp((1 + o(1))n \log(n))$.

Unfortunately, in [TYB17], the authors provided an almost matching lower bound of $L = \exp(\Omega(k \log(k)))$ on the degree of the field extension that is required in order for an RS code (and in general any scalar MDS code) to be an MSR code with a linear repair scheme. The results of [GW17b] were extended even further in [DDKM18] and [MBW18], to consider multiple node failures. As a final remark, the strong lower bound on the field size for linear repair schemes, given in [TYB17], gives a clear motivation for studying nonlinear repair schemes.

2.1.4 Linear and nonlinear repair schemes

As already discussed, any code, either scalar or array, can be viewed as an array code over some prime field $\mathbb{F}_p$. Therefore, consider an $[n, k]$ array code $\mathcal{C}$ over $\mathbb{F}_p$ and subpacketization L. We say that a repair scheme is linear if each computed function μ by a helper node is linear over the prime field $\mathbb{F}_p$. Equivalently, if for any $\alpha, \beta \in \mathbb{F}_p$ and $x, y \in \mathbb{F}_p^L$

$$\mu(\alpha x + \beta y) = \alpha\mu(x) + \beta\mu(y).$$

All the existing efficient repair schemes of linear codes rely on the fact that if they are viewed as array codes, their subpacketization level L is extremely large. On the other hand, it is known that this is indeed

needed if one employs linear repair schemes [AG19, TYB17]. Hence we are motivated to study nonlinear repair schemes. Furthermore, Guruswami and Wootters asked the following in [GW17b]: "How much better can one do with nonlinear repair schemes?"

A good starting point for constructing nonlinear repair schemes is to consider array codes with subpacketization $L = 1$. For RS codes, this means to be defined over a prime field. In such a case, any nontrivial repair scheme must be nonlinear. Indeed, any nonzero linear function $\mu : \mathbb{F}_p \to \mathbb{F}_p$ must be bijective, which in terms of the transmitted information means that the helper node sends its entire symbol, and note that this corresponds to the trivial repair scheme. We conclude that any linear repair scheme is equivalent to the trivial repair scheme for RS codes over a prime field. Thus, any improvement over the trivial repair scheme must be nonlinear, and it automatically implies that it outperforms all linear repair schemes.

2.1.5 Our contribution

In this chapter, we make the first step towards understanding nonlinear repair schemes' power for linear codes. In particular, we present a nonlinear repair scheme of RS codes that outperforms all the linear ones. For all we know, this is the first nonlinear repair scheme.

In the real-world scenario of this problem, large files are stored across a relatively small number of nodes. Therefore, each node stores a large chunk of the file. Hence, in this work, we think of k, d, and n as small constants, while the alphabet size p (which corresponds to the data stored by each node) is large and tends to infinity. We note that this point of view is different than the usual point of view in coding theory. In light of that, we say that a code symbol admits an asymptotically optimal repair (bandwidth) if the repair bandwidth tends to the cut-set bound (2.1) as p tends to infinity, and n is fixed.

Below, we summarize the main contributions of this chapter, where d is the number of helper nodes.

1. For any $2 < d < n$, we show that $1 - o(1)$ fraction of all the $[n, 2]_p$ RS codes are asymptotically $[n, 2, d]$ MSR codes, where the term $o(1)$ tends to zero as p tends to infinity. Namely, any code symbol admits an asymptotically optimal repair by any set of d helper nodes.

2. We show that the phenomenon of RS codes with nonlinear repair schemes that outperform all the linear ones also holds over infinitely many field extensions.

3. For any $k < d \leq n/2$ we present an explicit construction of an $[n, k]_p$ RS code such that its symbols can be partitioned into two sets of equal size, such that each symbol admits an asymptotically optimal repair using any d helper nodes from the other set.

4. We show that any full-length RS code over the prime field $\mathbb{F}_p$ exhibits some efficient repair properties. Specifically, for any $k < d$ and large enough p, we show that each node has $\Omega(p)$ distinct sets of helper nodes of size d that can repair it with asymptotically optimal repair bandwidth.

5. Unlike the problem of repairing RS codes over field extensions, we show that one can not achieve the cut-set bound with equality, and over prime fields, one can obtain a tighter lower bound on the total incurred bandwidth. Concretely, we improve the cut-set bound (in the symmetric case, details below) by showing that every node must transmit another additive factor of $\Omega(\log(k)/(d - k + 1))$ bits.

Recall that any linear repair scheme of RS codes over prime fields is equivalent to the trivial repair. Therefore, our nonlinear repair schemes for RS codes over prime fields outperform all the linear ones.

2.1.6 Organization of this chapter

In Section 2.2, we present a general framework for repairing a failed node and obtain a necessary condition for a successful repair. In Section 2.3, we show the existence of $[n, 2]_p$ RS codes, which are asymptotically MSR codes, then extend this result to field extensions. In Section 2.4, we turn to explicit constructions of RS codes with efficient repair schemes. We complement the achievability results (code constructions) given in the previous section by improving the cut-set bound in Section 2.5. In **??**, we discuss the implications of

our results on repairing RS codes for leakage-resilient of Shamir's secret sharing scheme over prime fields. We conclude in Section 2.6 with open questions.

2.2 A General framework for node repair

Throughout, let n, k, and d be the number of nodes (code's length), the RS code dimension, and the number of helper nodes, respectively. Also, let p be a prime number, where we think of n, k, and d as constants, and p tends to infinity.

This section describes a general repair framework for repairing a single failed node that applies to any repair scheme. For simplicity, we will assume that the last node, i.e., the nth node is the failed node that needs to be repaired using all the remaining $n-1$ other nodes, i.e., $d = n-1$. To simplify the notation even further, we assume symmetry between the nodes in terms of the amount of information transmitted, i.e., each helper node transmits the same amount of information. We note that the framework can be easily generalized to the most general case, i.e., an arbitrary failed node, an arbitrary number of helper nodes $k \leq d \leq n-1$, and the non-symmetric case. Lastly, we would like to emphasize that the model assumes no errors; namely, the received information from the helper nodes is error-free. One can generalize this model by removing this assumption, as it was done in [RSRK12, SRV15].

We begin with some needed notations. For positive integers $a < b$ let $[a, b] = \{a, a+1, \ldots, b\}$ and $[a] = \{1, 2, \ldots, a\}$. Throughout, let p be a prime number and for a positive integer m let $\mathbb{F}_{p^m}$ be the finite field of size p^m. An arithmetic progression in some field $\mathbb{F}$ of length N and a step $s \in \mathbb{F}$ is a set of the form $\{a, a+s, \ldots, a+(N-1)s\}$ for some $a \in \mathbb{F}$. For two sets $A, B \subseteq \mathbb{F}_p$, define their *sumset* as $A + B := \{a + b \mid a \in A, b \in B\}$. For an element $\gamma \in \mathbb{F}_p$ we denote by $\gamma \cdot A := \{\gamma \cdot a : a \in A\}$ all the possible products of γ with elements in A.

We are now ready to present the general repair framework. Let $\mathcal{C} \subseteq \mathbb{F}_p^n$ be a linear code over $\mathbb{F}_p$. A repair scheme for its nth symbol is a set of $n-1$ functions $\mu_i : \mathbb{F}_p \to [s]$ and a function $G : [s]^{n-1} \to \mathbb{F}_p$ such that for any codeword $(c_1, \ldots, c_n) \in \mathcal{C}$

$$G(\mu_1(c_1), \ldots, \mu_{n-1}(c_{n-1})) = c_n. \tag{2.2}$$

Upon a failure of the nth symbol, the ith symbol, which holds the symbol c_i computes $\mu_i(c_i)$ and transmits it over the network using $\lceil \log s \rceil$ bits. Upon receiving the $n-1$ messages $\mu_i(c_i)$, the repair scheme is completed by calculating the nth symbol using (2.2). The bandwidth of the repair scheme, which is the total number of bits transmitted across the network during the repair, equals $(n-1)\lceil \log(s) \rceil$ since every node transmits $\lceil \log(s) \rceil$ bits.

One can put any repair scheme under this framework, and the difficulty of the problem stems from finding the functions μ_i that are informative enough, i.e., they provide enough information about c_i, from which collectively it is possible to compute the symbol c_n. However, they should not be too informative, in the sense that the size of the image, s, should be small to minimize the total incurred bandwidth. We have the following simple observation.

Observation 2.2.1. *A set of $n-1$ functions $\mu_i : \mathbb{F}_p \to [s]$ can be extended to a repair scheme, i.e., there exists a function G that satisfies (2.2) if and only if for any two codewords $c, c' \in \mathcal{C}$, such that $\mu_i(c_i) = \mu_i(c'_i)$ for all $i \in [n-1]$ it holds that $c_n = c'_n$.*

Proof. Let μ_i for $i \in [n-1]$ be the $n-1$ functions as above, and define the function G as follows. For a codeword $c \in \mathcal{C}$ define the value of G as in (2.2), i.e., $G(\mu_1(c_1), \ldots, \mu_{n-1}(c_{n-1})) = c_n$. For all other points of $[s]^{n-1}$ define the value of G arbitrarily. By the property the functions μ_i satisfy, it is clear that the function G is well defined, and together they form a valid repair scheme. The other direction is trivial. □

Every function μ_i defines a partition $\{\mu_i^{-1}(a) : a \in [s]\}$ of $\mathbb{F}_p$. Vice versa, any partition of $\mathbb{F}_p$ to s sets defines a function whose value at the point $a \in \mathbb{F}_p$ is the index of the set that contains it. Hence, in the sequel, we will define the functions μ_i by partitions of $\mathbb{F}_p$ to s sets. This chapter's main contribution is identifying the 'right' functions μ_i, equivalently, the 'right' partitions of $\mathbb{F}_p$ that define the μ_i's. As it turns

out, arithmetic progressions are the key for constructing the needed partitions that give rise to an efficient nonlinear repair schemes for codes over prime fields, as explained next.

Fix an integer $1 \leq t \leq p$, set $s = \lceil p/t \rceil$ and define $A_0, \ldots, A_{s-1}$ to be the partition of $\mathbb{F}_p$ into the following s arithmetic progressions of length t and step 1

$$A_j = \begin{cases} \{jt, jt+1, \ldots, jt+t-1\} & 0 \leq j \leq s-2 \\ \{(s-1)t, \ldots, p-1\} & j = s-1. \end{cases} \tag{2.3}$$

For a nonzero $\gamma \in \mathbb{F}_p$, it is easy to verify that $\gamma \cdot A_0, \ldots, \gamma \cdot A_{s-1}$ is also a partition of $\mathbb{F}_p$ into arithmetic progressions of length t and step γ. Each function $\mu_i, i \in [n-1]$ of the repair scheme will be defined by a partition $\gamma_i \cdot A_0, \ldots, \gamma_i \cdot A_{s-1}$ for an appropriate selection of γ_i. Notice that the γ_i's will be distinct for distinct i's and therefore also the functions μ_i will be distinct for distinct i's. Phrasing Observation 2.2.1 in the language of partitions gives the following. The partitions defined by the γ_i's extend to a repair scheme if (and only if) for any two codewords $c, c' \in \mathcal{C}$ that belong to the same set in all of the $n-1$ different partitions, i.e., $c_i, c'_i \in \gamma_i \cdot A_{j_i}$ for all $i \in [n-1]$, it holds that c, c' agree on their nth symbol, i.e., $c_n = c'_n$. In such a case, we say that the γ_i's define a valid repair scheme, and in the following proposition, we provide a relatively simple sufficient but instrumental condition for it.

Proposition 2.2.2. *Let $\mathcal{C} \subseteq \mathbb{F}_p^n$ be a linear code, $t < p$ be an integer, and $\gamma_1, \ldots, \gamma_{n-1}$ be nonzero elements of $\mathbb{F}_p$. If for any $c \in \mathcal{C}$ with $c_i \in \gamma_i \cdot [-t, t]$ for all $1 \leq i \leq n-1$, it holds that $c_n = 0$, then, the γ_i's define a valid repair scheme for the nth node with a total bandwidth of $(n-1)\log(\lceil p/t \rceil)$ bits.*

Before proving the proposition, we remark that the actual number of bits each node sends is $\lceil \log(\lceil p/t \rceil) \rceil$ and thus the total bandwidth is $(n-1)\lceil \log(\lceil p/t \rceil) \rceil \leq (n-1)\log(\lceil p/t \rceil) + n - 1$. Since we think of n as being a small constant compared to p, the additive factor of $(n-1)$ is negligible when $\log(\lceil p/t \rceil) = \Omega(\log(p))$, which is the case throughout the chapter. Thus, for ease of notation we will omit the ceiling in the sequel, and assume that each node transmits $\log(\lceil p/t \rceil)$ bits.

Proof. Let $s = \lceil p/t \rceil$ and define the sets $A_j, 0 \leq j \leq s-1$ as in (2.3). For each $i \in [n-1]$, define the function μ_i according to the partition $\gamma_i \cdot A_0, \ldots, \gamma_i \cdot A_{s-1}$. Namely, $\mu_i(a) = j$ if and only if $a \in \gamma_i \cdot A_j$. Let $c, c' \in \mathcal{C}$ be two codewords that agree on the $n-1$ values $\mu_i(c) = \mu_i(c'), i \in [n-1]$. Then, $c - c'$ is a codeword such that its ith symbol belongs to the set $\gamma_i \cdot [-t, t]$ for $i \in [n-1]$. Therefore, its nth symbol is equal to zero which implies that $c_n = c'_n$, and by Observation 2.2.1 the γ_i's define a valid repair scheme. The claim about the bandwidth follows since each partition consists exactly s sets. □

Remark 2.2.3. *We emphasize here that by using arithmetic progressions we managed to have small sets in Proposition 2.2.2. Indeed, note that in the proof, each coordinate of $c - c'$ belongs to a set of the form $A - A$ and this difference is small in the case that A is an arithmetic progression. Informally speaking, if the partitions were random and each set A in every partition is random, then $|A - A| \approx |A|^2$. For a more formal description about our choice to use arithmetic progression, see the discussion after Theorem 2.5.2 in Section 2.5*

Since our primary focus is RS codes, the following proposition is a specialization of Proposition 2.2.2 to this case. In fact, we also slightly generalize it to address the case of arbitrary node repair and an arbitrary set of helper nodes.

Proposition 2.2.4. *Consider an $[n, k]_p$ RS code defined by the evaluation points $\alpha_1, \ldots, \alpha_n$. Let $\ell \in [n]$ be the failed node and $\mathcal{D} \subset [n] \backslash \{\ell\}$ be a set of d helper nodes for $k \leq d \leq n-1$. Furthermore, let $t < p$ be an integer and $\gamma_i, i \in \mathcal{D}$ be nonzero elements of $\mathbb{F}_p$. If for any polynomial $f(x) \in \mathbb{F}_p[x]$ of degree less than k with $f(\alpha_i) \in \gamma_i \cdot [-t, t]$ for all $i \in \mathcal{D}$, it holds that $f(\alpha_\ell) = 0$, then, the γ_i's define a valid repair scheme for the ℓth node with a total bandwidth of $d \log(\lceil p/t \rceil)$ bits.*

Proof. The result is obtained by applying Proposition 2.2.2 to the punctured $[d+1, k]_p$ RS code defined by the evaluation points $\{\alpha_i \mid i \in \mathcal{D} \cup \{\ell\}\}$. □

All the efficient repair schemes given in the chapter will follow by showing that there is a choice of evaluation points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_p$ such that any node (and not only the last node) can be efficiently repaired by invoking Proposition 2.2.4 with carefully designed partitions (γ_i's).

We say that a polynomial $f \in \mathbb{F}_p[x]$ *passes through* (α, A) for $\alpha \in \mathbb{F}_p$ and a subset $A \subseteq \mathbb{F}_p$ if $f(\alpha) \in A$. Figure 2.1 illustrates a valid repair scheme (defined by the γ_i's) of the nth node of a RS code, that satisfies the condition of Proposition 2.2.4. Namely, any two polynomials $f(x), g(x)$ that pass through the same set in each of the $n-1$ partitions, attain the same value at α_n, i.e., $f(\alpha_n) = g(\alpha_n)$.

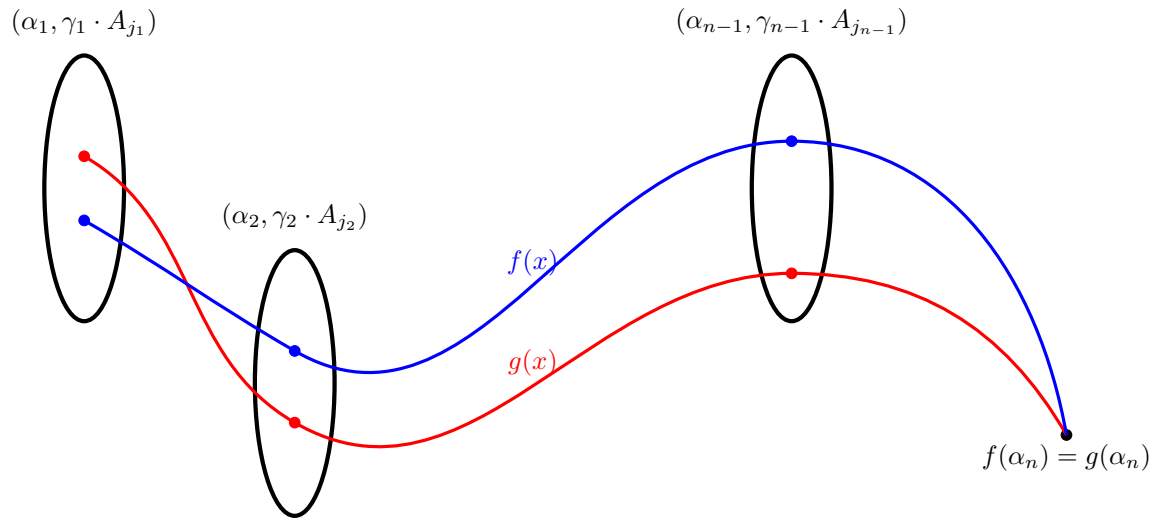

Figure 2.1: A valid repair of the nth node of an $[n, k]$ RS code, that satisfies the condition of Proposition 2.2.4. The two polynomials $f(x)$ and $g(x)$ of degree less than k pass through $(\alpha_i, \gamma_i \cdot A_{j_i})$ for every $i \in [n-1]$ and hence $f(\alpha_n) = g(\alpha_n)$.

2.3 Asymptotically MSR RS codes over $\mathbb{F}_p$

In this section, we show the existence of RS codes over prime fields that have efficient repair schemes, but first we begin with the following two definitions that will be used in the sequel.

Definition 2.3.1. *A code symbol of an $[n, k]$ (scalar) MDS code over a field $\mathbb{F}$ is said to admit an asymptotically optimal repair (bandwidth) if it can be repaired by some set of d helper nodes (for $k \leq d < n$) and bandwidth at most*

$$\log(|\mathbb{F}|) \left(\frac{d}{d-k+1} + o(1) \right) \tag{2.4}$$

bits, where the term $o(1)$ tends to zero as the field size $|\mathbb{F}|$ tends to infinity.

Definition 2.3.2. *An $[n, k]$ (scalar) MDS code is said to be asymptotically $[n, k, d]$ MSR if any of its code symbols admits asymptotically optimal repair, i.e., satisfy* (2.4)*, by* any *set of d helper nodes.*

We proceed to show that over large enough prime fields, there exist $[n, 2]$ RS codes that are asymptotically $[n, 2, d]$ MSR codes for every $2 < d < n$. Then, we proceed to generalize the result to RS over field extensions. We would like to stress that the constructions and those presented in the following sections are asymptotically optimal in the strong sense. Namely, the actual bandwidth differs from the cut-set bound by an additive constant that depends only on the parameters n, k, d and not on the alphabet size. Furthermore, the constructions' repair schemes outperform all the linear repair schemes, which is a phenomenon that was not known to exist before. In particular, for prime fields, the only known repair scheme is the trivial scheme

(which is a linear scheme). Therefore, by outperforming the trivial repair over prime fields, we obtain the first known example of a nontrivial repair over prime fields.

2.3.1 Existence of asymptotically $[n, 2, d]$ MSR RS codes over $\mathbb{F}_p$

In this section we show by a counting (encoding) argument the existence of an asymptotically $[n, 2, d]_p$ MSR RS code for a large enough prime p and any $2 \leq d \leq n-1$. In fact, we show a stronger result, that is, for any $\varepsilon > 0$ and a large enough prime p, a fraction of $1-\varepsilon$ of all the RS codes satisfy this property. In particular, for such a code every code symbol $i \in [n]$ can be repaired by any d helper nodes, where each helper node transmits $(1/(d-1))\log(p) + O_{n,\varepsilon}(1)$ bits, which is roughly a $1/(d-1)$ fraction of the amount of information it holds.

Theorem 2.3.3. *Let $\varepsilon > 0$ and $2 \leq d < n$, then for a large enough prime p, a fraction of $1-\varepsilon$ of all the $[n, 2]_p$ RS codes are asymptotically $[n, 2, d]$ MSR codes with repair bandwidth $\frac{d}{d-1}\log(p) + O_{n,\varepsilon}(1)$.*

Proof. Let $t < p$ be an integer to be determined later, and let $\alpha_1, \ldots, \alpha_n$ be the evaluation set of the $[n, 2]_p$ RS code. We would like to show that any code symbol admits an asymptotically optimal repair by applying Proposition 2.2.4. Assume that the ℓth symbol has failed and that the there exists a set $\mathcal{D} \subseteq [n]\backslash\{\ell\}, |\mathcal{D}| = d$ that does not satisfy the condition in Proposition 2.2.4 with $\gamma_i := \alpha_i - \alpha_\ell$, for $i \in \mathcal{D}$. Therefore, there exists a polynomial $f(x)$ of degree at most one that passes through $(\alpha_i, \gamma_i \cdot [-t, t])$ for $i \in \mathcal{D}$ and $f(\alpha_\ell) \neq 0$. Let $j = \min\{\mathcal{D}\}$ and define the polynomial

$$g(x) := f(x) - \frac{f(\alpha_j)}{\gamma_j}(x - \alpha_\ell).$$

Notice that $g(\alpha_j) = 0$ and $g(\alpha_\ell) = f(\alpha_\ell) \neq 0$, hence $g(x)$ is of the form $g(x) = m(x - \alpha_j)$ for some $m \neq 0$. Also, for $i \in \mathcal{D}\backslash\{j\}$

$$0 \neq g(\alpha_i) = f(\alpha_i) - \frac{f(\alpha_j)}{\gamma_j}\gamma_i \in \gamma_i \cdot [-2t, 2t], \tag{2.5}$$

since $f(\alpha_i), \frac{f(\alpha_j)}{\gamma_j}\gamma_i \in \gamma_i \cdot [-t, t]$. Here, the fact that $\gamma_i \cdot [-t, t]$ is an arithmetic progression plays a crucial rule, since it implies that the size of the set $\gamma_i \cdot [-t, t] - \gamma_i \cdot [-t, t]$ is small. We conclude that one can find a linear polynomial $g(x)$ with the above properties for any such evaluation set. Next, we give an encoding for the (bad) evaluation sets. We note that the term "encoding" is used here for representing our counting argument which counts the number of bad evaluation sets by determining the degrees of freedom in their description.

Encoding:

1. Encode the index of the failed symbol ℓ, and the set $\mathcal{D}$. There are $n\binom{n-1}{d}$ options for this.
2. Encode the evaluation points $\alpha_i, i \notin \mathcal{D} \cup \{\ell\}$. There are p^{n-d-1} options for this.
3. Encode the evaluation points α_j, α_ℓ and α_k, where k is the second smallest element of $\mathcal{D}$. There are at most p^3 options for this.
4. Encode the value $g(\alpha_k)$. By (2.5) there are at most $4t$ options for this.
5. For each $i \in \mathcal{D}\backslash\{j, k\}$ encode the value $g(\alpha_i)/\gamma_i$ which is in the set $[-2t, 2t]\backslash\{0\}$ by (2.5). Again, there at most $(4t)^{d-2}$ for this.

Next, we show that given the encoding, one can recover the original evaluation set. In other words, the encoding mapping is injective.

Decoding: Given Steps (1)-(2) one can recover the index of the failed symbol, the set $\mathcal{D}$ and the evaluation points $\alpha_i, i \notin \mathcal{D} \cup \{\ell\}$. Given Steps (3)-(4) and the fact that $g(x)$ is a nonzero polynomial of

degree at most one that vanishes at α_j, we can find the value of m and hence recover the polynomial $g(x) = m(x - \alpha_j)$. Next, it remains to recover the points α_i, for $i \in \mathcal{D}\backslash\{j, k\}$. Given the value of α_ℓ and $g(\alpha_i)/\gamma_i$ obtained in Steps (4) and (5), respectively, one can construct the *non-degenerate* equation

$$g(\alpha_i) - (\alpha_i - \alpha_\ell) \cdot \frac{g(\alpha_i)}{\gamma_i} = 0,$$

in the variable α_i. Since it is a linear equation, the value of α_i can be uniquely determined.

Hence, the number of such (bad) evaluation sets, i.e., that do not satisfy the condition in Proposition 2.2.4 for repairing any of the n symbols is at most the total number of possible encodings, which is at most $n\binom{n-1}{d}p^{n-d+2}(4t)^{d-1}$ options. Set

$$t = \left\lceil \frac{\varepsilon p^{\frac{d-2}{d-1}}}{10n^{\frac{d+1}{d-1}}} \right\rceil,$$

and then $n\binom{n-1}{d}p^{n-d+2}(4t)^{d-1} < \frac{\varepsilon}{2} \cdot p^n$. On the other hand, the number of possibilities to choose the α_i's is $p^n(1-o(1))$ where the term $o(1)$ tends to zero as p tends to infinity. This implies that at least $p^n(1-\frac{\varepsilon}{2}-o(1))$ of the evaluation sets are not bad. Hence, for large enough p a fraction of at least $1-\varepsilon$ of the possible RS codes satisfy the condition in Proposition 2.2.4. Lastly, for such an RS code, the total incurred bandwidth during the repair of any symbol is

$$d \cdot \log\left(\left\lceil \frac{p}{t} \right\rceil\right) \leq d \cdot \log\left(p^{\frac{1}{d-1}} \cdot 10\varepsilon^{-1} \cdot n^{\frac{d+1}{d-1}}\right) = \frac{d}{d-1}\log(p) + O_{n,\varepsilon}(1),$$

bits[1], as needed. □

2.3.2 Outperforming linear repair schemes over field extensions

In Section 2.3.1 we obtained the first existence result of an RS code that can be asymptotically repaired over a prime field. On the other hand, any linear repair scheme over a prime field is the trivial repair, i.e., repairing with any k helper nodes that transmit their entire symbol, where k is the dimension of the code. Indeed, prime fields have no proper subfields. The only option for a helper node is to transmit its entire symbol. Therefore, the previous section's result can be viewed as the first example wherein a nonlinear repair scheme outperforms all linear ones.

A natural question to consider is whether this phenomenon could be found over field extensions or it is solely a property of prime fields. In this section, we show by relying on the result of Section 2.3.1 and a simple extension of repair schemes over some field to repair schemes over its field extensions, that this phenomenon also occurs over field extensions. More precisely, we exhibit the existence of RS codes over $\mathbb{F}_{p^m}$ for infinitely many primes p and integers m that have a nonlinear repair scheme that outperforms (in terms of the total incurred bandwidth) any linear repair scheme. The outperformance of the nonlinear scheme over linear ones follows from the fact that the latter requires the transmission of $\mathbb{F}_p$-symobls, whereas there is no such constraint on the former.

The following proposition shows that an RS code over a field $\mathbb{F}$ and evaluation points in a subfield of $\mathbb{F}$ can be repaired by invoking any repair scheme of the RS code over the subfield and the same evaluation set. In other words, a repair scheme for an $[n, k]_p$ RS code can be translated to a repair scheme for an $[n, k]_{p^m}$ RS code. We note that a similar result was already given in [LWJ19, Theorem 1], but for the sake of completeness, we state and prove it here again.

Proposition 2.3.4. *Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_p$ be the evaluation set of an $[n, k]_p$ RS code, and suppose that there exists a repair scheme for node i with a set $D \subset [n]$ of d helper nodes and bandwidth of b bits. Then, for every positive integer m the $[n, k]_{p^m}$ RS code with the same evaluation set $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_p$ has a repair scheme for node i with the same set $D \subset [n]$ of d helper nodes and bandwidth of $b \cdot m$ bits.*

[1] As discussed above, the actual bandwidth is $d \cdot \lceil \log\left(\lceil \frac{p}{t} \rceil\right) \rceil$, but the outer ceiling adds at most d bits, which are absorbed in the $O_{n,\varepsilon}(1)$ term, as $d < n$. Hence, it does not affect the final bandwidth.

Proof. Let $\beta_0, \ldots, \beta_{m-1} \in \mathbb{F}_{p^m}$ be a basis of $\mathbb{F}_{p^m}$ over $\mathbb{F}_p$. It can be readily verified that any polynomial $f(x) \in \mathbb{F}_{p^m}[x]$ can be written as

$$f(x) = \sum_{j=0}^{m-1} f_j(x)\beta_j, \text{ where } f_j(x) \in \mathbb{F}_p[x].$$

Then, the problem of repairing the value $f(\alpha_i)$ for a polynomial $f(x) \in \mathbb{F}_{p^m}[x]$ of degree less than k boils down to m independent repairs of $f_j(\alpha_i)$ for $j = 0 \ldots, m-1$ over $\mathbb{F}_p$. By invoking m times the repair scheme for $\alpha_j, j = 0, \ldots, m-1$, we get that the total incurred bandwidth is $b \cdot m$, as needed. □

Proposition 2.3.5. *There are infinitely many primes p and positive integers m, and $2 \leq d < n$ for which there exist an $[n, 2]_{p^m}$ RS code with a nonlinear repair scheme with d helper nodes that outperforms any linear repair scheme.*

Proof. Fix an $\varepsilon > 0$, and consider the $[n, 2]_p$ RS code with evaluation set $\alpha_1, \ldots, \alpha_n$, given in Theorem 2.3.3. Let m be a positive integer not divisible by $d-1$, and consider the $[n, 2]_{p^m}$ RS code with the same evaluation set. By Proposition 2.3.4 there exists a nonlinear repair scheme for this code with bandwidth at most

$$m\left(\frac{d}{d-1}\log(p) + O_{n,\varepsilon}(1)\right).$$

On the other hand, any linear repair scheme over $\mathbb{F}_p$ requires the transmission of $\mathbb{F}_p$-symbols, then by the cut-set bound (2.1) the number of $\mathbb{F}_p$-symbols transmitted is at least $\lceil dm/(d-1) \rceil$. Since $d - 1 \nmid m$ then any linear repair scheme transmits at least

$$\left(\left\lceil \frac{dm}{d-1} \right\rceil - \frac{dm}{d-1}\right)\log(p) - O_{m,n,\varepsilon}(1), \tag{2.6}$$

more bits than the nonlinear repair scheme, as needed. □

Although the improvement of the nonlinear repair scheme over linear schemes is small and might seem very negligible, the difference (2.6) can be arbitrarily large as we increase the alphabet size p. The purpose of this result is to exemplify that nonlinear repair schemes can, in fact, outperform linear schemes even over field extensions and not only over prime fields. We believe that it is possible to exhibit even a more significant gap between the two.

2.4 Explicit constructions of RS codes

In the previous section, we showed the existence of $[n, 2]_p$ RS codes that are asymptotically MSR. In this section, we turn our focus to explicit constructions of RS codes that have efficient repair schemes. We present several such constructions by explicitly presenting the evaluation points α_i and the γ_i's that define the partitions (functions) computed by the helper nodes, as detailed below.

We begin with a toy example of an asymptotically $[4, 2, 3]_p$ MSR RS code. The result will follow by showing that the explicit γ_i's and the evaluation points α_i satisfy the condition of Proposition 2.2.4. Then, we continue to present the main construction of this section. Building on the ideas presented in the toy example, we explicitly construct for all $k < d \leq n/2$ an $[n, k]_p$ RS code such that every node admits an asymptotically optimal repair with many d-sets of nodes as helper nodes, although not all of them. Therefore, the code falls short of being asymptotically MSR. We conclude the section with two more explicit constructions of full-length and folded RS codes that follow directly from this section's main construction.

2.4.1 A toy example

Assume that we would like to construct a $[4,2]_p$ RS code with asymptotically optimal repair for the 4th symbol by invoking Proposition 2.2.4. Then, we need to choose distinct points $\alpha_1,\ldots,\alpha_4 \in \mathbb{F}_p$ and $\gamma_1,\ldots,\gamma_3 \in \mathbb{F}_p$ for which Proposition 2.2.4 holds for $t=\Omega(\sqrt{p})$. Indeed, for such a t, each helper nodes transmits $\frac{1}{2}\log(p)+O(1)$ bits.

Let p be a large enough prime, and let $\gamma_i := \alpha_i - \alpha_4$ for $i=1,2,3$, where

$$\alpha_1 = 0, \alpha_2 = -1, \alpha_3 = \frac{p-1}{2}, \alpha_4 = -(2t+1)\ ,$$

and $t = \lfloor \sqrt{p}/5 \rfloor$. Then, by Proposition 2.2.4 the repair of the 4th symbol is possible if for any polynomial $f \in \mathbb{F}_p[x]$ of degree at most one, that satisfies $f(\alpha_i) \in \gamma_i \cdot [-t,t]$ for $i=1,2,3$, it holds that $f(\alpha_4)=0$. Let f be such a polynomial, i.e., $\deg(f)\le 1$ and $f(\alpha_i)\in\gamma_i\cdot[-t,t]$ for $i=1,2,3$, write $f(\alpha_1)=m\cdot\gamma_1$ for some $m\in[-t,t]$, and consider the polynomial $\hat{f} := m(x-\alpha_4)$. We would like to show that by the above selection of the α_i's, f is equal to $\hat{f}$ and in particular $f(\alpha_4)=\hat{f}(\alpha_4)=0$, as needed. Note that $f(\alpha_i), \hat{f}(\alpha_i) \in \gamma_i\cdot[-t,t]$ for $i=1,2,3$ and $f(\alpha_1)=\hat{f}(\alpha_1)$, then their difference $g := f-\hat{f}$ satisfies, $\deg(g)\le 1$, $g(\alpha_1)=0$. Therefore, $g(x)=s(x-\alpha_1)$ for some $s\in\mathbb{F}_p$, and $g(\alpha_i)\in\gamma_i\cdot[-2t,2t]$ for $i=2,3$. To conclude, we will show that $s=0$ and therefore g is the zero polynomial.

Towards this end, notice first that $s=\frac{g(\alpha_i)}{\alpha_i-\alpha_1} \in \frac{\gamma_i}{\alpha_i-\alpha_1}\cdot[-2t,2t]$, for $i=2,3$, therefore

$$s \in \bigcap_{i=2,3} \frac{\gamma_i}{\alpha_i-\alpha_1}\cdot[-2t,2t]. \tag{2.7}$$

Next, assume that we take a realization of $\mathbb{F}_p$ as all the integers whose absolute value is less than $p/2$, i.e., $F_p=\{0,\pm1,\pm2,\ldots,\pm\frac{p-1}{2}\}$, then

$$\frac{\gamma_2}{\alpha_2-\alpha_1} = -2t \text{ and } \frac{\gamma_3}{\alpha_3-\alpha_1} = \frac{\alpha_3-\alpha_4}{\alpha_3-\alpha_1} = 1-\frac{\alpha_4}{\alpha_3} = 1-2(2t+1) = -4t-1.$$

Therefore, the following products (over $\mathbb{Z}$) satisfy

$$\left|2t\cdot\frac{\gamma_2}{\alpha_2-\alpha_1}\right|, \left|2t\cdot\frac{\gamma_3}{\alpha_3-\alpha_1}\right| < \frac{p}{2}.$$

This implies that there is no wrap around in the calculation of the sets $\gamma_i/(\alpha_i-\alpha_1)\cdot[-2t,2t]$ in (2.7) when viewed as subsets of $\mathbb{F}_p$, and they are equal to the same sets of products when calculated over $\mathbb{Z}$.

By (2.7), s is divisible by $-4t-1$ and by $2t$, and thus, s is divisible by their lcm. Since $-4t-1$ and $-2t$ are coprime, then

$$\operatorname{lcm}(-4t-1,-2t) = 2t(4t+1) > 2t\cdot\left|\frac{\gamma_2}{\alpha_2-\alpha_1}\right| = 4t^2. \tag{2.8}$$

Now assume to the contrary that $s\neq 0$, then by (2.8), s is a nonzero integer whose absolute value is greater than $4t^2$. We get that the absolute value of s is greater than the maximal absolute value of any element in the set $\frac{\gamma_2}{\alpha_2-\alpha_1}\cdot[-2t,2t]$, and we arrive at a contradiction (since (2.7) does not hold). Therefore, the repair of α_4 is possible, where each helper nodes transmits at most

$$\log\left(\left\lceil\frac{p}{t}\right\rceil\right) \le \frac{1}{2}\log(p)+O(1),$$

bits, as needed.

In section 2.7.2 we show that by applying again Proposition 2.2.4, the other evaluation points admit an asymptotically optimal repair with $d=3$. Therefore this is an asymptotically $[4,2,3]$ MSR code.

2.4.2 An RS code construction with $k < d \leq n/2$

This section presents an explicit construction of RS codes over $\mathbb{F}_p$ with efficient repair, as described next. Let d, k, and n be arbitrary positive integers such that $k < d \leq n/2$. We construct an $[n, k]_p$ RS code with the following repair properties. The set of n nodes can be partitioned into equally sized sets, of size $n/2$ (for simplicity, assume that n is even, for odd n the size of the sets differ by one), such that any failed node can be optimally repaired by any subset of d helper nodes from the other set, i.e., the set that does not contain the failed node. Clearly, this constraint provides for each failed node $\binom{n/2}{d}$ possible helper sets; however, the construction falls short in satisfying the definition of an asymptotically MSR code since not any set of d nodes can serve as a set of helper nodes. Another caveat of this construction that we should mention is its low rate, as $k/n \leq 1/2$. It is an interesting open question whether it is possible to modify this construction to a code without these constraints.

Before stating and proving the main result of this section, we state a lemma that provides a lower bound on the least common multiple of several integers.

Lemma 2.4.1. *Let $a_1, \ldots, a_s$ be positive integers, then*

$$\operatorname{lcm}(a_1, \ldots, a_s) \geq \frac{\prod_{i=1}^{s} a_i}{\prod_{1 \leq i < j \leq s} \gcd(a_i, a_j)} .$$

The proof of the Lemma 2.4.1 is given in the Appendix (Section 2.7.1). Next, we present the explicit construction of the RS code with the claimed properties.

Construction 2.4.2. *Let k, d, n be integers such that $k < d \leq n/2$ and n is even. Let $r := \lfloor p^{\frac{1}{d-k+1}} \rfloor$, where p is a large enough prime, and define the $[n, k]_p$ RS with n evaluation points α_i where*

$$\alpha_i = \begin{cases} i & i \in [n/2] \\ r + i - \frac{n}{2} & i \in [n/2 + 1, n]. \end{cases}$$

The following theorem shows that the constructed RS code admits an asymptotically optimal repair for any of its nodes (symbols).

Theorem 2.4.3. *The $[n, k]_p$ RS code given in Construction 2.4.2 admits a partition of its nodes to two sets $[n/2]$ and $[n/2 + 1, n]$ such that any node from one set can be repaired by any set of d helper nodes from the other set, with total bandwidth of $d/(d - k + 1) \log(p) + O_{n,k,d}(1)$ bits.*

Proof. The result will follow by showing that for carefully designed γ_i's, Proposition 2.2.4 holds true. Let $\delta \in [r + 1, r + n/2]$ be the evaluation point of the failed node and note that the proof for the other evaluation points is almost identical and thus is omitted. Let $\alpha_1, \ldots, \alpha_d \in [n/2]$ be a set of d distinct helper nodes, and define the γ_i's as

$$\gamma_i = \begin{cases} (-1)^k (\alpha_i - \delta) \prod_{\substack{j=1 \\ j \neq i}}^{d} (\alpha_j - \alpha_i) & 1 \leq i \leq k - 1 \\ (\alpha_i - \delta) \prod_{j=1}^{k-1} (\alpha_i - \alpha_j) & k \leq i \leq d. \end{cases}$$

Consider a polynomial $f(x)$ of degree less than k such that $f(\alpha_j) := \beta_j \in \gamma_j \cdot T$ for all $1 \leq j \leq d$, where $T = [-t, t]$ and t is a positive integer to be determined later. We will show that in such a case $f(\delta) = 0$ and hence Proposition 2.2.4 holds. Let $h(x)$ be the polynomial of degree less than k defined by the k constraints, $h(\delta) = 0$ and $h(\alpha_i) = f(\alpha_i) = \beta_i$ for all $1 \leq i \leq k - 1$. By the Lagrange interpolation formula one can easily verify that $h(x)$ takes the following form

$$h(x) = \sum_{i=1}^{k-1} \frac{x - \delta}{\alpha_i - \delta} \prod_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{x - \alpha_j}{\alpha_i - \alpha_j} \beta_i.$$

Next, define $g(x) := f(x) - h(x)$ and note that since $g(x)$ is a polynomial of degree less than k that vanishes at α_j for all $1 \leq j \leq k-1$, it takes the following form

$$g(x) = a \prod_{j=1}^{k-1} (x - \alpha_j), \tag{2.9}$$

for some $a \in \mathbb{F}_p$. We wish to show that $a = 0$ which implies that $g \equiv 0$ and $f(\delta) = 0$, as needed. For $m \in [k, d]$

$$g(\alpha_m) = f(\alpha_m) - h(\alpha_m) = \beta_m - \sum_{i=1}^{k-1} \frac{\alpha_m - \delta}{\alpha_i - \delta} \prod_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{\alpha_m - \alpha_j}{\alpha_i - \alpha_j} \beta_i \ .$$

Combined with (2.9) we have

$$a = \frac{\beta_m}{\prod_{j=1}^{k-1}(\alpha_m - \alpha_j)} - \sum_{i=1}^{k-1} \frac{\alpha_m - \delta}{\alpha_i - \delta} \frac{1}{\alpha_m - \alpha_i} \prod_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{1}{\alpha_i - \alpha_j} \beta_i \ .$$

Since $\beta_j \in \gamma_j \cdot T$ for $j = 1, \ldots, d$ then

$$a \in (\alpha_m - \delta) \cdot T - (\alpha_m - \delta) \sum_{i=1}^{k-1} \prod_{\substack{j=k \\ j \neq m}}^{d} (\alpha_j - \alpha_i) \cdot T \ ,$$

i.e., a belongs to a set which is a sum of k sets. Equivalently we can write

$$a \in (\alpha_m - \delta) \cdot \left(T - \sum_{i=1}^{k-1} \prod_{\substack{j=k \\ j \neq m}}^{d} (\alpha_j - \alpha_i) \cdot T \right) \ . \tag{2.10}$$

Consider a realization of $\mathbb{F}_p$ as all the integers whose absolute value is less than $p/2$, i.e., $F_p = \{0, \pm 1, \pm 2, \ldots, \pm \frac{p-1}{2}\}$, and consider the set in (2.10) as a subset of $\mathbb{Z}$, where all the multiplications and additions are done over $\mathbb{Z}$. We claim that the absolute value of any of its elements is at most $|\alpha_m - \delta| tk(n/2)^{d-k}$. Indeed, $\alpha_j \in [n/2]$ for each $j = 1, \ldots, d$ and therefore $|a_j - a_i| \leq n/2$. We conclude that the set in (2.10) is contained in the set

$$(a_m - \delta) \cdot [-Ct, Ct],$$

where $C = C_{n,k,d}$ is a positive constant that depends only on n, k and d. Let $t = \xi p^{1 - \frac{1}{d-k+1}}$, where ξ is a small positive constant which will be determined later. Next, if ξ is small enough, then for any $m = k, \ldots, d$ the absolute value of an element of the set $(a_m - \delta) \cdot [-Ct, Ct]$ is less than $p/2$, and therefore there is no wrap around when it is viewed as an element of $\mathbb{F}_p$. To conclude, by (2.10) the integer a is divisible by $\alpha_m - \delta$ for $m = k, \ldots, d$, hence it is divisible by their least common multiple. By Lemma 2.4.1, and the fact that for distinct $\alpha_i, \alpha_j \in [n/2]$, it holds that $\gcd(\alpha_i - \delta, \alpha_j - \delta) = \gcd(\alpha_i - \delta, \alpha_j - \alpha_i) \leq n/2$, then

$$\operatorname{lcm}(\alpha_k - \delta, \ldots, \alpha_d - \delta) \geq \frac{|(\alpha_k - \delta) \cdots (\alpha_d - \delta)|}{\left(\frac{n}{2}\right)^{\binom{d-k+1}{2}}} = \Omega(\delta^{d-k+1}) = \Omega(r^{d-k+1}) = \Omega(p),$$

since k, n and d are constants with respect to p. Thus, there exists a positive constant $\varepsilon = \varepsilon_{n,k,d} < 1$ such that $|a| > \varepsilon p$ or $a = 0$. On the other hand, recall that $a \in (\alpha_m - \delta) \cdot [-Ct, Ct]$ and thus, for small enough ξ we get that $|a| < \varepsilon p$. Thus, it must be that $a = 0$, which implies that $g \equiv 0$ and in particular $g(\delta) = f(\delta) = 0$. Therefore, by Proposition 2.2.4, it is possible to repair the failed symbol node δ, as needed. The total incurred bandwidth by the scheme is $d \cdot \log(p/t) = d \log(p^{1/(d-k+1)}) + d \log(1/\xi)$ and the results follows immediately since ξ depends only on n, k, and d. $\square$

Next, we shall make use of Construction 2.4.2 to construct a full-length RS code over $\mathbb{F}_p$ that has good repair properties. Namely, we show that every code symbol has at least $\Omega(p)$ distinct helper sets that enable an asymptotically optimal repair.

Theorem 2.4.4. *Let $d > k$ be positive integers. Let p be a large enough prime and consider the full length $[p,k]_p$ RS code, Then, every failed code symbol has $\Omega(p)$ distinct sets of helper nodes of size d that can repair it with bandwidth at most $d/(d-k+1)\log(p) + O_{k,d}(1)$.*

Proof. Let $G = \{ax + b : a, b \in \mathbb{F}_p, a \neq 0\}$ be the affine general linear group acting on $\mathbb{F}_p$. It is well-known that G is sharply 2-transitive, and therefore the subgroup $G_\delta = \{ax + b \in G : a\delta + b = \delta\}$ that stabilizes a point $\delta \in \mathbb{F}_p$ is sharply transitive on $\mathbb{F}_p \backslash \{\delta\}$.

Let $\delta \in \mathbb{F}_p$ be the failed node. Let $h \in G$ be an affine transformation such that $h(1) = \delta$ and set $A := h([r+1, r+d]) = \{h(a) : a \in [r+1, r+d]\}$, where $r := \lfloor p^{\frac{1}{d-k+1}} \rfloor$. We claim that $G^A_\delta = \{g(A) : g \in G_\delta\}$, the orbit of the set A under the action of G_δ is of size $\Omega(p)$, and each set in the orbit forms a set of helper nodes for repairing the failed node δ. Indeed, it is well-known that the size of the orbit satisfies

$$|G^A_\delta| = \frac{|G_\delta|}{|G_{\delta,A}|} = \frac{p-1}{|G_{\delta,A}|},$$

where $G_{\delta,A} = \{g \in G_\delta : g(A) = A\}$ and the second equality follows from the fact that G_δ is sharply transitive on $\mathbb{F}_p \backslash \{\delta\}$. Again by the sharp transitivity of G_δ there exists exactly one affine transformation $g \in G_{\delta,A}$ such that $g(a_1) = a_2$ for any two $a_1, a_2 \in A$, therefore, $|G_{\delta,A}| \leq |A| = d$ and $|G^A_\delta| \geq (p-1)/d = \Omega(p)$.

Next, let be $B \in G^A_\delta$, where $g(A) = B$, then the affine transformation $g \circ h$ satisfies $g \circ h(1) = \delta$ and $g \circ h([r+1, r+d]) = B$. Next, consider the punctured code of the full-length RS code, defined by the $d+1$ evaluation points δ and B. Since RS codes are invariant under linear transformation, the code can be viewed as a RS code with evaluation set 1 and $[r+1, r+d]$ due to the affine transformation $g \circ h$. Hence, repairing the failed node δ with helper nodes B is equivalent to repairing the failed node 1 with helper nodes $[r+1, r+d]$. By Theorem 2.4.3 this can be done with bandwidth at most $d/(d-k+1)\log(p) + O_{k,d}(1)$ bits. Note that we invoked Theorem 2.4.3 with the punctured code $[2d, k]_p$ RS code, and thus the term $O_{n,k,d}(1)$ is in this case $O_{k,d}(1)$. Hence, the set $B \in G^A_\delta$ is indeed a valid set of helper nodes, and the result follows. □

2.4.3 Repairing by any d helper nodes

In the quest of constructing an RS code over a prime field that is asymptotically MSR, we give a construction of a folded RS code that follows from Construction 2.4.2. This new construction falls short in achieving this goal in two aspects compared with Construction 2.4.2. First, the bandwidth is not asymptotically optimal, and second, the alphabet is not of prime order, albeit very close to it. However, it allows us to repair each failed node with any $d \geq k$ helper nodes and very small bandwidth. The construction is derived from Construction 2.4.2 by a simple folding operation, and it was observed by Amir Shpilka (who kindly allowed us to include it here).

Corollary 2.4.5. *Let p be a large enough prime. Let k, d, and n be positive integers such that $2k < d < n$, then there exists an $[n,k]_{p^2}$ folded-RS code such that any node can be repaired using* any *d helper nodes with bandwidth of at most $(2d/(d-2k+1))\log(p) + O_n(1)$ bits.*

Proof. Consider the $[2n, 2k]_p$ RS code provided in Construction 2.4.2, and recall that it is defined by the evaluation points $[1, n] \cup [r+1, r+n]$. Next, define the $[n,k]_{p^2}$ folded-RS as follows. For a polynomial $f \in \mathbb{F}_p[x]$ of degree less than $2k$ define a codeword with ith super-symbol to be $(f(i), f(i+r))$ for $i = 1, \ldots, n$.

If the ith symbol fails, then any set of d nodes can repair it. Indeed, fix a set of helper nodes $D \subseteq [n], |D| = d$ then the values $f(i)$ and $f(r+i)$ can be repaired by the values $f(j), j \in D$ and $f(j+r), j \in D$, respectively. The claim on the total bandwidth follows from the bandwidth guaranteed in Theorem 2.4.3, and the result follows. □

Note that by the cut-set bound, the lower bound on the repair bandwidth for codes with these parameters is at least $(2d/(d-k+1))\log(p)$ bits. Hence, the guaranteed repair bandwidth given in Corollary 2.4.5 is not asymptotically optimal, yet, for large values of d compared with k, it is very close to it.

The folded RS construction presented above can be viewed as an array code with subpacketization $L = 2$ over the field $\mathbb{F}_p$. We are aware of only one more code construction with such a small subpacketization level over its prime field $\mathbb{F}_p$ and with an efficient repair scheme. We state this result next, rephrased to fit this context.

Theorem 2.4.6. *[GW17b, Theorem 10] Let p be a prime. For any $n \leq 2(p-1)$, there is an $[n,k]_{p^2}$ RS code and linear repair scheme such that any failed node can be repaired using the $n-1$ remaining nodes with bandwidth*

$$\left(\frac{3n}{2} - 2\right)\log(p).$$

By setting $d = n-1$ in Corollary 2.4.5 we obtain a bandwidth of

$$\frac{2(n-1)}{n-2k}\log(p) + O_n(1)\ ,$$

which is significantly smaller in the regime where $k \leq C \cdot n/2$ for any constant $C < 1/2$ and even in the regime where $n - 2k = \omega(1)$. We note that it is still smaller (though not significantly) when $n-2k$ is a small constant integer. The major downside is that Corollary 2.4.5 requires that $k < n/2$ while Theorem 2.4.6 holds for any $k \leq n-2$.

2.5 An improved lower bound on the bandwidth

All of the constructions presented in this chapter do not achieve the cut-set bound (2.1), and the incurred bandwidth is larger than it by an additive factor that depends on k or n. Hence, we ask if this is indeed necessary, i.e., whether the cut-set bound is not tight for RS codes over prime fields, and if not, can it be improved?

We answer this question in the affirmative by showing that the bandwidth is at least $d\log(p)/(d-k+1) + \Omega(d\log(k)/(d-k+1))$ which is an improvement over the cut-set bound by an additive factor of $\Omega(d\log(k)/(d-k+1))$.

Before stating and proving the main result, we shall recall the following well-known theorem from additive combinatorics used in the proof.

Theorem 2.5.1 (Cauchy-Davenport inequality)**.** *[Cau12, Dav35] If p is a prime and $A, B \subseteq \mathbb{Z}_p$ then, $|A+B| \geq \min(|A|+|B|-1, p)$.*

Theorem 2.5.2. *Consider a k-dimensional RS code over a prime field $\mathbb{F}_p$, and assume that there is a repair scheme for node $\alpha \in \mathbb{F}_p$ by the d helper nodes $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_p$ where each helper node transmits the same amount of information. Then, the bandwidth for each helper node is at least*

$$\frac{\log(kp) - 1}{d-k+1}$$

bits.

Proof. For $i = 1, \ldots, d$, let $\mu_i : \mathbb{F}_p \to [s]$ be the function calculated by the helper node (symbol) α_i, and let $\mathcal{A}_i$ be the partition of $\mathbb{F}_p$ defined by μ_i, i.e.,

$$\mathcal{A}_i = \{\mu_i^{-1}(m) : m \in [s]\}.$$

It is clear that since the μ_i's define a repair scheme, then for any choice of sets $A_1 \in \mathcal{A}_1, \ldots, A_d \in \mathcal{A}_d$, all the polynomials that pass through $(\alpha_1, A_1), \ldots, (\alpha_d, A_d)$ attain the same value at α. Note that it might be possible that there are no such polynomials.

Fix for $i = 1, \ldots, k$, $A_i \in \mathcal{A}_i$ of size at least p/s, and denote by $\mathcal{F}$ the set of polynomials of degree less than k, that pass through $(\alpha_1, A_1), \ldots, (\alpha_k, A_k)$. Since the code dimension is k, the size of $\mathcal{F}$ is exactly $\prod_{i=1}^{k} |A_i|$. Next, set $U = \{f(\alpha) : f \in \mathcal{F}\}$ to be the set of values attained by the polynomials in $\mathcal{F}$ at α. We have the following claim on the size of U.

Claim 2.5.3. $|U| \geq kp/s - k$.

Proof. Let $f = \sum_{i=0}^{k-1} f_i(x-\alpha)^i$ be a polynomial in $\mathcal{F}$, and note that $f(\alpha) = f_0$. By abuse of notation let also $f = (f_0, \ldots, f_{k-1})$ to be the vector of coefficients of f. Let V be the Vandermonde matrix defined by the k distinct elements $\alpha_1 - \alpha, \ldots, \alpha_k - \alpha$, i.e., $V_{ij} = (\alpha_j - \alpha)^{i-1}$, where $i, j = 1, \ldots, k$, and let $A_1 \times \ldots \times A_k = \{(a_1, \ldots, a_k) : a_i \in A_i)\}$. Thus, $f \in \{wV^{-1} : w \in A_1 \times \ldots \times A_k\}$ for any $f \in \mathcal{F}$, and in particular

$$U = \sum_{i=1}^{k} A_i (V^{-1})_{i1} \ . \tag{2.11}$$

We claim that the first column of V^{-1} has nonzero entries. Indeed, the ith row of the inverse Vandermonde matrix corresponds to the coefficients of a polynomial $p(x)$ that vanishes at $\alpha_j - \alpha$ for $j \in [k] \setminus \{i\}$ and $p(\alpha_i - \alpha) = 1$. Hence, the first entry of the row equals $p(0)$ which is clearly nonzero since $p(x)$ is a nonzero polynomial of degree less than k with $k-1$ nonzero roots.

Now, by applying k times the Cauchy-Davenport inequality we get that $|U| \geq \sum_{j=1}^{k} |A_j| - k \geq kp/s - k$. □

We conclude that there are at least $kp/s - k$ polynomials g_i in $\mathcal{F}, i = 1, \ldots, kp/s - k$ that attain distinct values at α. Define a mapping $\mathcal{F} \to \mathcal{A}_{k+1} \times \ldots \times \mathcal{A}_d$ by $f \mapsto (A'_{k+1}, \ldots, A'_d)$, where A'_i is the set in the partition $\mathcal{A}_i$ that contains $f(\alpha_i)$. Clearly, this map has to be injective on the polynomials g_i, otherwise the repair scheme would not be able to repair the failed node. Hence

$$s^{d-k} \geq k \cdot \frac{p}{s} - k \geq \frac{kp}{2s}.$$

By rearranging, we get that

$$\log(s) \geq \frac{\log(kp) - 1}{d - k + 1} \ .$$

□

The following theorem is an inverse theorem to the Cauchy-Davenport inequality, which characterizes the sets that attain the bound with equality.

Theorem 2.5.4. *[Vos56, Vosper's theorem] Let $A, B \subseteq \mathbb{Z}_p$ where $|A|, |B| \geq 2$ and $|A + B| \leq p - 2$. Then $|A + B| = |A| + |B| - 1$ if and only if A and B are arithmetic progressions with the same step.*

By examining the improved bound's proof, it is clear that a large set U in the proof of Theorem 2.5.2 would imply a strong lower bound on the bandwidth. Considering the extreme case where we replace the size of U with the trivial lower bound of p/s, we recover the cut-set bound. Therefore, the improvement follows from sumsets' expansion phenomenon over prime fields, exemplified by the Cauchy-Davenport inequality. On the other hand, it is known that the cut-set bound is tight over large field extensions, which implies that the trivial lower bound of p/s is, in fact, tight. And indeed, field extensions do contain nontrivial subsets whose sumset do not exhibit any expansion, i.e., subsets A, B such that $|A + B| = |A|$. Hence, we arrive at the following conclusion. If one wants to construct efficient repair schemes for RS codes over prime fields, then the size of the set U should have little expansion as possible. Identifying the structure of such sets (a.k.a. sets with small doubling constant) is a well-studied problem in additive combinatorics, known as the *inverse sum set problem*. Over prime fields, Vosper's theorem (Theorem 2.5.4) shows that the only sets that attain the Cauchy-Davenport lower bound with equality are arithmetic progressions with the same step. This fact lies at the core of all the constructions given in this chapter, as arithmetic progressions define all

the functions computed by the helper nodes with the same step. For example, consider the $[n, 2]$ RS code given in Section 2.3.1, and assume that we would like to repair the nth symbol. Then, a simple computation shows that by the partitions defined for the repair scheme, the set U takes the following form

$$U = \frac{(\alpha_1 - \alpha_n)(\alpha_2 - \alpha_n)}{\alpha_1 - \alpha_2}(A - B)$$

where A and B are both arithmetic progressions of size t and step 1, and therefore $|U| = |A| + |B| - 1$, which is as small as possible.

As a final remark, one might ask whether a random repair scheme is expected to have low bandwidth. By a random scheme, we mean that the function computed by each helper node is randomly picked among all functions with a fixed range size, which is equivalent to picking a random partition of the field to a fixed number of sets. One can verify that in such a case, with high probability, the size of U is as large as possible since the sumset of two random subsets is large with high probability. Thus, the event of picking an efficient repair scheme is improbable, and one needs to carefully construct the repair scheme to obtain low bandwidth.

2.6 Concluding remarks and open problems

The study presented in this chapter was inspired by the interesting open question raised in [GW17b] regarding whether nonlinear repair schemes exist, and if so, can they outperform linear schemes. Since for codes over prime fields, any linear repair scheme is the trivial scheme, any efficient repair scheme, i.e., a repair scheme that outperforms the trivial one, must be nonlinear. Hence, our primary focus was on constructing repair schemes of RS codes over prime fields.

We were able to exhibit the first nonlinear repair scheme of RS codes over prime fields, which is also the only known example of a nonlinear repair scheme of any code. As a byproduct, we showed that nonlinear ones can outperform linear repair schemes. Furthermore, some of the repair schemes are asymptotically optimal, as the alphabet size tends to infinity. Lastly, we also improved the cut-set bound for RS codes over prime fields and discussed connections to leakage-resilient Shamir's Secret Sharing over prime fields.

We end this discussion by mentioning several open questions that, in our opinion, could further improve the study of nonlinear repair schemes.

1. Is it possible to apply our approach of using arithmetic progressions to obtain RS codes over prime fields that are asymptotically MSR codes for any positive integers $k < d < n$? Note that it is unknown if such codes exist, although they likely do. Moreover, is it possible to generalize the approach to other codes over prime fields?

2. The work of [BDIR18] showed that for some parameters, an adversary learns almost nothing on the secret in SSS, whereas we showed in this chapter the other extreme case. Namely, for some other parameters, the adversary learns the entire secret. Hence, it is interesting to fill in the gaps and improve our understanding of SSS performance under the remaining parameters regime. In particular, better understand the dependence between the field size p and the number of bits m leaked to the adversary to learn something or the whole secret. For example, what can be said for $m = O(\log(p))$, $k = O(n)$, and $p > 2n$. Notice that any new result would automatically have implications on the other model of repairing RS codes.

3. It is known [TYB17, AG19] that linear MSR code with a linear repair exists only over alphabet size, which is at least doubly exponential in the code dimension. Can this result be generalized to nonlinear repair schemes? In particular, does the field size have to be large for efficient repair schemes to exist over prime fields?

2.7 Appendix

2.7.1 Proof of Lemma 2.4.1

Proof. We prove it by induction on s. If $s = 2$ it holds that

$$\operatorname{lcm}(a_1, a_2) = \frac{a_1 \cdot a_2}{\gcd(a_1, a_2)} .$$

Assume that the claim holds for $s - 1$. It holds that

$$\begin{aligned}
\operatorname{lcm}(a_1, \ldots, a_s) &= \operatorname{lcm}(a_1, \operatorname{lcm}(a_2, \ldots, a_s)) \\
&= \frac{a_1 \cdot \operatorname{lcm}(a_2, \ldots, a_s)}{\gcd(\ a_1, \operatorname{lcm}(a_2, \ldots, a_s))} \\
&\geq \frac{a_1 \cdots a_s}{\gcd(a_1, a_2 \cdots a_s) \cdot \prod_{2 \leq i < j \leq s} \gcd(a_i, a_j)} \qquad (2.12) \\
&\geq \frac{a_1 \cdots a_s}{\prod_{i=2}^{s} \gcd(a_1, a_i) \prod_{2 \leq i < j \leq s} \gcd(a_i, a_j)} \qquad (2.13) \\
&= \frac{a_1 \cdots a_s}{\prod_{1 \leq i < j \leq s} \gcd(a_i, a_j)} .
\end{aligned}$$

Inequality (2.12) follows from the induction hypothesis and from the fact that $\operatorname{lcm}(a_2, \ldots, a_s) \mid a_2 \cdots a_s$ which implies that

$$\gcd(a_1, \operatorname{lcm}(a_2, \ldots, a_s)) \leq \gcd(a_1, a_2 \cdots a_s) .$$

Inequality (2.13) follow from Claim 2.7.1 which is stated and proved below. □

Claim 2.7.1. *Let $b, a_1, \ldots, a_s$ be integers. It holds that*

$$\gcd(b, a_1 \cdots a_s) \leq \gcd(b, a_1) \cdot \gcd(b, a_2) \cdots \gcd(b, a_s)$$

Proof. It is easy to check that $\gcd(b, a_1 a_2)$ divides the product $\gcd(b, a_1) \cdot \gcd(b, a_2)$. Thus, by induction on s, we get that $\gcd(b, a_1 \cdots a_s) | \gcd(b, a_1 \cdots a_{s-1}) \gcd(b, a_s) | \gcd(b, a_1) \cdots \gcd(b, a_s)$. □

2.7.2 Repairing α_i for $i \in \{1, 2, 3\}$

Recall the four evaluation points

$$\alpha_1 = 0, \alpha_2 = -1, \alpha_3 = \frac{p-1}{2}, \alpha_4 = -(2t+1) ,$$

where we assume that $t = \lfloor \sqrt{p}/5 \rfloor$. In the following, we show that Proposition 2.2.4 holds also for all the remaining symbols.

Assume that we wish to repair the ith node for some $i \in \{1, 2, 3\}$ and define $\gamma_j = \alpha_j - \alpha_i$ for every $j \in [4] \setminus \{i\}$. By Proposition 2.2.4, the repair of α_i succeeds if for any polynomial f of degree at most one, that satisfies $f(\alpha_j) \in \gamma_j \cdot [-t, t]$ for $j \in [4] \setminus \{i\}$, it holds that $f(\alpha_i) = 0$. Let f be such a polynomial, i.e., $\deg(f) \leq 1$ and $f(\alpha_j) \in \gamma_j \cdot [-t, t]$ for $j = [4] \setminus \{i\}$, write $f(\alpha_4) = m \cdot \gamma_4$ for some $m \in [-t, t]$, and consider the polynomial $\hat{f} := m(x - \alpha_i)$. We will show that $f(\alpha_i) = \hat{f}(\alpha_i) = 0$, as needed.

Define $g := f - \hat{f}$ and denote $\{k, \ell\} = [3] \setminus \{i\}$. It holds that

$$\begin{aligned}
g(\alpha_k) &\in \gamma_k \cdot [-2t, 2t] \\
g(\alpha_\ell) &\in \gamma_\ell \cdot [-2t, 2t] \\
g(\alpha_4) &= 0 \\
g(\alpha_i) &= f(\alpha_i) .
\end{aligned}$$

Thus, $g(x) = s(x - \alpha_4)$ and calculating, we get

$$s \in \frac{\alpha_k - \alpha_i}{\alpha_k - \alpha_4}[-2t, 2t]$$
$$s \in \frac{\alpha_\ell - \alpha_i}{\alpha_\ell - \alpha_4}[-2t, 2t] \ .$$

We will show that if a certain condition is satisfied, then there are no $a, b \in [-2t, 2t] \setminus \{0\}$ such that

$$(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1} \cdot a = (\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1} \cdot b \tag{2.14}$$

which implies that $s = 0$, as needed.

Assume that we take a realization of $\mathbb{F}_p$ as all the integers whose absolute value are less than $p/2$, i.e., $F_p = \{0, \pm 1, \pm 2, \cdots, \pm\frac{p-1}{2}\}$. Assume that the absolute value of the products $(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1} \cdot 2t$ and $(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1} \cdot 2t$ are less than $p/2$. Hence, in such a case the calculation in equation (2.14) holds over $\mathbb{Z}$.

Lastly, if the lcm of $|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}|$ and $|(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}|$ is greater than $2t \cdot \min(|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}|, |(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}|)$ (again we view them as integers), then it is easy to verify that there are no a and b in $[-2t, 2t] \setminus \{0\}$ such that equation (2.14) holds. Therefore, it must be that $a = b = 0$ which implies that $s = 0$, and we are done.

Now, note that there is symmetry between k and ℓ, thus we check only the following three options:

- $k = 1, \ell = 2, i = 3$. In this case, we get that $|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}| = 4t + 2$, $|(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}| = 4t$, and $\mathrm{lcm}(4t + 2, 4t) = 4t \cdot (2t + 1)$.
- $i = 1, \ell = 2, k = 3$. In this case, we get that $|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}| = 4t - 1$, $|(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}| = 2t$, and $\mathrm{lcm}(4t - 1, 2t) = 2t \cdot (4t + 1)$.
- $k = 1, i = 2, \ell = 3$. In this case, we get that $|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}| = 2t + 1$, $|(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}| = 4t + 1$, and $\mathrm{lcm}(4t + 1, 2t + 1) = (4t + 1) \cdot (2t + 1)$.

In all cases, one can verify that $|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1} \cdot 2t|, |(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1} \cdot 2t| < p/2$. Furthermore, in all cases, it holds that the lcm of $|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}|$ and $|(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}|$ is strictly greater than $2t \cdot \min(|(\alpha_k - \alpha_4)(\alpha_k - \alpha_i)^{-1}|, |(\alpha_\ell - \alpha_4)(\alpha_\ell - \alpha_i)^{-1}|)$. We conclude that α_i can be repaired with the desired bandwidth.

Chapter 3

Linear codes correcting insertions and deletions

3.1 Introduction

In this chapter, We construct linear codes over $\mathbb{F}_q$, for $q = \text{poly}(1/\varepsilon)$, that can efficiently decode from a δ fraction of insdel errors and have rate $(1-4\delta)/8-\varepsilon$. We also show that by allowing codes over $\mathbb{F}_{q^2}$ that are linear over $\mathbb{F}_q$, we can improve the rate to $(1-\delta)/4-\varepsilon$ while not sacrificing efficiency. Using this latter result, together with a careful code concatenation and placing buffers, we construct fully linear codes over $\mathbb{F}_2$ that can efficiently correct up to $\delta < 1/54$ fraction of deletions and have rate $R = (1-54\cdot\delta)/1216$. Cheng, Guruswami, Haeupler, and Li [CGHL21] constructed codes with (extremely small) rates bounded away from zero that can correct up to a $\delta < 1/400$ fraction of insdel errors. They also posed the problem of constructing linear codes that get close to the *half-Singleton bound* (proved in [CGHL21]) over small fields. Thus, our results significantly improve their construction and get much closer to the bound.

3.1.1 Insertion and Deletions

We denote the ith symbol of a string s (or of a vector v) as s_i (equivalently v_i). Throughout this chapter, we shall move freely between representations of vectors as strings and vice versa. Namely, we shall view each vector $v = (v_1, \ldots, v_n) \in \mathbb{F}_q^n$ also as a string by concatenating all the symbols of the vector into one string, i.e., $(v_1, \ldots, v_n) \leftrightarrow v_1 \circ v_2 \circ \ldots \circ v_n$. Thus, if we say that s is a subsequence of some vector v, we mean that we view v as a string and s is a subsequence of that string. A *run* r in a string s is a single-symbol substring of s such that the symbol before the run and the symbol after the run are different from the symbol of the run.

We first describe what are insertions and deletions and the metric that is used in this context.

Definition 3.1.1. *Let $s \in \Sigma^*$. The operation in which we remove a symbol from s is called a* deletion *and the operation in which we place a new symbol from Σ between two consecutive symbols in s is called an* insertion.

A substring *of s is a string obtained by taking consecutive symbols from s. A* subsequence *of s is a string obtained by removing some (possibly none) of the symbols in s.*

Note a symbol substitution can be achieved via a deletion followed by an insertion at the same location (an erasure can be interpreted as a deletion together with information of where this deletion has taken place). However, insertions and deletions are much harder to deal with than substitutions. A crucial observation is that they can affect the length of the string received (unlike substitutions or erasures). Thus, we can not use the usual hamming distance. The metric that is used in this scenario is the *edit distance* metric which is closely related to the problem of finding the longest common subsequence

Definition 3.1.2. *Let $s, s' \in \Sigma^*$. A* longest common subsequences *between s and s', is a subsequence s_{sub} of both s and s', of maximal length. We denote by $|\text{LCS}(s, s')|$ the length of a longest common subsequence.*[1]

The edit distance *between s and s', denoted by $ED(s, s')$, is the minimal number of insertions and deletions needed in order to turn s into s'.*

Lemma 3.1.3 (See e.g. Lemma 12.1 in [CR03])**.** *It holds that* $\text{ED}(s, s') = |s| + |s'| - 2\,|\text{LCS}(s, s')|$.

This implies that in order to compute the edit distance, it suffices to compute the LCS for which we have a dynamic programming algorithm that runs in time $O(m^2)$ where m is the length of the strings.

3.1.2 Linear codes

Linear codes are desirable for many reasons: they have a compact representation (they are determined by their generating matrix), they are efficiently encodable, and in some settings, we even have linear codes with linear encoding and decoding time, and often they are simpler to analyze.

3.1.3 Basic definitions and notation

A linear code over a field $\mathbb{F}$ is a linear subspace $\mathcal{C} \subseteq \mathbb{F}^n$. The rate of a linear code $\mathcal{C}$ of block length n is $\mathcal{R} = \dim(\mathcal{C})/n$. Every linear code of dimension k can be described as the image of a linear map, which, abusing notation, we also denote with $\mathcal{C}$, i.e., $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$. Equivalently, a linear code $\mathcal{C}$ can be defined by a *parity check matrix* H such that $x \in \mathcal{C}$ if and only if $Hx = 0$. The minimal distance of $\mathcal{C}$ with respect to a metric $d(\cdot, \cdot)$ is defined as $\text{dist}_{\mathcal{C}} := \min_{v \neq u \in \mathcal{C}} d(v, u)$. When $\mathcal{C} \subseteq \mathbb{F}_q^n$ has dimension k and minimal distance d we say that it is an $[n, k, d]_q$ code, or simply an $[n, k]_q$ code.

3.1.4 Previous results

In [AGFC07], it was shown that linear codes that can correct even one deletion, have a rate at most $1/2$, which is achieved by a trivial repetition code. More generally, in [CGHL21], it was shown that codes that can decode from a δ fraction of insdel errors cannot have rate larger than $(1 - \delta)/2 + o(1)$ where the $o(1)$ term goes to zero as the block length tends to infinity. This bound is called the "half-Singleton bound," and it is in sharp contrast to the fact that nonlinear insdel codes, or even *affine* codes (codes that form an affine space) can achieve rate close to 1 while still being able to decode from a constant fraction of insdel errors [CGHL21].

Theorem 3.1.4 (Theorem 4.2 in [CGHL21])**.** *For any $\delta > 0$ and prime power q, there exists a family of linear codes over $\mathbb{F}_q$ that can correct up to δn insertions and deletions, with rate $(1 - \delta)/2 - h(\delta)/\log_2(q)$.*

The proof of Theorem 3.1.4 uses the probabilistic method, showing that, with high probability, a random linear map generates such code. Complementing their result, they proved that their construction is almost tight. Specifically, they provided the following upper bounds, which they call the "half-Plotkin bound and the "half-Singleton bound," respectively.

Theorem 3.1.5 (Half-Plotkin bound: Theorem 5.1 in [CGHL21])**.** *Fix a finite field $\mathbb{F}_q$. Every $\mathbb{F}_q$-linear code that is capable of correcting a $\delta > 0$ fraction of insdel errors has rate at most $\frac{1}{2}(1 - \frac{q}{q-1}\delta) + o(1)$.*

Theorem 3.1.6 (Half-Singleton bound: Corollary 5.1 in [CGHL21])**.** *Every linear insdel code that is capable of correcting a δ fraction of deletions has rate at most $(1 - \delta)/2 + o(1)$.*

The authors in [CGHL21] also constructed explicit linear codes for insdel errors, given next.

Theorem 3.1.7. *[CGHL21, Theorem 1.4] Fix any finite field $\mathbb{F}_q$. There is an explicit construction of an $\mathbb{F}_q$-linear code family with rate bounded away from zero, a linear time encoding algorithm, and a polynomial time decoding algorithm to correct a positive constant fraction of insertions and deletions. The generator matrix of codes in the family can be deterministically computed in polynomial time in the code block length.*

[1] Note that a longest common subsequence may not be unique as there can be a number of subsequences of maximal length.

These codes have rate $\mathcal{R} < 2^{-80}$ [GK], a linear time encoding algorithm, and an $O(n^4)$ time decoding algorithm from a $\delta < 1/400$ fraction of insdel errors. Improving upon these parameters was left as an open an question in [CGHL21].

3.1.5 Our results

In this chapter, we improve the results presented in [CGHL21]. We give explicit constructions of codes over small fields that are efficient (namely, have polynomial-time encoding and decoding algorithms) and almost attain the half-singleton bound. Specifically,

Theorem 3.1.8. *For every small enough constant $\varepsilon > 0$, $\delta \in (0, 1/4)$ and $q = \mathrm{poly}(1/\varepsilon)$, there is an explicit construction of a linear code over $\mathbb{F}_q$ of rate $\mathcal{R} > (1 - 4\delta)/8 - \varepsilon$ that can correct from a δ fraction of adversarial insdel errors. Furthermore, the running time of the decoding algorithm is $O(n^3)$.*

By relaxing the linearity requirement, we construct "half-linear" codes. We say that a code is half-linear when it is defined over the field $\mathbb{F}_{q^2}$ and is linear over $\mathbb{F}_q$. The half-linear codes that we construct can decode from any δ fraction of insdel errors, and their rate is close to $(1 - \delta)/4$.

Theorem 3.1.9. *For every small enough constant $\varepsilon > 0$, $\delta \in (0, 1)$, and $q = \mathrm{poly}(1/\varepsilon)$ there is an explicit construction of a code over $\mathbb{F}_{q^2}$, which is linear over the subfield $\mathbb{F}_q$, that has rate $\mathcal{R} > (1-\delta)/4-\varepsilon$ and can correct from δ fraction of insdel errors. Furthermore, the running time of the decoding algorithm is $O(n^3)$.*

Using this construction, we obtain linear binary codes that can efficiently correct from adversarial deletions.

Theorem 3.1.10. *There exists an explicit linear binary code that can correct from $\delta < 1/54$ fraction of adversarial deletions in $O(n^3)$ time and has the rate $\mathcal{R} = (1 - 54 \cdot \delta)/1216$.*

While the algorithm in Theorem 3.1.10 is only guaranteed to decode from deletions, we note that information-theoretically, the code can also decode from 1/54 fraction of adversarial insdel errors, as the claim implies a lower bound on the edit distance between any two codewords.

Theorems 3.1.8 and 3.1.10 improve upon the (explicit) constructions of linear codes given in [CGHL21], which can handle a fraction $\delta < 1/400$ of insdel errors and whose rate is $< 2^{-80}$. We note, however, that Theorem 3.1.10 only gives an efficient decoder against deletions, whereas the algorithm in [CGHL21] decodes from both insertions and deletions.

3.1.6 Proof idea

We first observe that it is easy to construct codes against deletions from any code that can correct erasures: simply add indices to the coordinates of each codeword. Specifically, if $\mathcal{C}$ is a code that can correct from e erasures, then we can consider the following code

$$\mathcal{C}' = \{((1, c_1), \ldots, (n, c_n)) \mid c \in \mathcal{C}\} \ .$$

It is easy to see that this code can decode from e adversarial deletions - the missing indices indicate the location of the deletions, and therefore we can treat them as erasures. With a slightly more advanced algorithm, this code can also decode from adversarial insertions (for this to work, we need a code that can decode from errors as well). This construction has two problems. The first is that it is not linear. The second is that it requires an alphabet of size $\Omega(n)$.

The problem of linearity can be solved as follows. Assume $\mathcal{C} \subseteq \mathbb{F}_q^n$ is linear. To add indices while preserving linearity we replace (i, c_i) with $(c_i, i \cdot c_i)$. Observe that the resulting code is linear over $\mathbb{F}_q$, but symbols of the codeword are in $\mathbb{F}_{q^2}$. We shall call such codes half-linear codes. To make the code fully linear, we replace each symbol $(c_i, i \cdot c_i)$ with two symbols, c_i and $i \cdot c_i$. The problem is that now, after adversarial deletions, it is unclear which indices "survived" and which were deleted or corrupted. To overcome this difficulty, we add small "buffers" of zeros between the different indices. That is, the new

codeword is $(c_1, 1 \cdot c_1, 0, 0, c_2, 2 \cdot c_2, 0, 0, c_3, \ldots)$. Note that we still need a large alphabet to have n different field elements that can serve as indices.

To reduce the alphabet size, we use synchronization strings instead of field elements for the indices. Synchronization strings were defined in the breakthrough work of Haeupler and Shahrasbi [HS17].

Definition 3.1.11. *A string $S \in \Sigma^n$ is called an ε-synchronization string if for every $1 \leq i < j < k \leq n+1$ it holds that $ED(S[i,j), S[j,k)]) > (1-\varepsilon) \cdot (k-i)$, where $S[i,j)$ denotes the string $S_i \circ S_{i+1} \circ \cdots \circ S_{j-1}$ and S_i is the ith coordinate of S.*

Haeupler and Shahrasbi proved the existence of such strings and gave a polynomial-time randomized algorithm for constructing them. An explicit construction, with improved alphabet size, was given in [CHL$^+$19].

Theorem 3.1.12 (Theorem 1.2 in [CHL$^+$19])**.** *For every $n \in \mathbb{N}$ and for every $\varepsilon \in (0,1)$, there is a polynomial time (in n) deterministic construction of an ε-synchronization string, of length n, over an alphabet of size $O(\varepsilon^{-2})$.*

In [HS17] Haeupler and Shahrasbi showed that synchronization strings could be used instead of indices. Specifically, they proved that if $\mathcal{C}$ can decode from d hamming errors and e erasures, for $2d + e < \delta$, and $S = (S_1 S_2 \ldots S_n)$ is an ε-synchronization string, then the code

$$\mathcal{C}^{\mathrm{ID}} := \{((S_1, c_1), \ldots, (S_n, c_n)) \mid c \in \mathcal{C}\} \ , \tag{3.1}$$

can decode from $(\delta - O(\sqrt{\varepsilon}))n$ insdel errors.

Theorem 3.1.13 ([HS21])**.** *Let $\delta, \varepsilon \in (0,1)$ and let S be an ε-synchronization string. Let $\mathcal{C}$ be a code that can decode, in time $T(n)$, from d hamming errors and e erasures, where $2d + e < \delta n$. Then, the code $\mathcal{C}^{\mathrm{ID}} := \{((S_1, c_1), \ldots, (S_n, c_n)) \mid c \in \mathcal{C}\}$ can decode from $(\delta - 12\sqrt{\varepsilon})n$ insdel errors in time $O(n^2/\sqrt{\varepsilon}) + T(n)$.*

We note that this code is not linear, even when $\mathcal{C}$ is a linear code, as the synchronization string S is fixed. However, as outlined above, we can tweak this construction to make the code linear while still maintaining its decoding property. We combine this idea with an algebraic geometry code (AG-code) as the base code $\mathcal{C}$ to obtain our results. We choose these codes as our base codes as they have the best-known rate-distance tradeoff, and in addition, they come with efficient decoding algorithms. Thus, codewords of our code have the form

$$\mathcal{C}' = \{(c_1, S_1 \cdot c_1, 0, 0, \ldots, 0, 0, c_n, S_n \cdot c_n) \mid c \in \mathcal{C}\} \ .$$

To further reduce the alphabet to binary, we perform two additional steps. First, we concatenate our code from Theorem 3.1.9 with a carefully chosen binary code of fixed length. Then we add *buffers* of zeroes between any two concatenated words. A short buffer between the encodings of c_i and $S_i \cdot c_i$ and a long buffer between the encodings of $S_i \cdot c_i$ and c_{i+1}. The buffers allow our decoding algorithm to correctly identify the encoding of many pairs $(c_i, S_i \cdot c_i)$. Then, by using the synchronization string, S, and the decoder of $\mathcal{C}$, we obtain a decoding algorithm.

3.1.7 Organization of the chapter

The chapter is organized as follows. In Section 3.2 we construct linear (and half-linear) codes, over small alphabets, that can handle insdel errors, and prove Theorem 3.1.9 and Theorem 3.1.8. In Section 3.3, we give the construction of linear binary codes that can decode from deletions, thus proving Theorem 3.1.10.

3.2 Linear Insdel Codes over Finite Alphabet via Synchronization Strings

In this section, we prove Theorems 3.1.8 and 3.1.9. We follow the strategy outlined in Section 3.1.6.

Algorithm 1: Decode $\mathcal{C}'$

input : A corrupted codeword $y = (e_1, \ldots, e_t)$.
output: A message $x \in \mathbb{F}_q^k$.

[1] Set L to be an empty list
[2] **for** $i = 1, \ldots, t$ **do**
  Let $e_i = (a, b)$
  if $b = 0$ **then**
    Go to the next i
  end
  Add to L the tuple $(b/a, a)$
end
[3] If L is empty, return the zero codeword $c = 0$; else decode L using the decoding algorithm of C^{ID} given in Theorem 3.1.13.
[4] Let $c^{\mathrm{ID}} = ((S_1, c_1), \ldots, (S_n, c_n))$ be the decoded codeword. Return the codeword $c = ((c_1, S_1 c_1), \ldots, (c_n, S_n c_n))$.

As our base code $\mathcal{C}$, we shall use an AG-code. The well-known constructions of [TVZ82, GS95, GS96, SAK+01] beat the Gilbert-Varshamov bound[2] over $\mathbb{F}_q$, for $q \geq 49$. Moreover, these code have an efficient decoder that can correct both errors and erasures, almost up to its correction capability [SV90, Kot96]. The interested reader is referred to [Sti09] for further information on AG-codes and their decoding.

Theorem 3.2.1 ([TVZ82, SV90, Kot96])**.** *Let $q = p^{2m}$ be a square where p is a prime and m is a positive integer. For every $0 < \delta \leq 1 - \frac{1}{\sqrt{q}-1}$ there exists an explicit linear code $\mathcal{C}$ over $\mathbb{F}_q$, of minimal distance δ and rate*

$$\mathcal{R} \geq 1 - \frac{1}{\sqrt{q}-1} - \delta \, .$$

Moreover, there is a decoding algorithm that runs in time $O(n^3)$ and can correct from d hamming errors and e erasures, for $2d + e < \left(\delta - \frac{1}{\sqrt{q}-1}\right) n$.

We first prove Theorem 3.1.9 as the proof of its decoding algorithm is easier and then prove Theorem 3.1.8.

3.2.1 Half-linear insdel codes

Construction 3.2.2. *Let $\delta \in (0, 1)$ and ε a small constant. Let p be a prime such that $p = \Theta(\varepsilon^{-2})$ and set $q = p^2 = \Theta(\varepsilon^{-4})$. Set $\delta_{\mathcal{C}} = (1 + \delta + 13\varepsilon)/2$ and let $\mathcal{C}$ be the code from Theorem 3.2.1, defined over the finite field $\mathbb{F}_q$, with rate $\mathcal{R}_{\mathcal{C}} > 1 - \delta_{\mathcal{C}} - \varepsilon$. Let $S = (S_1 S_2 \ldots S_n)$ be an ε^2-sync string, where $S_i \in \mathbb{F}_q \setminus \{0\}$ for all $i \in [n]$. Let $Enc_{\mathcal{C}} : \mathbb{F}_q^k \to \mathbb{F}_q^n$ be the encoding map of $\mathcal{C}$. We define the code $\mathcal{C}'$ via the encoding map $Enc_{\mathcal{C}'}$: For $a \in \mathbb{F}_q^k$, let $Enc_{\mathcal{C}}(a) = c = (c_1, \ldots, c_n)$. Then,*

$$Enc_{\mathcal{C}'}(a) = ((c_1, S_1 \cdot c_1), (c_2, S_2 \cdot c_2), \ldots, (c_n, S_n \cdot c_n)) \ . \tag{3.2}$$

Namely, $\mathcal{C}'$ is the image of $\mathbb{F}_q^k$ under $\mathrm{Enc}_{\mathcal{C}'}$. One can easily observe that the rate $\mathcal{C}'$ is $\mathcal{R}_{\mathcal{C}'} = \mathcal{R}_{\mathcal{C}}/2 > (1 - \delta)/4 - 4\varepsilon$, and that the code is linear over $\mathbb{F}_q$.

Proposition 3.2.3. *Algorithm 1 runs in time $O(n^3)$ and can decode $\mathcal{C}'$ (given in Construction 3.2.2), from δn worst case insdel errors.*

Proof. For $c = ((c_1, S_1 c_1), \ldots, (c_n, S_n c_n)) \in \mathcal{C}'$ let $c^{\mathrm{ID}} = ((S_1, c_1), \ldots, (S_n, c_n)) \in \mathcal{C}^{\mathrm{ID}}$. Observe that $\mathcal{C}^{\mathrm{ID}}$ is as in Equation (3.1). To prove the claim we shall interpret insdel errors in $\mathcal{C}'$ as insdel errors in $\mathcal{C}^{\mathrm{ID}}$ and then apply Theorem 3.1.13.

[2] The Gilbert-Varshamov bound shows what parameters random (linear) codes achieve.

Assume first that the corrupted codeword is the zero vector. Then, since the hamming-weight of each nonzero codeword of $\mathcal{C}'$ is at least $\delta_{\mathcal{C}} n > \delta n$, the only codeword that would produce this corrupted codeword from δn insdel errors is the zero codeword, hence, in Step 3 successfully decodes the zero codeword. Next, we assume that the corrupted codeword is *not* the zero vector.

The map $(a, b) \to (b/a, a)$ maps each nonzero coordinate of $c \in \mathcal{C}'$ to the corresponding coordinate of $c^{\mathrm{ID}} \in \mathcal{C}^{\mathrm{ID}}$ and therefore, by applying it coordinate-wise, we can interpret any insdel error to c as an insdel error to c^{ID}.

Observe that in addition to the errors introduced by the adversary, in Step 2 of Algorithm 1 we treat any zero coordinate as a deletion. Since the minimal distance of $\mathcal{C}$ is $\delta_{\mathcal{C}}$, a nonzero codeword $c \in \mathcal{C}'$ has at most $n(1 - \delta_{\mathcal{C}})$ zero coordinates. Therefore, Step 2 can cause $(1 - \delta_{\mathcal{C}})n$ additional insdel errors. In conclusion,

$$\mathrm{ED}(c^{\mathrm{ID}}, L) \leq (1 - \delta_{\mathcal{C}} + \delta)n = (\delta_{\mathcal{C}} - 13\varepsilon)n \ ,$$

where the equality follows from the choice of $\delta_{\mathcal{C}}$ in Construction 3.2.2. As $\mathcal{C}$ can correct from d hamming errors and e erasures, for $2d + e \leq (\delta_{\mathcal{C}} - \varepsilon)n$, Theorem 3.1.13 implies that Step 3 succeeds, and the decoder outputs c^{ID}. Step 4 clearly returns the codeword c.

To prove the claim regarding the running time we note that Steps 1 and 2 take linear time and that by Theorem 3.2.1, the decoding algorithm of Theorem 3.1.13 runs in time $O(n^2/\varepsilon) + O(n^3) = O(n^3)$. □

Remark 3.2.4. *As the proof shows, Step 2 of Algorithm 1 ignores the symbol* $(0, 0)$*. In other words, it treats this symbol as a deletion. Thus, from the point of view of the adversary, there is no need to corrupt the zero symbol.*

Proof of Theorem 3.1.9. The proof follows immediately from Construction 3.2.2 and Proposition 3.2.3. Indeed, the code described in Construction 3.2.2 maps k symbols of $\mathbb{F}_q$ to n symbols of $\mathbb{F}_{q^2}$ and hence its rate is $k/(2n)$. As k was chosen so that $\mathcal{R}_{\mathcal{C}} = k/n > 1 - \delta_{\mathcal{C}} - \varepsilon = (1 - \delta - 15\varepsilon)/2$, we get that $\mathcal{R}_{\mathcal{C}'} = \mathcal{R}_{\mathcal{C}}/2 > (1 - \delta)/4 - 4\varepsilon$. By construction the code is linear over $\mathbb{F}_q$. □

3.2.2 Full linear insdel codes

We next prove Theorem 3.1.8. As described in Section 3.1.6, to get full linear insdel codes we use a similar construction albeit with two significant modifications: First, we "flatten" the code, i.e., we expand each symbol $(c_i, S_i \cdot c_i) \in \mathbb{F}_{q^2}$ to two symbols $c_i, S_i \cdot c_i \in \mathbb{F}_q$. Secondly, to protect our codeword from insdel errors, we additionally insert two zeros between every two adjacent pairs. Thus, the corresponding word to $((c_1, S_1 \cdot c_1), (c_2, S_2 \cdot c_2), \ldots, (c_n, S_n \cdot c_n))$ is $(c_1, S_1 \cdot c_1, 0, 0, c_2, S_2 \cdot c_2, 0, 0, \ldots, c_n, S_n \cdot c_n)$. It is clear that in this way we get a linear code. Formally:

Construction 3.2.5. *Let* $\delta \in (0, 1/4)$ *and* ε *a small enough constant. Set* $\delta_{\mathcal{C}} = (1 + 4\delta + 13\varepsilon)/2 < 1$*. Let* p *be a prime such that* $p = \Theta(\varepsilon^{-2})$ *and set* $q = p^2$*. Let* $\mathcal{C}$ *be the code from Theorem* 3.2.1*, defined over the finite field* $\mathbb{F}_q$*, with minimal distance* δ_C *and rate* $\mathcal{R}_{\mathcal{C}} = 1 - \delta_{\mathcal{C}} - \varepsilon$*. Let* $S = (S_1 S_2 \ldots S_n)$ *be an* ε^2*-sync string, where* $S_i \in \mathbb{F}_q \backslash \{0\}$ *for all* $i \in [n]$ *Let* $Enc_{\mathcal{C}} : \mathbb{F}_q^k \to \mathbb{F}_q^n$ *be the encoding map of* $\mathcal{C}$*. We define the code* $\mathcal{C}''$ *via the encoding map* $Enc_{\mathcal{C}''}$*: For* $a \in \mathbb{F}_q^k$*, let* $Enc_{\mathcal{C}}(a) = c = (c_1, \ldots, c_n)$*. Then,*

$$Enc_{\mathcal{C}''}(a) = (c_1, S_1 \cdot c_1, 0, 0, c_2, S_2 \cdot c_2, 0, 0, \ldots, c_n, S_n \cdot c_n) \ . \tag{3.3}$$

Namely, $\mathcal{C}''$ is the image of $\mathbb{F}_q^k$ under $\mathrm{Enc}_{\mathcal{C}''}$. Clearly, $\mathcal{C}'' \subset \mathbb{F}_q^{4n-2}$ is an $\mathbb{F}_q$ linear space.

Proposition 3.2.6. *Algorithm 2 runs in time* $O(n^3)$ *and can decode* $\mathcal{C}''$*, given in Construction 3.2.5, from* δn *worst case insdel errors.*

Proof. Let $c = (c_1, S_1 c_1, 0, 0, \ldots, 0, 0, c_n, S_n c_n) \in \mathcal{C}''$ and denote by $c^{\mathrm{ID}} = ((S_1, c_1), \ldots, (S_n, c_n)) \in \mathcal{C}^{\mathrm{ID}}$ the corresponding codeword, where $\mathcal{C}^{\mathrm{ID}}$ is as in the proof of Proposition 3.2.3. We will follow the same reasoning as in the proof of Proposition 3.2.3; translate insdel errors in $\mathcal{C}''$ to insdel errors in $\mathcal{C}^{\mathrm{ID}}$, and then apply Theorem 3.1.13.

Algorithm 2: Decode $\mathcal{C}''$

input : A corrupted codeword $y = (e_1, \ldots, e_t)$.

output: A message $x \in \mathbb{F}_q^k$.

[1] Set L to be an empty list

Remove any run of zeros at the beginning and the end of y, and write y as

$$y = s_1 \circ \bar{0} \circ s_2 \circ \bar{0} \circ \cdots \circ \bar{0} \circ s_m,$$

where $0 \leq m \leq t$, the s_i's are strings of symbols that do not contain any 0's, and the notation $\bar{0}$ corresponds to a string of consecutive zeros of any length.

[2] **for** $i = 1, \ldots, m$ **do**

  if $|s_i| \neq 2$ **then**

    Continue

  end

  Let a, b be the first and second elements in s_i. Add to L the tuple $(b/a, a)$

end

[3] If L is empty, return the zero codeword, $c = 0$; else decode L using the algorithm of C^{ID} given in Theorem 3.1.13.

[4] Let $c^{\mathrm{ID}} = ((S_1, c_1), \ldots, (S_n, c_n))$ be the decoded codeword. Return the codeword $c = (c_1, S_1 c_1, 0, 0, \ldots, 0, 0, c_n, S_n c_n)$.

Assume first that the corrupted codeword is the zero vector. Then, since the hamming-weight of each nonzero codeword of $\mathcal{C}$ is at least $\delta_{\mathcal{C}} n$ and $S_i \neq 0$ for each i, the normalized minimum distance of $\mathcal{C}''$ is at least $2\delta_{\mathcal{C}} n/(4n-2) > \delta_{\mathcal{C}}/2$. On the other hand,

$$\delta < \frac{1}{4} < \frac{1 + 4\delta + 13\varepsilon}{4} = \frac{\delta_{\mathcal{C}}}{2}.$$

Hence, the only codeword that would produce this corrupted codeword from $\delta(4n-2)$ insdel errors is the zero codeword, and Step 3 successfully decodes the zero codeword. Next, we assume that the corrupted codeword is *not* the zero vector.

Since the minimal distance of $\mathcal{C}$ is at least $\delta_{\mathcal{C}} n$, any nonzero $c \in \mathcal{C}''$ contains at most $n(1 - \delta_{\mathcal{C}})$ pairs $c_i, S_i c_i$ that are equal to $0, 0$. Every such zero pair is interpreted as a deletion in Step 2 of Algorithm 2. These deletions are in addition to those made by the adversary. The adversary, who knows the decoding algorithm, will clearly ignore the zero pairs $c_i, c_i S_i$ for $c_i = 0$, and therefore will either "ruin" nonzero pairs by converting them to nonzero blocks (i.e., blocks with no zeros) of lengths different than 2, or by constructing erroneous pairs.

The most economic way to construct the former is by inserting (deleting) a symbol to (from) an existing nonzero pair, respectively. This increases $\mathrm{ED}(c^{\mathrm{ID}}, L)$ by 1. Also, the adversary can merge, say $b \geq 2$ consecutive blocks, into a single block by deleting the buffers between them. This "costs" $2(b-1)$ deletions that translate to an increase to $\mathrm{ED}(c^{\mathrm{ID}}, L)$ by b. Hence, on average, each deletion or insertion in a nonzero block of length different than 2 increases the edit distance by at most 1.

The construction of the latter, i.e., an erroneous pair, would clearly cost 2 insertions between the zeros of a buffer or by a symbol deletion from an existing nonzero pair, followed by a new nonzero symbol insertion. This is clearly less economical than ruining nonzero pairs, since in this case, on average, in order to increase $\mathrm{ED}(c^{\mathrm{ID}}, L)$ by 1, the adversary must perform two edit operations.

To conclude, the accounting above indicates that every insdel error made by the adversary increases the edit distance between L and c^{ID} by at most one. It follows that after the adversary performs $\delta \cdot (4n-2)$ insdel errors (recall that $c \in \mathbb{F}_q^{4n-2}$),

$$\mathrm{ED}(c^{\mathrm{ID}}, L) \leq (1 - \delta_{\mathcal{C}})n + 4\delta n = (\delta_{\mathcal{C}} - 13\varepsilon)n \ .$$

Thus, by Theorem 3.1.13 and since the code $\mathcal{C}$ can correct from d hamming errors and e erasures where $2d + e \leq (\delta_{\mathcal{C}} - \varepsilon)n$, Steps 3 and 4 succeed.

The claim regarding the running time follows exactly as in the proof of Proposition 3.2.3. □

We now conclude the proof of Theorem 3.1.8.

Proof of Theorem 3.1.8. As before, the proof is immediate from Construction 3.2.5 and Proposition 3.2.6. The rate satisfies

$$\mathcal{R}_{\mathcal{C}''} = \frac{R_{\mathcal{C}}}{4} > \frac{1 - \delta_{\mathcal{C}} - \varepsilon}{4} = \frac{1 - 4\delta - 15\varepsilon}{8} \geq \frac{1 - 4\delta}{8} - 2\varepsilon \,. \qquad \square$$

3.3 Binary Linear Codes

In this section we prove Theorem 3.1.10. To ease the reading, we repeat the statement of the theorem.

Theorem 3.1.10. *There exists an explicit linear binary code that can correct from $\delta < 1/54$ fraction of adversarial deletions in $O(n^3)$ time and has the rate $\mathcal{R} = (1 - 54 \cdot \delta)/1216$.*

As explained in Section 3.1.6 our construction concatenates the code of Theorem 3.1.9 with an adequately chosen short binary code and then adds buffers between the encoding of different symbols: short buffers between the encoding of c_i and $S_i \cdot c_i$ and long buffers between the encodings of $S_i \cdot c_i$ and c_{i+1}. The specially tailored inner code is a linear binary code that can correct from a small fraction of insdel errors and has the property that, with the exception of the zero word, no codeword has large runs of zeroes. We shall prove that such codes exist and then construct one greedily.

3.3.1 The inner code

The following proposition describes the properties that our inner code should possess and is proved using the probabilistic method. As the code has a fixed length, we shall use the brute force algorithm to construct it.

Proposition 3.3.1. *Set $\delta_{\text{in}} = 1/6$ and $\rho = 1/17$. There exists $m_0 \in \mathbb{N}$ such that for any $m > m_0$, which is a multiple of 102,[3] there is a binary linear code $\mathcal{C}_{\text{in}} \subset \{0,1\}^m$ of rate $\mathcal{R}_{\text{in}} = \delta_{\text{in}}/16$ such that*

1. *For any two substrings c_s, c'_s of any two distinct codewords $c \neq c' \in \mathcal{C}_{\text{in}}$ such that $|c_s|, |c'_s| \geq (1 - 2\delta_{\text{in}} + \rho)m$, it holds that $LCS(c_s, c'_s) < \min(|c_s|, |c'_s|) - \rho m$.*
2. *Any substring c_{sub} of length $\delta_{\text{in}} m$, of any nonzero codeword $c \in \mathcal{C}_{\text{in}}$ contains at least $\rho m + 1$ ones.*

Observe that Proposition 3.3.1(1) implies that $\text{ED}(c_s, c'_s) > 2\rho m$ so in particular we can brute force correct any ρm insdel errors in $\mathcal{C}_{\text{in}}$ in time $\exp(m)$.

Proof. Let $G \in \mathbb{F}_2^{m \times \mathcal{R}_{\text{in}} m}$ be a uniformly chosen random matrix. G will serve as a generator matrix for a linear code $\mathcal{C}$, i.e., $\mathcal{C} = \{Gv \mid v \in \mathbb{F}_2^{\mathcal{R}_{\text{in}} m}\}$. We next prove that the probability that $\mathcal{C}$ does not satisfy any of the properties in the proposition is small.

The proof that Proposition 3.3.1(1) holds with high probability relies on the following simple and intuitive claim given in [CGHL21].

Claim 3.3.2 (Claim 4.1 of [CGHL21])**.** *Let $\mathcal{C}$ be a random linear code and let $c \neq c'$ be any two distinct codewords. Fix two sets of indices $\{s_1, \ldots, s_t\}, \{s'_1, \ldots, s'_t\} \subset [n]$. Then,*

$$\Pr[\forall i \in [t], (c)_{s_i} = (c')_{s'_i}] \leq 2^{-t} \,.$$

[3] We require this to ensure that both ρm and $\delta_{\text{in}} m$ are integers, in order to avoid the use of ceilings and floors.

Let $c \neq c' \in \mathcal{C}$ be distinct and c_s and c'_s be substrings of c and c', such that $r = \min(|c_s|, |c'_s|) \geq (1 - 2\delta_{\text{in}} + \rho)m$. Let $\{s_1, \ldots, s_{r-\rho m}\}$ and $\{s'_1, \ldots, s'_{r-\rho m}\}$ be two sequences of indices. The claim implies that,

$$\Pr\left[\forall i \in [r - \rho m], (c_s)_{s_i} = (c'_s)_{s'_i}\right] \leq 2^{-(r-\rho m)} \leq 2^{-(1-2\delta_{\text{in}})m} .$$

By the union bound, the probability that c_s and c'_s share a common subsequence of length $(r - \rho m)$ is at most

$$\binom{r}{r-\rho m}^2 \cdot 2^{-(1-2\delta_{\text{in}})m} \leq 2^{m\cdot(2h(\rho)-(1-2\delta_{\text{in}}))} ,$$

where we used $\binom{r}{r-\rho m} = \binom{r}{\rho m} \leq \binom{m}{\rho m}$. Now, the number of substrings of c (c') of length $\geq (1 - 2\delta_{\text{in}} + \rho)m$ is at most $m^2 \cdot (2\delta_{\text{in}} - \rho)$ and the number of codewords is $2^{\mathcal{R}_{\text{in}} m}$. Thus, the probability that there exist $c \neq c'$, and substrings c_s and c'_s of c and c', respectively, such that $|c_s|, |s'_s| \geq (1 - 2\delta_{\text{in}} + \rho)m$ and they share a common subsequence of length $r - \rho m$ is at most

$$2^{2m\cdot\mathcal{R}_{\text{in}}} \cdot m^2 \cdot 2^{m\cdot(2h(\rho)-(1-2\delta_{\text{in}}))} = 2^{2m\cdot\left(\mathcal{R}_{\text{in}}+h(\rho)-(1-2\delta_{\text{in}})/2+\frac{O(\log m)}{m}\right)} .$$

Thus, as long as

$$\mathcal{R}_{\text{in}} + h(\rho) - (1 - 2\delta_{\text{in}})/2 < 0 , \tag{3.4}$$

there exists $m'_0 \in \mathbb{N}$ such that for every integer $m \geq m'_0$, the probability that Proposition 3.3.1(1) does not holds is smaller than $1/4$.

To prove that Proposition 3.3.1(2) holds with high probability, consider any $0 \neq v \in \mathbb{F}_2^{\mathcal{R}_{\text{in}} m}$. As G was chosen uniformly at random, Gv is uniform random vector in $\mathbb{F}_2^m$. The probability that Gv contains a substring of length $\delta_{\text{in}} m$ that has $\leq \rho m$ ones is at most

$$m \cdot \sum_{i=0}^{\rho m} \binom{\delta_{\text{in}} m}{i} 2^{-\delta_{\text{in}} m} \leq m(\rho m + 1) \cdot \binom{\delta_{\text{in}} m}{\rho m} \cdot 2^{-\delta_{\text{in}} m} \leq 2^{\delta_{\text{in}} m\left(-1+h\left(\frac{\rho}{\delta_{\text{in}}}\right)+\frac{O(\log(m))}{m}\right)} .$$

Thus, by the union bound, the probability that there exists $v \in \mathbb{F}_2^{\mathcal{R}_{\text{in}} m} \setminus \{0\}$, such that Gv contains a substring of length $\delta_{\text{in}} m$ with $\leq \rho m$ ones is at most

$$2^{m\left(\mathcal{R}_{\text{in}}-\delta_{\text{in}}+\delta_{\text{in}} h\left(\frac{\rho}{\delta_{\text{in}}}\right)+\frac{O(\log(m))}{m}\right)} .$$

Hence, if

$$\mathcal{R}_{\text{in}} - \delta_{\text{in}} + \delta_{\text{in}} h\left(\frac{\rho}{\delta_{\text{in}}}\right) < 0 \tag{3.5}$$

then there exists $m''_0 \in \mathbb{N}$ such that for every integer $m \geq m''_0$, the probability that $\mathcal{C}$ does not satisfy this property is $\leq 1/4$.

It can be verified that for $\delta_{\text{in}} = 1/6$, $\rho = 1/17$, $\mathcal{R}_{\text{in}} = \delta_{\text{in}}/16$, and $m_0 = \max(m'_0, m''_0)$, inequalities (3.4) and (3.5) hold true and therefore the probability that a random code $\mathcal{C}$ satisfies both properties is at least $1/2$ and the proposition follows. □

Construction and decoding To explicitly construct codes as in Proposition 3.3.1 we simply go over all possible linear codes and pick one that satisfies both properties. This requires $\exp(m^2)$ many steps. In our final construction we need $m = O(\log(1/\varepsilon_{\text{out}}))$ and hence the cost of constructing the inner code is $\exp(\log^2(1/\varepsilon_{\text{out}}))$.

Similarly, we decode from deletions using the following brute force algorithm: Set L' to be an empty list. On input $\tilde{c}$, the algorithm runs over every codeword $c \in \mathcal{C}$ and checks if $\tilde{c}$ is a subsequence of c. If the answer is yes and c is not in L', then the algorithm adds c to L'. If L' contains only c, then the algorithm returns c. Otherwise, it returns $\perp$. Clearly, the running time of this algorithm is $\exp(m) = \text{poly}(1/\varepsilon_{\text{out}})$.

Remark 3.3.3. *An important observation is that our decoding algorithm cannot output a wrong answer. Indeed, if $\tilde{c}$ was obtained from c by performing any number of deletions, then c will be one of the codewords in L' (as $\tilde{c}$ is a subsequence of c).*

3.3.2 Construction of our code

Let $\delta_{\text{out}} > 0$ and $\varepsilon_{\text{out}} < \delta_{\text{out}}/1400$ small enough. Let $\mathcal{C}_{\text{out}} \subset \mathbb{F}_{q^2}^n$ be the code given in Theorem 3.1.9, with parameters $\delta = \delta_{\text{out}}$ and $\varepsilon = \varepsilon_{\text{out}}$. Recall that the rate of $\mathcal{C}_{\text{out}}$ is $\mathcal{R}_{\text{out}} = (1 - \delta_{\text{out}})/4 - \varepsilon_{\text{out}}$ and the code is defined over the alphabet $\mathbb{F}_{q^2}$ where $q = \text{poly}(1/\varepsilon_{\text{out}})$. Denote $k = \mathcal{R}_{\text{out}} \cdot n$. Let $\mathcal{C}_{\text{in}} : \{0,1\}^{m \cdot \mathcal{R}_{\text{in}}} \to \{0,1\}^m$ be the code obtained in Section 3.3.1, where m is such that $\mathcal{R}_{\text{in}} m = \log(q)$ (we pick ε_{out} small enough so that $m \geq m_0$ as in Proposition 3.3.1).

Construction 3.3.4. *The encoding works as follows. Given a message $x \in \mathbb{F}_q^k$ we:*

1. *Encode x using the outer code $\mathcal{C}_{\text{out}}$ to obtain $\sigma = \mathcal{C}_{\text{out}}(x)$. Denote*

$$\sigma = ((\sigma_1, S_1 \cdot \sigma_1), \ldots, (\sigma_n, S_n \cdot \sigma_n)) \ .$$

2. *Let $0^{(\text{in})}$ denote a string of $2\delta_{\text{in}} m$ many zeroes. Encode every symbol $(\sigma_i, S_i \cdot \sigma_i)$ using the inner code to obtain $(\mathcal{C}_{\text{in}}(\sigma_i), \mathcal{C}_{\text{in}}(S_i \cdot \sigma_i))$ and place the string $0^{(\text{in})}$ between $\mathcal{C}_{\text{in}}(\sigma_i)$ and $\mathcal{C}_{\text{in}}(S_i \cdot \sigma_i)$. We refer to those $0^{(\text{in})}$ strings as* inner buffers. *At the end of this step we have the string*

$$\mathcal{C}_{\text{in}}(\sigma_1) \circ 0^{(\text{in})} \circ \mathcal{C}_{\text{in}}(S_1 \cdot \sigma_1) \circ \ldots \circ \mathcal{C}_{\text{in}}(\sigma_n) \circ 0^{(\text{in})} \circ \mathcal{C}_{\text{in}}(S_n \cdot \sigma_n) \ .$$

3. *Let $0^{(\text{out})}$ denote a string of $5\delta_{\text{in}} m$ many zeroes. Place the string $0^{(\text{out})}$ between every two adjacent symbols of the form $\mathcal{C}_{\text{in}}(S_i \cdot \sigma_i) \circ \mathcal{C}_{\text{in}}(\sigma_{i+1})$ to get*

$$\mathcal{C}_{\text{in}}(\sigma_1) \circ 0^{(\text{in})} \circ \mathcal{C}_{\text{in}}(S_1 \cdot \sigma_1) \circ 0^{(\text{out})} \circ \ldots \circ \mathcal{C}_{\text{in}}(\sigma_n) \circ 0^{(\text{in})} \circ \mathcal{C}_{\text{in}}(S_n \cdot \sigma_n) \ .$$

   We refer to those $0^{(\text{out})}$ strings as outer buffers.

The encoding of x is the string

$$\text{ENC}(x) = \mathcal{C}_{\text{in}}(\sigma_1) \circ 0^{(\text{in})} \circ \mathcal{C}_{\text{in}}(S_1 \cdot \sigma_1) \circ 0^{(\text{out})} \circ \ldots \circ \mathcal{C}_{\text{in}}(\sigma_n) \circ 0^{(\text{in})} \circ \mathcal{C}_{\text{in}}(S_n \cdot \sigma_n) \ .$$

Rate: The length of the codewords is $2mn + 2\delta_{\text{in}} mn + 5\delta_{\text{in}} m(n-1)$ bits. Recalling that $\log(q) = m \cdot \mathcal{R}_{\text{in}}$ we get

$$\begin{aligned} \mathcal{R} &= \frac{\log(q^{\mathcal{R}_{\text{out}} n})}{2mn + 2\delta_{\text{in}} mn + 5\delta_{\text{in}} m(n-1)} \\ &> \frac{\mathcal{R}_{\text{in}} \mathcal{R}_{\text{out}}}{2 + 7\delta_{\text{in}}} \ . \end{aligned} \tag{3.6}$$

The decoding algorithm is given in Algorithm 3.

3.3.3 Analysis

Proposition 3.3.5. *The code defined in Construction 3.3.4 can correct from $\rho\delta_{\text{out}} mn$ adversarial deletions, using Algorithm 3, in $O(n^3)$ time.*

Proof. Let $x \in \mathbb{F}_q^k$ be a message and denote by $\sigma := ((\sigma_1, S_1 \cdot \sigma_1), \ldots, (\sigma_n, S_n \cdot \sigma_n))$, the outer codeword corresponding to x, i.e., $\sigma = \mathcal{C}_{\text{out}}(x)$. We first note that if x is the zero message then since the adversary is allowed to perform only deletions to $\text{ENC}(x)$, the input to the algorithm is a single run of zeros. Therefore the algorithm will output the zero message as required. Thus, from now on, we assume that x is not the zero message.

We will upper bound the edit distance between σ and L that is obtained after performing step 3 of Algorithm 3. If it holds that $\text{ED}(\sigma, L) \leq \delta_{\text{out}} n$, then the decoding succeeds since our outer code, $\mathcal{C}_{\text{out}}$, can correct from $\delta_{\text{out}} n$ insdel errors.

Algorithm 3: Decoding algorithm for Construction 3.3.4

input : Binary string y which is the output of the deletion adversary on $\mathrm{ENC}(x)$.
output: A message $x' \in \mathbb{F}_q^k$.

[0] Set L to be the empty list.
[1] **if** *y is a single run of zeros* **then**
  output $\bar{0} \in \mathbb{F}_q^k$ and return
end
[2] Every run of zeros of length at least $4\delta_{\mathrm{in}}m$ is identified as an outer buffer.
Let $r_1, \ldots, r_t$ be the strings between the outer buffers.
[3] **for** *every r_j* **do**
  Every run of zeros of length at least $\delta_{\mathrm{in}}m$ and less than $4\delta_{\mathrm{in}}m$ is identified as an inner buffer.
  if *exactly 1 inner buffer was identified* **then**
    Denote by c_j the string before the inner buffer and by c'_j the string after the inner buffer. In particular $r_j = c_j \circ (\text{identified inner buffer}) \circ c'_j$.[4]
    if $m - 2\delta_{\mathrm{in}}m < |c_j| \leq m$ *and* $m - 2\delta_{\mathrm{in}}m < |c'_j| \leq m$ **then**
      $a = \mathrm{Dec}(c_j)$ and $b = \mathrm{Dec}(c'_j)$.
      if *a is not $\perp$ and b is not $\perp$* **then**
        Add to L the tuple (a, b).
      end
    end
  end
end
[4] Decode L using the algorithm of $\mathcal{C}_{\mathrm{out}}$ given in Theorem 3.1.9.

Before we continue with the proof, we note that the outer codeword, σ, might have zero symbols (which are of the form $(0,0)$). Note that such a symbol is encoded, by the inner code, to a long run of zeros, which is then interpreted by our algorithm as an outer buffer. As can be seen in the proof of Theorem 3.1.9 (see Remark 3.2.4), we only care about nonzero symbols. Namely, if we denote by σ^0 the string obtained from σ by deleting all the zero symbols, then as long as $\mathrm{ED}(\sigma^0, L) < \delta_{\mathrm{out}}n$, the decoding algorithm succeeds. Thus, we do not need to insert these zero symbols to L.

Assume then that $x \neq 0$. In Step 2 the decoding algorithm identifies outer buffers. We say that the algorithm identified correctly the ith outer buffer if in Step 2 it identified an outer buffer that contains one of the surviving symbols of the ith outer buffer of $\mathrm{ENC}(x)$, and that contains no symbol of any other outer buffer of $\mathrm{ENC}(x)$. We call such an identified outer buffer a *genuine outer buffer*. Observe, that if the ith outer symbol is $\sigma_i = (0,0)$, then the algorithm may identify the entire run between the $(i-1)$th and the ith outer buffers as a single outer buffer. In this case, too we say that this is a genuine outer buffer. The reason for that will become clear during the analysis. In a nutshell, the reason for not treating it as an erroneous buffer follows from the discussion above that shows that our algorithm ignores the zero outer symbol (see Remark 3.2.4). In all other cases, we say that the decoder identified a *fake outer buffer*. We call an outer buffer that was not identified as an outer buffer (because the adversary deleted many 0s from it) a *corrupted outer buffer*.

After identifying the outer buffers in Step 2, we get t strings $r_1, \ldots, r_t$. We distinguish between three different types of r_js, depending on the outer buffers that the algorithm identified:

Type-1 r_j – there exists an $i \in [n-1]$ such that the algorithm identified the $(i-1)$th genuine outer buffer before r_j and the ith genuine outer buffer after r_j. If $j = 1$ (t) then we require the algorithm to identify only the right (left) outer buffer.

Type-2 r_j – if the buffers surrounding r_j are genuine outer buffers that do not correspond to consecutive outer buffers in $\mathrm{ENC}(x)$.

Type-3 r_j – if at least one of the buffers surrounding r_j is a fake outer buffer.

We first study how the adversary can create a Type-1 r_j that is not decoded correctly in Step 3. In what follows, for a substring s of $\mathrm{ENC}(x)$, we denote with $\tilde{s}$ the remaining subsequence of s after the deletions performed by the adversary.

Type-1 r_j: In this case, r_j is the form

$$r_j = \widetilde{\mathcal{C}_{\mathrm{in}}(\sigma_i)} \circ \widetilde{0^{(\mathrm{in})}} \circ \widetilde{\mathcal{C}_{\mathrm{in}}(S_i\sigma_i)} \ ,$$

and we assume that the (original) ith buffer preceding r_j and the $(i+1)$th buffer following r_j were identified by the algorithm.

We say that r_j is a *surviving outer symbol* if a single inner buffer was identified inside r_j (thus $r_j = c_j \circ$ (identified inner buffer) $\circ\, c'_j$), and the decoding algorithm of the inner code returns σ_i and $S_i \cdot \sigma_i$ when given c_j and c'_j, respectively. If in Step 3 the algorithm adds to L the tuple $(a,b) \neq (\sigma_i, S_i\sigma_i)$, when going over r_j, then we call r_j a *fake outer symbol*. Note that the algorithm can also ignore r_j in Step 3 and in this case, we call r_j an *ignored outer symbol*. For example, if r_j contains several runs of zeros of length $\geq \delta_{\mathrm{in}} m$, then several inner buffers are identified inside r_j, in which case the algorithm will not add anything to L.

Our objective is to show that the adversary has to perform at least $\rho m + 1$ deletions to $\mathcal{C}_{\mathrm{in}}(\sigma_i) \circ 0^{(\mathrm{in})} \circ \mathcal{C}_{\mathrm{in}}(S_i\sigma_i)$ in order to create a Type-1 r_j that gets ignored by our algorithm and at least $\delta_{\mathrm{in}} m + \rho m$ deletions in order to create a Type-1 r_j that is a fake outer symbol. We say that the algorithm *identified correctly the inner buffer* if exactly one inner buffer was identified inside r_j and at least one of the bits in the identified inner buffer belongs to the original inner buffer.

The following claim shows that if the inner buffer was identified correctly and the adversary performed at most ρm deletions to each of the inner codewords, then the decoding algorithm of the inner code successfully decodes c_j and c'_j.

Claim 3.3.6. *Assume that the algorithm identified correctly the inner buffer inside r_j (thus, $r_j = c_j \circ$ (identified inner buffer)$\circ c'_j$). Then, as long as the adversary performed $\leq \rho m$ deletions to $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ ($\mathcal{C}_{\mathrm{in}}(S_i \cdot \sigma_i)$), the decoding algorithm of the inner code, outputs correctly σ_i ($S_i \cdot \sigma_i$) when given c_j (c'_j).*

Proof. First, note that it may be the case that a string of 0s of an inner codeword (i.e., of $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ or of $\mathcal{C}_{\mathrm{in}}(S_i\sigma_i)$) are identified as a part of the inner or outer buffers. This is because our algorithm identifies buffers whenever it encounters a long enough run of zeros. Therefore, if $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ starts with a run of zeros, then this run is identified by our algorithm as part of the first outer buffer. The same phenomenon happens if $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ ends with a run of zeros, only this time the zeroes are identified as part of the inner buffer. Denote by $\mathcal{C}_{\mathrm{in}}(\sigma_i)'$ the substring of $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ obtained by deleting the first and last run of zeros. By Proposition 3.3.1(2), $\mathcal{C}_{\mathrm{in}}(\sigma_i)'$ is of length $\geq (1 - 2(\delta_{\mathrm{in}} - \rho))m$.

Note that the adversary has the option to delete 1s from the beginning (or end) of $\mathcal{C}_{\mathrm{in}}(\sigma_i)'$ and as a result, further 0s will be identified as part of a buffer by the algorithm. For example, assume 11010010 to be the first eight bits of $\mathcal{C}_{\mathrm{in}}(\sigma_i)'$ and further assume that the adversary deletes the first three 1s from the left. In this case, we have 11010010, where the red 1s were deleted by the adversary and the blue 0s are interpreted, by the algorithm, as part of the left outer buffer. Denote by b_1 the number of consecutive 1s deleted from the beginning of $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ and by e_1 the number of consecutive 1s deleted from the end of $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ where $b_1 + e_1 \leq \rho m$, then, the number of zeros merged to the buffer is at most

$$\lceil (b_1+1)/(\rho m+1) \rceil \, (\delta_{\mathrm{in}} m - \rho m) + \lceil (e_1+1)/(\rho m+1) \rceil \, (\delta_{\mathrm{in}} m - \rho m) = 2(\delta_{\mathrm{in}} m - \rho m) \ .$$

Denote the resulting string (after removing the first and last runs of 0s that were created by the adversary after deleting $b_1 + e_1$ 1s) by $\mathcal{C}_{\mathrm{in}}(\sigma_i)''$ and note that $\mathcal{C}_{\mathrm{in}}(\sigma_i)''$ is a substring of $\mathcal{C}_{\mathrm{in}}(\sigma_i)$ of length $\geq (1 - 2\delta_{\mathrm{in}} + \rho)m$. Now, the adversary can perform another $\rho m - (b_1 + e_1)$ deletions to the rest of the bits of $\mathcal{C}_{\mathrm{in}}(\sigma_i)''$. In total, $\mathrm{LCS}(c_j, \mathcal{C}_{\mathrm{in}}(\sigma_i)'') \geq |\mathcal{C}_{\mathrm{in}}(\sigma_i)''| - \rho m$. Proposition 3.3.1(1) guarantees that we decode this corrupted codeword successfully. □

Thus, in order for the adversary to make the algorithm ignore r_j or interpret it as a fake outer symbol, it must either

Case 1: delete enough 0s so that no inner buffer is identified, or

Case 2: delete many 1s so that more than one inner buffer is identified, or

Case 3: delete bits so that only a single inner buffer is identified, but that the decoding algorithm fails.

We study each of these cases separately.

Analysis of Case 1: In this case, the adversary must have deleted at least $\delta_{\text{in}}m + 1$ bits from the original inner buffer. In this case, r_j is ignored by the algorithm.

Analysis of Case 2: In this case, the algorithm identifies (at least) two inner buffers in r_j, and as a result, ignores it. Proposition 3.3.1(2) implies that the adversary must delete at least $\rho m + 1$ many 1s from an inner codeword in order to create a second long run of 0s that is interpreted as an inner buffer.

Analysis of Case 3: We now assume that the algorithm identified a single inner buffer. If this inner buffer does not contain any bit of the original inner buffer, then, by the two previous cases, the adversary must have deleted at least $\delta_{\text{in}}m + 1$ many 0 from the original inner buffer and additionally at least $\rho m + 1$ many 1s from an inner codeword. In total, at least $\delta_{\text{in}}m + \rho m + 2$ many bits were deleted. In this case, either r_j is ignored by the algorithm, or it becomes a fake outer symbol.

If the algorithm correctly identified the inner buffer, then Claim 3.3.6 implies that, for the algorithm to fail to decode, the adversary must have deleted more than ρm bits inside $\mathcal{C}_{\text{in}}(\sigma_i)$ or $\mathcal{C}_{\text{in}}(S_i \cdot \sigma_i)$. In particular, the adversary must perform more than ρm deletions for the decoding to fail. Notice that in this case, the decoding algorithm of the inner code will output $\perp$ and will not return a fake outer symbol.

To conclude, if the adversary wishes to create a Type-1 r_j that is an ignored outer symbol, it needs to perform at least $\rho m + 1$ deletions. In order to create a Type-1 r_j that is a fake outer symbol, the adversary needs to delete at least $\delta_{\text{in}}m + \rho m + 2$ many bits.

Observe that an ignored outer symbol increases $\text{ED}(\sigma^0, L)$ by 1 since the corresponding outer symbol, $(\sigma_i, S_i \cdot \sigma_i)$, was not added to L. A Type-1 r_j that is a fake outer symbol increases $\text{ED}(\sigma^0, L)$ by 2 since instead of the original outer symbol, a fake outer symbol is added to L. Thus, the number of deletions that the adversary has to "pay" in order to increase $\text{ED}(\sigma^0, L)$ by 1, in the case of Type-1 r_j, is at least

$$\min\left\{\rho m + 1, \frac{\delta_{\text{in}}m + \rho m + 2}{2}\right\} = \rho m + 1 \ ,$$

where the equality follows as $\delta_{\text{in}} > 2.5\rho$. Thus, in the case of Type-1 r_j, it is more "economical" for the adversary to make the algorithm ignore it rather than make it a fake outer symbol.

Type-2 r_j: In this case, we assume that r_j is such that the outer buffer identified before r_j and the outer buffer identified after r_j are genuine but not consecutive (and there is no fake outer buffer in between). Assume that the outer buffer before r_j corresponds to the i_1th outer buffer in $\text{ENC}(x)$ and that the outer buffer after r_j corresponds to the i_2th original outer buffer. In particular, the $i_2 - i_1 - 1$ outer buffers between the i_1th and i_2th were corrupted by the adversary.

We now consider how many deletions the adversary had to perform in order for the algorithm to return a fake outer symbol. Note that the substring of the original codeword that starts at the first 1 following the i_1th outer buffer and ends at the last 1 preceding the i_2th outer buffer is of length at least

$$\begin{aligned} 2((1 - \delta_{\text{in}} + \rho) + 2\delta_{\text{in}} + 1)m + 5\delta_{\text{in}}m + (i_2 - i_1 - 2)(2 + 7\delta_{\text{in}})m \\ = (i_2 - i_1)(2 + 7\delta_{\text{in}})m - (7\delta_{\text{in}} - 2\rho)m \ . \end{aligned}$$

Observe that for the algorithm to not ignore r_j we must have that $|r_j| < 2m + 4\delta_{\text{in}}m$. It follows that for the algorithm not to ignore r_j, the adversary must have deleted at least

$$(i_2 - i_1)(2 + 7\delta_{\text{in}})m - (7\delta_{\text{in}} - 2\rho)m - (2 + 4\delta_{\text{in}})m = (i_2 - i_1)(2 + 7\delta_{\text{in}})m - (2 + 11\delta_{\text{in}} - 2\rho)m$$

many bits. Creating such a fake outer symbol increases $\text{ED}(\sigma^0, L)$ by $i_2 - i_1 + 1$ as it corresponds to deleting the outer symbols in locations $i_1, \ldots, i_2 - 1$ and an insertion of the fake outer symbol.

If the adversary only corrupted the outer buffers between the i_1th and the i_2th outer buffers, without creating a fake outer symbol, then it must have deleted at least $(i_2 - i_1 - 1)(\delta_{\text{in}}m + 1)$ many 0s. Indeed, to corrupt a single outer buffer (at least) $\delta_{\text{in}}m+1$ many 0s have to be deleted. Such a behaviour by the adversary increases $\text{ED}(\sigma^0, L)$ by $i_2 - i_1$ as it is equivalent to deleting the outer symbols in locations $i_1, \ldots, i_2 - 1$.

Thus, the number of deletions that the adversary has to "pay" in order to increase $\text{ED}(\sigma^0, L)$ by 1, in the case of Type-2 r_j, is at least

$$\begin{aligned}&\min\left\{\frac{(i_2 - i_1)(2 + 7\delta_{\text{in}})m - (2 + 11\delta_{\text{in}} - 2\rho)m}{(i_2 - i_1 + 1)}\,,\ \frac{(i_2 - i_1 - 1)(\delta_{\text{in}}m + 1)}{(i_2 - i_1)}\right\}\\ &= \frac{(i_2 - i_1 - 1)(\delta_{\text{in}}m + 1)}{(i_2 - i_1)}\,.\end{aligned}$$

Observe that $\frac{(i_2 - i_1 - 1)(\delta_{\text{in}}m + 1)}{(i_2 - i_1)} > \rho m + 1$ and hence the adversary has to make more deletions in the case of Type-2 r_j than in the case of Type-1 r_j in order to increase $\text{ED}(\sigma^0, L)$ by 1.

Type-3 r_j: Let us assume without loss of generality that the outer buffer to the left of r_j is a fake outer buffer.

To create a fake outer buffer, the adversary has to create a run of 0s of length $\geq 4\delta_{\text{in}}m$ such that all the bits in this run do not belong to any outer buffer in $\text{ENC}(x)$ (or that belong to two different outer buffers in $\text{ENC}(x)$. We treat this case later). The adversary faces two options; it can either merge many 0s to an inner buffer or create a run of 0s of length $\geq 4\delta_{\text{in}}m$ inside an inner codeword. By Proposition 3.3.1(2), the second case requires at least $4\rho m + 4$ many deletions. In the first case, the adversary needs to merge $\geq 2\delta_{\text{in}}m$ many 0s to an inner buffer. We claim that in this case, it must delete more than $\rho m + 1$ many 1s from the inner codewords. Indeed, by Proposition 3.3.1(2), any $\delta_{\text{in}}m$ coordinates of an inner codeword contain at least $\rho m + 1$ many 1s. As at least $\delta_{\text{in}}m$ 0s must come from either the inner codeword to the left of the inner buffer or from the one to the right of the inner buffer, the claim follows.

Now that we know the "cost" of creating a fake outer buffer, we shall analyze several cases. Denote with i_1 the index such that the last bit of the fake outer buffer came from the encoding of $(\sigma_{i_1}, S_{i_1} \cdot \sigma_{i_1})$.

1. The outer buffer to the right of r_j is a genuine outer buffer corresponding to the (i_1)th outer buffer in $\text{ENC}(x)$: In this case it is not hard to verify that $|c_j| + |c'_j| < 2m - 4\delta_{\text{in}}m$ and r_j gets ignored. This increases $\text{ED}(\sigma^0, L)$ by 1, and, by the analysis above, the adversary had to make at least $\rho m + 1$ many deletions.

2. The outer buffer to the right of r_j is a genuine outer buffer, but not the i_1th one: Let us assume that the genuine outer buffer to the right of r_j is the i_2th outer buffer (observe that we must have $i_2 > i_1$). We now consider two subcases:

   (a) The algorithm ignored r_j: As all the outer buffers between the i_1th and the i_2th were corrupted, the adversary must have deleted at least $(i_2 - i_1)(\delta_{\text{in}}m + 1)$ many 0s. This increases $\text{ED}(\sigma^0, L)$ by at most $i_2 - i_1 + 1$ as it causes the deletion of all symbols in locations $i_1 + 1, \ldots, i_2$, and potentially also the i_1th symbol. Thus, the average cost of increasing the edit distance by 1 in this case is at least $\frac{(i_2 - i_1)(\delta_{\text{in}}m + 1)}{i_2 - i_1 + 1} > (\delta_{\text{in}}m + 1)/2 > \rho m + 1$.

   (b) The algorithm decoded r_j to a fake outer symbol: Similarly to the analysis of Type-2 r_j, we see that in this case, as the algorithm has to identify a single inner buffer inside r_j, and the length of r_j is $|r_j| \leq 2m + 4\delta_{\text{in}}m$, the adversary must have deleted at least

$$(i_2 - i_1)(7\delta_{\text{in}} + 2)m - (2 + 4\delta_{\text{in}})m = (i_2 - i_1 - 1)(7\delta_{\text{in}} + 2)m + 3\delta_{\text{in}}m$$

many bits. This increases $\mathrm{ED}(\sigma^0, L)$ by at most $i_2 - i_1 + 2$ since (as in the previous case) this caused at most $i_2 - i_1 + 1$ many deletions and a single insertion. Thus, the average cost of increasing the edit distance by 1 in this case is at least $\frac{(i_2-i_1-1)(7\delta_{\mathrm{in}}+2)m+3\delta_{\mathrm{in}}m}{i_2-i_1+2} \geq \delta_{\mathrm{in}}m > \rho m + 1$.

3. The outer buffer to the right of r_j is also a fake outer buffer: Let us assume that the outer buffer to the right of r_j was created inside the encoding of the i_2th outer symbol. We analyze two cases:

   (a) $i_2 = i_1$: In this case, it is not hard to see that r_j is too short and hence gets ignored by the algorithm. This increases $\mathrm{ED}(\sigma^0, L)$ by 1. Note that by the analysis above, the adversary had to make at least $(4\rho m + 4) + (\rho m + 1) = 5\rho m + 5$ many deletions.

   (b) $i_2 > i_1$: Similar calculations as in the case of Type-2 r_j show that in this case, the adversary has to make more than $\rho m + 1$ many deletions in order to increase $\mathrm{ED}(\sigma^0, L)$ by 1. Indeed, we recall that at least $\rho m+1$ deletions occurred to create the outer buffer to the left of r_j (we do not charge anything for the right one in order to avoid double-counting). Now let us assume that the first bit in the second fake outer buffer came from $(\sigma_{i_2}, S_{i_2} \cdot \sigma_{i_2})$. It follows that in order to corrupt all the outer buffers between the i_1th and the $(i_2 - 1)$th outer buffers, the adversary must delete at least $(i_2 - i_1)\delta_{\mathrm{in}}m$ many bits. In this case, if r_j is not interpreted as a fake outer symbol, then $\mathrm{ED}(\sigma^0, L)$ grew by at most $i_2 - i_1 + 1$. If r_j was decoded to a fake outer symbol, then $\mathrm{ED}(\sigma^0, L)$ grew by at most $i_2 - i_1 + 2$. Thus, the average cost of increasing the edit distance by 1 in this case is at least $\frac{(i_2-i_1)\delta_{\mathrm{in}}m+\rho m+1}{i_2-i_1+2} > \frac{(2.5(i_2-i_1)+1)\rho m}{i_2-i_1+2} \geq \frac{3.5}{3}\rho m > \rho m + 1$ (the first inequality follows since $\delta_{\mathrm{in}} > 2.5\rho$).

Finally, we note that if the fake outer buffer before r_j contains bits from two different original outer buffers, the i_1th and the i_2th, then at least $2(i_2 - i_1)\frac{(1-\delta+\rho)\rho}{\delta}m$ many 1s had to be deleted. Such an operation increases the edit distance by at most $i_2 - i_1$. In addition, we have to repeat the analysis above and take into consideration the cost of creating the buffer to the right of r_j, and the additional effect of r_j on the edit distance (i.e., whether r_j was ignored or decoded as a fake outer symbol, etc.). It is clear that in this case, the cost of increasing the edit distance by 1 is much larger than $\rho m+1$.

In conclusion, in all cases, in order to increase $\mathrm{ED}(\sigma^0, L)$ by 1, the adversary has to make at least $\rho m + 1$ many deletions. Since the adversary can make at most $\delta_{\mathrm{out}}\rho mn$ deletions, it follows that $\mathrm{ED}(\sigma^0, L) < \delta_{\mathrm{out}}n$. Hence, by the assumption on the outer code, Step 4 of Algorithm 3 returns the correct message. This completes the correctness part of Proposition 3.3.5. All that is left is to analyze the running time complexity of the algorithm.

Running time: The claim about the running time follows by first noting that Step 2, in which we identify the outer buffers, runs in linear time. Secondly, for each r_j, the run time of Step 3 is determined by the cost of the brute force decoding algorithm. This algorithm runs in exponential time in m, where $m = \mathrm{poly}(1/\varepsilon_{\mathrm{out}})$. Hence, Step 3 runs in time $n \cdot \mathrm{poly}(1/\varepsilon_{\mathrm{out}})$. Finally, according to Theorem 3.1.9, the decoding algorithm of the outer code runs in time $O(n^3)$. In conclusion, the running time of the decoding algorithm is $O(n^3)$. This concludes the proof of Proposition 3.3.5.

3.3.4 Proof of Theorem 3.1.10

Proposition 3.3.5 implies that the code constructed in Construction 3.3.4 can decode from $\rho\delta_{\mathrm{out}}mn$ many deletions. By Equation (3.6), its rate is $\frac{\mathcal{R}_{\mathrm{in}}\mathcal{R}_{\mathrm{out}}}{2+7\delta_{\mathrm{in}}}$.

Recall that $\varepsilon_{\mathrm{out}} < \delta_{\mathrm{out}}/1400$, $\delta_{\mathrm{in}} = 1/6$, $\rho = 1/17$, $\mathcal{R}_{\mathrm{out}} = (1-\delta_{\mathrm{out}})/4 - \varepsilon_{\mathrm{out}}$ and $\mathcal{R}_{\mathrm{in}} = \delta_{\mathrm{in}}/16$. It follows

that the rate of our code is

$$\begin{aligned}\mathcal{R} &\geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}}{2+7\delta_{\text{in}}} \\ &= \frac{1}{304}\cdot\left(\frac{1-\delta_{\text{out}}}{4}-\varepsilon_{\text{out}}\right) \\ &\geq \frac{1}{304}\cdot\left(\frac{1-1.0029\delta_{\text{out}}}{4}\right) ,\end{aligned}$$

and it can correct from more than $\delta = \delta_{\text{out}}\rho/(2+7\delta_{\text{in}}) > \delta_{\text{out}}/53.84$ fraction of adversarial deletions. Thus, we conclude that the final rate-error trade-off is

$$\mathcal{R} \geq \frac{1-54\cdot\delta}{1216} .$$

□

3.4 Open questions

In this chapter, we studied linear codes that can handle insdel errors. Naturally, our main goal was to construct codes that get close (or match) the half-Singleton bound. Over constant size and even binary alphabets, we constructed efficient linear codes that have significantly higher rates compared to previous constructions. However, we were not able to achieve the half-Singleton bound, and this is left as an open question for future work.

Chapter 4

Reed–Solomon codes correcting insertions and deletions

4.1 Introduction

In this chapter, we first prove that there are RS-codes that achieve the half-Singleton bound. In other words, (some) RS-codes are optimal also against insdel errors. We also give explicit constructions (i.e. sets of evaluation points) that define RS-codes that achieve this bound. As the field sizes that we get grow very fast, our construction runs in polynomial time only for very small values of k. We also explicitly construct 2-dimensional RS-codes over a field of size smaller than all previous known constructions. Unfortunately, we do not have efficient decoding algorithms, and we leave this as an open problem for future research.

4.1.1 Previous results

The performance of RS-codes against insdel errors was studied much earlier than the work of Cheng et al. [CGHL21]. To the best of our knowledge, Safavi-Naini and Wang [SNW02] were the first to study the performance of RS-codes against insdel errors. They gave an algebraic condition that is sufficient for an RS-code to decode from insdel errors, yet they did not provide any construction. In fact, in our work, we consider an almost identical algebraic condition, and by simply using the Schwartz-Zippel-DeMillo-Lipton lemma [Sch80, Zip79, DL78], we prove that there are RS-codes that meet this condition and, in addition, achieve the half-Singleton bound. In particular, RS-codes are optimal for insdel errors (see discussion in Section 4.2). Wang, McAven, and Safavi-Naini [WMSN04] constructed a $[5, 2]$ RS-code capable of correcting a single deletion. Then, in [TSN07], Tonien and Safavi-Naini constructed an $[n, k]$ generalized-RS-codes capable of correcting from $\log_{k+1} n - 1$ insdel errors. Similar to our results, they did not provide an efficient decoding algorithm.

In another line of work Duc, Liu, Tjuawinata, and Xing [DLTX19], Liu and Tjuawinata [LT21], Chen and Zhang [CZ21], and Liu and Xing [LX21] studied the specific case of 2-dimensional RS-codes.

In [DLTX19, LT21], the authors presented constructions of $[n, 2]$ RS-codes that for every $\varepsilon > 0$ can correct from $(1 - \varepsilon) \cdot n$ insdel errors, for codes of length $n = \text{poly}(1/\varepsilon)$ over fields of size $\Omega(\exp((\log n)^{1/\varepsilon}))$ and $\Omega(\exp(n^{1/\varepsilon}))$, respectively. In [DLTX19, CZ21], the authors present constructions of two-dimensional RS-codes that can correct from $n - 3$ insdel errors where the field size is exponential in n. After a draft of this work appeared online, Liu and Xing [LX21] constructed, using a different approach than ours, a two dimensional RS-codes that can correct from $n - 3$ insdel errors, over a field of size $O(n^5)$. Specifically, they proved the following.

Theorem 4.1.1. *[LX21, Theorem 4.8] Let $n \geq 4$. If $q > \frac{n(n-1)^2(n-2)^2}{4}$, then there is an $[n, 2]_q$ RS-code, constructable in polynomial time, that can decode from $n - 3$ insdel errors.*

4.1.2 Our results

First, we prove that there are RS-codes that achieve the half-Singleton bound. Namely, they are optimal linear codes for insdel errors.

Theorem 4.1.2. *Let k and n be positive integers such that $2k-1 \leq n$. For $q = O(n^{4k-2})$ there exists an $[n,k]_q$ RS-code defined by n distinct evaluation points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$, that can decode from $n-2k+1$ adversarial insdel errors.*

The proof of Theorem 4.1.2 uses a standard union bound combined with the Schwartz-Zippel-DeMillo-Lipton lemma. By using the Lovász-local-lemma, we were able to improve the dependence on the field size. Specifically, we show the following.

Theorem 4.1.3. *For integers n and $k < n/9$, there exists an $[n,k]_q$ RS-code, where $q = O\left(k^4 \cdot \left(\frac{4en}{4k-3}\right)^{4k-3}\right)$ is a prime power, that can decode from $n-2k+1$ adversarial insdel errors.*

Observe that these codes achieves the half Singleton bound: their rate is $\mathcal{R} = k/n = (1-\delta)/2 + o(1)$ and their minimal relative distance is $\delta = (n-2k+1)/n$.

Theorem 4.1.2 and Theorem 4.1.3 are existential results and do not give explicit constructions. Using ideas from number theory and algebra, we construct RS-codes that can decode from $n-2k+1$ adversarial insdel errors, in particular, they achieve the half-Singleton bound. Specifically,

Theorem 4.1.4. *Let k and n be positive integers, where $2k-1 \leq n$. There is a deterministic construction of an $[n,k]_q$ RS-code that can correct from $n-2k+1$ insdel errors where $q = O\left(n^{k^2 \cdot ((2k)!)^2}\right)$. The construction runs in polynomial time for $k = O(\log(n)/\log\log(n))$.*

We note that for $k = \omega(\log(n)/\log\log(n))$ the field size is $\exp(n^{\omega(1)})$ and in particular, there is no efficient way to represent arbitrary elements of $\mathbb{F}_q$ in this case.

As discussed before, special attention was given in the literature to the case of RS-codes of dimension 2. By using Sidon spaces from [RRT17], we explicitly construct a family of $[n,2]_q$ RS-codes that can decode from $n-3$ insdel errors for $q = O(n^4)$. The field size of this explicit construction matches the field size of the non-explicit randomized construction given in Theorem 4.1.3.

Theorem 4.1.5. *For any $n \geq 4$, there exists an explicit $[n,2]_q$ RS-code that can correct from $n-3$ insdel errors, where $q = O(n^4)$.*

We also prove a (very) weak lower bound on the field size.

Proposition 4.1.6. *Any $[n,k]_q$ RS-code that can correct from $n-2k+1$ worst case insdel errors must satisfy*

$$q \geq \frac{1}{2} \cdot \left(\frac{n}{(2k-1)(k-1)}\right)^{\frac{2k-1}{k-1}}.$$

While for large values of k, this bound is meaningless, it implies that when $k=2$, the field size must be $\Omega(n^3)$. Thus, the construction given in Theorem 4.1.5 is nearly optimal. The gap between the field size in our construction and the one implied by the lower bound raises an interesting question: what is the minimal field size q for which an optimal $[n,2]_q$ RS-code exists?

4.1.3 Proof idea

As explained above, we first provide an algebraic condition that is sufficient for n evaluation points to define an RS-code that can decode from insdel errors. This condition requires that a certain set of $n^{O(k)}$ matrices, determined by the evaluation points, must all have full rank. Then, a simple application of the Schwartz-Zippel-DeMillo-Lipton lemma [Sch80, Zip79, DL78] implies the existence of good evaluation points over fields of size $n^{O(k)}$. To obtain a deterministic construction, we show that by going to much larger field

size, one can find evaluation points satisfying the full-rank condition. While the field size needs to be of size roughly $\Omega(n^{k^k})$, we note that, for not too large values of k, it is of exponential size, and in this case, our construction runs in polynomial time. A key ingredient in the analysis of this construction is our use of the 'abc theorem' for polynomials over finite fields [VW03]. To get the improved bound of Theorem 4.1.3 we study the dependencies between the different matrices and then rely on the Lovász-local-lemma to obtain an improved upper bound on the field size.

For the case of $k = 2$, we use a different idea that gives a better field size than the one implied by the probabilistic argument above. We do so by noting that in this case the full-rank condition can be expressed as the requirement that no two different triples of evaluation points (x_1, x_2, x_3) and (y_1, y_2, y_3) satisfy

$$\frac{y_1 - y_2}{x_1 - x_2} = \frac{y_2 - y_3}{x_2 - x_3} .$$

This condition is reminiscent of the condition behind the construction of Sidon spaces of [RRT17], and indeed, we build on their construction of Sidon spaces to define good evaluation points in a field of size $O(n^4)$.

4.1.4 Organization

The chapter is organized as follows. In Section 4.2, we prove Theorem 4.1.2 and Theorem 4.1.3. In Section 4.3, we prove Theorem 4.1.4. Finally, in Section 4.4, we prove Theorem 4.1.5 and Proposition 4.1.6. Section 4.5 is devoted to conclusion and open questions.

4.2 Optimal Reed-Solomon codes exist

In this section, we prove that there are RS-codes that achieve the half-Singleton bound. The proofs will follow by standard analysis of the LCS between any two distinct codewords.

We begin by reformulating the condition on the maximum length of an LCS as an algebraic condition (invertibility of certain matrices). Then we show that an RS-code that satisfies this condition would have the maximum possible edit distance and hence would be able to decode from the maximum number of insdel errors. We remark that a similar approach already appeared in [SNW02, Section 2.2] and we shall repeat some of the details here.

4.2.1 An algebraic condition

The following proposition is the main result of this section as it provides a sufficient condition for an RS-code to recover from the maximum number of insdel errors. We first make the following definitions. We say that a vector of indices $I \in [n]^s$ is an *increasing* vector if its coordinates are monotonically increasing, i.e., for any $1 \leq i < j \leq s$, $I_i < I_j$, where I_i is the ith coordinate of I. For a codeword c of length n and an increasing vector I, let c_I denote the restriction of c to the coordinates with indices in I, i.e., $c_I = (c_{I_1}, \ldots, c_{I_s})$. For two vectors $I, J \in [n]^{2k-1}$ with distinct coordinates we define the following variant of a vandermonde matrix of order $(2k-1) \times (2k-1)$ in the formal variables $\mathbf{X} = (X_1, \ldots, X_n)$:

$$V_{I,J}(\mathbf{X}) = \begin{pmatrix} 1 & X_{I_1} & \ldots & X_{I_1}^{k-1} & X_{J_1} & \ldots & X_{J_1}^{k-1} \\ 1 & X_{I_2} & \ldots & X_{I_2}^{k-1} & X_{J_2} & \ldots & X_{J_2}^{k-1} \\ \vdots & \vdots & \ldots & \vdots & \vdots & \ldots & \vdots \\ 1 & X_{I_{2k-1}} & \ldots & X_{I_{2k-1}}^{k-1} & X_{J_{2k-1}} & \ldots & X_{J_{2k-1}}^{k-1} \end{pmatrix} . \tag{4.1}$$

Proposition 4.2.1. *Consider the $[n,k]_q$ RS-code defined by an evaluation vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. If for every two increasing vectors $I, J \in [n]^{2k-1}$ that agree on at most $k-1$ coordinates, it holds that $\det(V_{I,J}(\boldsymbol{\alpha})) \neq 0$, then the code can correct any $n-2k+1$ insdel errors. Moreover, if the code can correct any $n-2k+1$ insdel errors, then the only possible vectors in Kernel$(V_{I,J}(\boldsymbol{\alpha}))$ are of the form $(0, f_1, \ldots, f_{k-1}, -f_1, \ldots, -f_{k-1})$.*

Proof. Assume that the claim does not hold; therefore, there exist two distinct codewords $c \neq c'$ whose LCS is at least $2k-1$, i.e., $c_I = c'_J$ for two increasing vectors $I, J \in [n]^{2k-1}$. Assume further that c and c' are the encodings of the polynomials $f = \sum_i f_i x^i$ and $g = \sum_i g_i x^i$ of degree less than k, respectively. If $I_\ell = J_\ell$ for at least k coordinates, then for every such ℓ

$$f(\alpha_{I_\ell}) = c_{I_\ell} = c'_{J_\ell} = g(\alpha_{I_\ell}) \ .$$

Hence $f \equiv g$, in contradiction to the fact that $c \neq c'$. Thus, we can assume that I, J agree on at most $k-1$ coordinates. In this case, $V_{I,J}(\boldsymbol{\alpha})$ is singular, since the vector $(f_0 - g_0, f_1, \ldots, f_{k-1}, -g_1, \ldots, -g_{k-1})^t$ is in its right kernel, which contradicts our assumption. From Lemma 3.1.3 it follows that the code can correct $n - 2k + 1$ insdel errors.

To prove the moreover part note that the argument above implies that if the code can correct any $n-2k+1$ insdel errors and $f \neq g$ then the vector $(f_0 - g_0, f_1, \ldots, f_{k-1}, -g_1, \ldots, -g_{k-1})$ is not in the kernel. □

In [SNW02] Safavi-Naini and Wang identified (almost) the same condition (see Remark 4.2.2 below) and used it in their construction of traitor tracing schemes. Interestingly, the later work of [TSN07], which gave a construction of RS-codes capable of decoding from $\log_k(n+1) - 1$ insdel errors, did not use this condition. In particular, as far as we know, prior to this work the condition in Proposition 4.2.1 was not used in order to show the existence of optimal RS-codes against insdel errors.

The following remark explains the difference between Proposition 4.2.1 and the condition in [SNW02].

Remark 4.2.2. *The main difference between the condition presented in [SNW02] and ours, is that they considered a $2k \times 2k$ matrix and a generalized RS-code. Given evaluation points $(\alpha_1, \ldots, \alpha_n)$ and a vector with nonzero coordinates $(v_1, \ldots, v_n) \in \mathbb{F}_q^n$, the generalized $[n,k]_q$ RS-code is defined as the set of all vectors $(v_1 \cdot f(\alpha_1), \ldots, v_n \cdot f(\alpha_n))$, such that $\deg(f) < k$. The matrix studied in [SNW02] is:*

$$V^v_{I,J}(\mathbf{X}) = \begin{pmatrix} v_{I_1} & v_{I_1} \cdot X_{I_1} & \ldots & v_{I_1} \cdot X_{I_1}^{k-1} & v_{J_1} & v_{J_1} \cdot X_{J_1} & \ldots & v_{J_1} \cdot X_{J_1}^{k-1} \\ v_{I_2} & v_{I_2} \cdot X_{I_2} & \ldots & v_{I_2} \cdot X_{I_2}^{k-1} & v_{J_2} & v_{J_2} \cdot X_{J_2} & \ldots & v_{J_2} \cdot X_{J_2}^{k-1} \\ \vdots & \vdots & \ldots & \ldots & \vdots & \vdots & \ldots & \vdots \\ v_{I_{2k}} & v_{I_{2k}} \cdot X_{I_{2k}} & \ldots & v_{I_{2k}} \cdot X_{I_{2k}}^{k-1} & v_{J_{2k}} & v_{J_{2k}} \cdot X_{J_{2k}} & \ldots & v_{J_{2k}} \cdot X_{J_{2k}}^{k-1} \end{pmatrix} . \tag{4.2}$$

In our matrix, we save a coordinate (which leads to optimal codes) as we do not have two columns for the free terms of f and g. In contrast, the matrix (4.2) has a column for the free term of f (the first) and a column for the free term of g (the column $(v_{J_1}, \ldots, v_{J_{2k}})$). This also leads to the requirement that I and J are of length $2k$ (they can still agree on at most $k-1$ indices).

4.2.2 Existence using Schwarz-Zippel-DeMillo-Lipton lemma

In this section, we show that over large enough fields, there exist RS-codes that attain the half-Singleton bound. Specifically, we show that there exist RS-codes that can decode from a δ fraction of insdel errors and have rate $\mathcal{R} = (1-\delta)/2 + o(1)$. For convenience, we repeat the statement of Theorem 4.1.2.

Theorem 4.1.2. *Let k and n be positive integers such that $2k - 1 \leq n$. For $q = O(n^{4k-2})$ there exists an $[n,k]_q$ RS-code defined by n distinct evaluation points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$, that can decode from $n - 2k + 1$ adversarial insdel errors.*

For a vector I and an element a, we write $a \in I$ if a appears as one of the coordinates of I; otherwise, we write $a \notin I$.

Lemma 4.2.3. *Let $s \geq 2$ be an integer and $I, J \in [n]^s$ two increasing vectors that do not agree on* any *coordinate, i.e., $I_i \neq J_i$ for all $1 \leq i \leq s$. Then, there are two distinct indices $i \neq j \in [s]$ such that $I_i \notin J$ and $J_j \notin I$.*

Proof. W.l.o.g. assume that $I_1 < J_1$. Since J is an increasing vector, $I_1 \notin J$. In addition, some coordinate among $\{J_1, \ldots, J_s\}$ does not appear in $\{I_2, \ldots, I_s\}$, and any such coordinate is clearly different from I_1. □

Proposition 4.2.4. *Let $I, J \in [n]^{2k-1}$ be two increasing vectors that agree on at most $k-1$ coordinates. Then, in the expansion of* $\det(V_{I,J}(\mathbf{X}))$ *as a sum over permutations, there is a monomial that is obtained at exactly one of the $(2k-1)!$ different permutations. In particular, its coefficient is ± 1, depending on the sign of its corresponding permutation. Consequently,* $\det(V_{I,J}(\mathbf{X})) \neq 0$.

Proof. We use induction on k to prove the claim. For $k = 1$, $V_{I,J}(\mathbf{X}) = 1$ and the result follows. For the induction step, assume that the claim holds for every $1 \leq \ell \leq k-1$. Consider two coordinates i, j, determined as follows. If I and J agree on some coordinate, say j, then we set i to be such that $I_i \notin J$. If they do not agree on any coordinate, then we let i, j be the two coordinates guaranteed by Lemma 4.2.3.

Next, in the determinant expansion of $V_{I,J}$ as a sum of $(2k-1)!$ monomials, collect all the monomials that are divisible by $X_{I_i}^{k-1} X_{J_j}^{k-1}$, and write them together as

$$X_{I_i}^{k-1} X_{J_j}^{k-1} f(\mathbf{X}),$$

for some polynomial f in the variables $(X_\ell : \ell \in (I \setminus \{I_i\}) \cup (J \setminus \{J_j\}))$. Note that the choice of i and j guarantees that such monomials exist. Observe that any monomial in the determinant expansion of $V_{I,J}$ that is divisible by $X_{I_i}^{k-1} X_{J_j}^{k-1}$ must be obtained by picking the (i,k) and the $(j, 2k-1)$ entries in the matrix (4.1). Hence, f equals the determinant of the submatrix $V'_{I,J}$ obtained by removing rows i, j and columns $k, 2k-1$ from $V_{I,J}$. Note that $V'_{I,J}$ is a matrix satisfying the conditions of the claim: it is a $(2k-3) \times (2k-3)$ matrix defined by two increasing vectors of length $2k-3$ that agree on at most $k-2$ coordinates. Indeed, i and j were chosen so that by removing them we remove one agreement if such existed.

Hence, by the induction hypothesis, $\det(V'_{I,J})$ has a monomial m (with a ± 1 coefficient) that is uniquely obtained among the $(2k-3)!$ different monomials. Therefore, $X_{I_i}^{k-1} X_{J_j}^{k-1} m$ is a monomial of $X_{I_i}^{k-1} X_{J_j}^{k-1} f$ with a ± 1 coefficient. Since there is no other way to obtain this monomial in the determinant of $V_{I,J}$, this monomial is uniquely obtained in $\det(V_{I,J})$, and the result follows. □

We now state the Schwarz-Zippel-DeMillo-Lipton lemma and then prove Theorem 4.1.2.

Lemma 4.2.5 (Schwarz-Zippel-DeMillo-Lipton lemma)**.** *[Sch80, Zip79, DL78] Let $F \in \mathbb{F}_q[X_1, \ldots, X_n]$ be a nonzero polynomial of total degree d and let $S \subseteq \mathbb{F}_q$. If we pick $\alpha_1, \ldots, \alpha_n$ independently and uniformly from S, then*

$$\Pr\left[F(\alpha_1, \ldots, \alpha_n) = 0\right] \leq \frac{d}{|S|}\,.$$

Proof of Theorem 4.1.2. Define

$$F(\mathbf{X}) = \prod_{i<j} (X_i - X_j) \prod_{I,J} \det(V_{I,J}(\mathbf{X})),$$

where the second product runs over all possible pairs of increasing vectors that agree on at most $k-1$ coordinates. Clearly, by Proposition 4.2.4, $F(\mathbf{X})$ is a nonzero polynomial in the ring $\mathbb{Z}[\mathbf{X}]$. Next, we make two observations regarding the polynomial F. First, since there are $\binom{n}{2k-1}$ increasing vectors, and the degree of each $\det(V_{I,J}(\mathbf{X}))$ is at most $k(k-1)$, it follows that

$$\deg(F) \leq n^2 + \binom{n}{2k-1}^2 \cdot k(k-1) < n^{4k-2}\,.$$

Second, as each $\det(V_{I,J}(\mathbf{X}))$ is a nonzero polynomial with nonzero coefficients bounded in absolute values by $(2k-1)!$, the absolute value of any nonzero coefficients of F is at most

$$((2k-1)!)^{\binom{n}{2k-1}^2} \leq ((2k-1)!)^{\frac{n^{4k-2}}{((2k-1)!)^2}} < e^{n^{4k-2}}.$$

We claim that there is a prime q in the range $[n^{4k-2}, 2n^{4k-2}]$ that does not divide at least one of the nonzero coefficients of the polynomial F. Indeed, consider a nonzero coefficient of F, and assume towards

a contradiction that it is divisible by all such primes. Then, by the growth rate of the primorial function, the absolute value of the coefficient is $\Omega(e^{n^{4k-2}(1+o(1))})$, in contradiction. Now, it is easy to verify that F is also a nonzero polynomial in $\mathbb{F}_q[\mathbf{X}]$, since the monomial whose nonzero coefficient is not divisible by q does not vanish. Therefore, since q is strictly greater than the degree of F, we can apply Lemma 4.2.5 with $S = \mathbb{F}_q$ and get that there exists an assignment $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$ to $\mathbf{X}$ for which $F(\boldsymbol{\alpha}) \neq 0 \mod q$. This assignment clearly corresponds to n *distinct* evaluation points, which by Proposition 4.2.1, define an $[n,k]_q$ RS-code that can correct any $n-2k+1$ worst-case insdel errors, as claimed. □

We remark again that Theorem 4.1.2 merely shows the existence of $[n,k]_q$ RS-codes that can decode from the maximum number of insdel errors over fields of size $q = O\left(n^{4k-2}\right)$. Further, the above argument is a standard union bound over all variable assignments that make the matrix defined in (4.1) singular. This by no means implies that such a large finite field is necessary. For example, a similar union-bound argument for showing the existence of MDS codes would require an exponentially large field for codes with a constant rate. In contrast, it is well-known that MDS codes over fields of linear size exist (e.g., RS-codes). It would be interesting to construct codes with the same or even better parameters than the ones given in Theorem 4.1.2. In Section 4.2.3 we achieve such improvement using the Lovász-local-lemma, but we still don't know whether this is the best possible result. We thus leave it as an open question for further research.

We do not have explicit constructions that match the bound of Theorem 4.1.2, and obtaining such constructions is another interesting open problem. Nonetheless, in Section 4.3, we provide a deterministic construction of an RS-code for any admissible n, k, at the expense of a larger field size than the one guaranteed by Theorem 4.1.2.

4.2.3 Existence using the Lovász-local-lemma

In this section we prove Theorem 4.1.3, which improves on the required field size from Theorem 4.1.2. More precisely, we show that for large enough n and any $k < n/9$, there is an $[n,k]_q$ RS-code that can correct from any $n-2k+1$ insdel errors with $q = O(k^4 \cdot (4en/(4k-3))^{4k-3})$. For constant dimensional codes, the field size is of order $O(n^{4k-3})$, and in particular, for $k=2$ the field size is of order $O(n^5)$. For convenience, we restate the theorem.

Theorem 4.1.3. *For integers n and $k < n/9$, there exists an $[n,k]_q$ RS-code, where $q = O\left(k^4 \cdot \left(\frac{4en}{4k-3}\right)^{4k-3}\right)$ is a prime power, that can decode from $n-2k+1$ adversarial insdel errors.*

As in the previous section, we choose $\boldsymbol{\alpha}$ uniformly at random from $\mathbb{F}_q^n$. The difference is that instead of using the Schwarz-Zippel-DeMillo-Lipton lemma we apply the Lovász-local-lemma [EL73] to show that with positive probability the condition in Proposition 4.2.1 holds. Finally, we make a small adjustment in order to guarantee that the evaluation points α_i are all distinct. The full details of the proof are given next, but first we recall the Lovász-local-lemma.

Theorem 4.2.6. *[EL73] Let $E_1, \ldots, E_N$ be events such that $\Pr[E_i] \leq p$ for all $i \in [N]$, every event E_i depends on at most d other events and $4pd \leq 1$. Then,*

$$\Pr\left[\bigwedge_{i=1}^{N} \overline{E_i}\right] > 0 \,.$$

We begin with some definitions and notations. Let $\mathcal{S}$ be the set containing all pairs (I,J) where $I, J \in [n]^{2k-1}$ are two increasing vectors that agree on at most $k-1$ coordinates. Denote by $E_{I,J}$ the (bad) event that $\det(V_{I,J}(\boldsymbol{\alpha})) = 0$, where $V_{I,J}(\boldsymbol{\alpha})$ is the matrix defined in (4.1). We shall also use the notation $I \cup J$ to refer to the set of indices that appear in I or in J.

Let $\boldsymbol{\alpha}$ be a vector chosen uniformly at random from $\mathbb{F}_q^n$. The following claim shows that an event $E_{I,J}$ is independent from a set of $\{E_{I',J'}\}$ as long as $I \cup J$ is disjoint from all $I' \cup J'$ for which $E_{I',J'} \in \{E_{I',J'}\}$.

Claim 4.2.7. *Fix $(I,J) \in \mathcal{S}$ and define*

$$T_{I,J} := \{(I',J') \in \mathcal{S} \mid |(I \cup J) \cap (I' \cup J')| = 0\} .$$

Then, the event $E_{I,J}$ is mutually independent from the set of events $S_{I,J} = \{E_{I',J'} \mid (I',J') \in T_{I,J}\}$

Proof. Since $I \cup J$ is disjoint from all $I' \cup J', (I',J') \in T_{I,J}$, the matrix $V_{I,J}(\mathbf{X})$ does not share any variables with any of the matrices $V_{I',J'}(\mathbf{X})$ for $(I',J') \in T_{I,J}$. The result follows since $\boldsymbol{\alpha}$ chosen uniformly at random from $\mathbb{F}_q^n$. □

Claim 4.2.8. *An event $E_{I,J}$ is mutually independent of all but at most $O\left(k^2 \cdot \left(\frac{4en}{4k-3}\right)^{4k-3}\right)$ of the events $E_{I',J'}$.*

Proof. By Claim 4.2.7, the number of events $E_{I',J'}$ that are not independent of $E_{I,J}$ is at most

$$\binom{|I \cup J|}{1} \cdot \sum_{|I' \cup J'|=2k}^{4k-2} \left(\binom{n}{|I' \cup J'| - 1} \cdot \binom{|I' \cup J'|}{2k-1}^2 \right) \tag{4.3}$$

where $\binom{|I \cup J|}{1}$ stands for choosing a shared index, $\binom{n}{|I' \cup J'|-1}$ is an upper bound on the number of ways to choose the rest of the indices in $I' \cup J'$, and $\binom{|I' \cup J'|}{2k-1}^2$ is an upper bound on the number of ways to choose I' and J' out of $I' \cup J'$. Note that the size of $I' \cup J'$ (and $I \cup J$) must be at least $2k$ and at most $4k-2$ and thus we have the sum in (4.3). For $4k-3 < n/2$, (4.3) is at most

$$\begin{aligned}
&\binom{4k-2}{1}\binom{4k-2}{2k-1}^2 \sum_{|I' \cup J'|=2k}^{4k-2} \binom{n}{|I' \cup J'| - 1} \\
&\leq (4k-2) \cdot 2^{2(4k-2)} \cdot (2k-1) \cdot \binom{n}{4k-3} \\
&\leq 2(2k-1)^2 \cdot 2^{2(4k-2)} \cdot \left(\frac{en}{4k-3}\right)^{4k-3} \\
&= O\left(k^2 \cdot \left(\frac{4en}{4k-3}\right)^{4k-3}\right) .
\end{aligned}$$

□

We are now ready to prove Theorem 4.1.3.

Proof. Recall that $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is chosen uniformly at random from $\mathbb{F}_q^n$. Proposition 4.2.4 guarantees that $\det(V_{I,J}(\mathbf{X}))$ is a nonzero polynomial of degree less than k^2, when $I, J \in [n]^{2k-1}$ are increasing vectors that agree on at most $k-1$ coordinates. Thus, by the Schwarz-Zippel-DeMillo-Lipton lemma $\Pr[E_{I,J}] < k^2/q$. Consequently, if we choose $q = \Omega\left(k^4 \left(\frac{4en}{4k-3}\right)^{4k-3}\right)$, we get from Theorem 4.2.6 and Claim 4.2.8 that with positive probability none of the events $E_{I,J}$ hold. In other words, there exists an evaluation vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ for which all the matrices $V_{I,J}(\boldsymbol{\alpha})$ are non-singular.

Consider such a vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. If $k = 2$, we claim that each evaluation point appears at most twice in the vector $\boldsymbol{\alpha}$. Indeed, assume otherwise, say $\alpha_i = \alpha_j = \alpha_s$ for indices $i < j < s$. Then, it can be easily verified that $V_{I,J}(\boldsymbol{\alpha})$ is singular for $I = (i,j,s)$ and any increasing vector J, in contradiction. Thus, by puncturing the vector $\boldsymbol{\alpha}$ at each repeated evaluation point we get the desired RS-code, of length at least $n' \geq n/2$, over a field of size $q = O((n')^5)$, which can correct any $n'-3$ insdel errors.

If $k > 2$, we claim that all the evaluation points are distinct. Indeed, assume $\alpha_i = \alpha_j$ for some $i < j$. Then, there are two increasing vectors $I, J \in [n]^{2k-1}$ that $I_{\ell_1} = J_{\ell_1} = \alpha_i$ and $I_{\ell_2} = J_{\ell_2} = \alpha_j$ for some $\ell_1 < \ell_2$

(recall that since $k > 2$ then I and J can agree on two coordinates). We get that $\det(V_{I,J}(\boldsymbol{\alpha})) = 0$ as the rows ℓ_1 and ℓ_2 of the matrix are equal, and the theorem follows.

□

In Section 4.4, we provide an explicit construction of an $[n, 2]_q$ RS-code that can correct from $n-3$ insdel errors and $q = O(n^4)$. This result outperforms the non-explicit result given in Theorem 4.1.3.

4.3 Deterministic construction for any k

In this section, we give our main construction of an $[n, k]$ RS-code that can correct any $n - 2k + 1$ insdel errors. Specifically, we prove Theorem 4.1.4 which is restated for convenience

Theorem 4.1.4. *Let k and n be positive integers, where $2k-1 \leq n$. There is a deterministic construction of an $[n, k]_q$ RS-code that can correct from $n-2k+1$ insdel errors where $q = O\left(n^{k^2 \cdot ((2k)!)^2}\right)$. The construction runs in polynomial time for $k = O(\log(n)/\log\log(n))$.*

Remark 4.3.1. *The downside of this construction is the field size $q = n^{k^{O(k)}}$, which renders it to run in polynomial time only for $k = O(\log(n)/\log\log(n))$. For larger values of k, the representation of each field element requires a super polynomial number of bits.*

The Mason-Stothers theorem [Mas84, Sto81] is a result about polynomials that satisfy a non-trivial linear dependence, which is analogous to the well-known *abc conjecture* in number theory [Mas85, Oes88]. Our main tool is one of the many extensions of the Mason-Stothers theorem. For stating the theorem we need the following notation: For a polynomial $Y(x) \in \mathbb{F}_q[x]$ over a field with characteristic $\text{char}(\mathbb{F}_q) = p \neq 0$, denote by $\nu(Y(x))$ the number of distinct roots of $Y(x)$ with multiplicity not divisible by p.

Theorem 4.3.2 ("Moreover part" of Proposition 5.2 in [VW03])**.** *Let $m \geq 2$ and $Y_0(x) = Y_1(x) + \ldots + Y_m(x)$ with $Y_j(x) \in \mathbb{F}_p[x]$. Suppose that $\gcd(Y_0(x), \ldots, Y_m(x)) = 1$, and that $Y_1(x), \ldots, Y_m(x)$ are linearly independent over $\mathbb{F}_p(x^p)$.*[1] *Then,*

$$\deg(Y_0(x)) \leq -\binom{m}{2} + (m-1)\sum_{j=0}^{m} \nu(Y_j(x)) \,.$$

Construction 4.3.3. *Let k be a positive integer and set $\ell = ((2k)!)^2$. Fix a finite field $\mathbb{F}_p$ for a prime $p > k^2 \cdot \ell$ and let n be an integer such that $2k-1 < n \leq p$. Let $\mathbb{F}_q$ be a field extension of $\mathbb{F}_p$ of degree $k^2 \cdot \ell$ and let $\gamma \in \mathbb{F}_q$ be such that $\mathbb{F}_q = \mathbb{F}_p(\gamma)$. Hence, each element of $\mathbb{F}_q$ can be represented as a polynomial in γ, of degree less than $k^2\ell$, over $\mathbb{F}_p$. Define the $[n, k]_q$ RS-code by setting $\alpha_i := (\gamma - i)^\ell$ for $1 \leq i \leq n$.*

Proposition 4.3.4. *The $[n, k]_q$ RS-code defined in Construction 4.3.3 can correct any $n-2k+1$ worst case insdel errors.*

Proof. Let $I, J \in [n]^{2k-1}$ be two increasing vectors that agree on at most $k-1$ coordinates. By Proposition 4.2.1 it is enough to show that $V_{I,J}(\boldsymbol{\alpha})$ is non-singular, for every such I, J. By the Leibniz formula, $\det(V_{I,J}(\boldsymbol{\alpha}))$ is a sum of $(2k-1)!$ terms corresponding to the different permutations. Denote these terms as $P_i(\gamma)$ for $i = 0, \ldots, (2k-1)!-1$. Each of the terms is a product of the sign of the corresponding permutation with some $2k-1$ elements of the form $(\gamma - s)^{\ell \cdot j}$, for some $s \in I \cup J$ and $0 \leq j \leq k-1$. Assume towards a contradiction that $\det(V_{I,J}(\boldsymbol{\alpha})) = 0$ in $\mathbb{F}_q$, i.e.,

$$\det(V_{I,J}(\boldsymbol{\alpha})) = P_0(\gamma) + \ldots + P_{(2k-1)!-1}(\gamma) = 0 \,, \tag{4.4}$$

in $\mathbb{F}_q$. By viewing every term in (4.4) as a univariate polynomial in γ over $\mathbb{F}_p$, one can verify that, for any j, $\deg(P_j) = \ell \cdot k(k-1) < k^2\ell$. As $\mathbb{F}_q$ is an extension of $\mathbb{F}_p$ of degree $k^2\ell$, it follows that (4.4) holds also in

[1] $\mathbb{F}_p(x^p)$ is the field of rational functions in x^p. Namely, its elements are $f(x^p)/g(x^p)$ where $f(x), g(x) \in \mathbb{F}_p[x]$ and $g(x) \not\equiv 0$.

$\mathbb{F}_p[\gamma]$, the ring of polynomials in the variable γ over $\mathbb{F}_p$. By Proposition 4.2.4 the determinant of the variable matrix (4.1) has a monomial that is uniquely obtained and therefore has a ± 1 coefficient. Assume, without loss of generality, that P_0 is the image of this monomial under the mapping defined by the assignment $X_i \mapsto (\gamma - i)^\ell$. Note that since this mapping is injective on the set of monomials, no other monomial is mapped to a scalar multiple of P_0. In other words, P_0 and P_i are linearly independent for any $i \geq 1$. Assume further that (without loss of generality) $P_1, \ldots, P_m$ is a minimal subset among $\{P_i\}_{i \geq 1}$ that spans P_0 over $\mathbb{F}_p$. The existence of such a set follows from (4.4). Hence, we can write

$$P_0 = \sum_{i=1}^{m} a_i P_i, \text{ where } a_i \in \mathbb{F}_p \backslash \{0\}. \tag{4.5}$$

Clearly, by minimality, $P_1, \ldots, P_m$ are linearly independent over $\mathbb{F}_p$. Further, $m \geq 2$, since otherwise there would be an $i > 0$ such that P_i is a multiple of P_0.

Since the P_i's are of degree $\ell k(k-1)$, and P_0 was obtained from a unique monomial in the expansion of the determinant, it follows that the greatest common divisor $Q := \gcd(P_0, \ldots, P_m)$ has degree at most $\ell(k(k-1)-1)$. By dividing (4.5) by Q we have

$$\overline{P_0} = \sum_{i=1}^{m} a_i \overline{P_i}, \tag{4.6}$$

where $\overline{P_i} = P_i/Q$. We will need the following claim, whose proof is deferred to the end of this section.

Claim 4.3.5. *The polynomials $\overline{P_1}, \ldots, \overline{P_m}$ are linearly independent over $\mathbb{F}_p(\gamma^p)$.*

The contradiction will follow by invoking Theorem 4.3.2. Towards this end note that (i) By Claim 4.3.5 the polynomials $\overline{P_1}, \ldots, \overline{P_m}$ are linearly independent over $\mathbb{F}_p(\gamma^p)$ (ii) $\gcd(\overline{P_0}, \ldots, \overline{P_m}) = 1$, and (iii) $\nu(\overline{P_j}) \leq 2k-2$, as P_j is the multiplication of $2k-2$ non-constant polynomials, each having a single root. Thus, by (4.6) and Theorem 4.3.2

$$\begin{aligned} \ell \leq \deg(P_0) - \deg(Q) = \deg(\overline{P_0}) &\leq -\binom{m}{2} + (m-1) \cdot \sum_{i=1}^{m} \nu(\overline{P_j}) \\ &< m^2(2k-2) \\ &\leq ((2k-1)!)^2 \cdot (2k-2)\ , \end{aligned}$$

which is a contradiction by the choice of ℓ. This completes the proof. □

It remains to prove Claim 4.3.5.

Proof of Claim 4.3.5. Assume towards a contradiction that there exist $\lambda_1, \ldots, \lambda_m \in \mathbb{F}_p(\gamma^p)$ not all zero, such that

$$\sum_{j=1}^{m} \lambda_j(\gamma^p)\overline{P_j}(\gamma) = 0\ . \tag{4.7}$$

By clearing the denominators of the λ_j's and any common factor they might have, we can assume that the λ_j's are polynomials in the variable γ^p with no common factors. Since $\deg(\overline{P_j}) \leq \deg(P_j) < p$, we get by reducing (4.7) modulo γ^p that

$$\sum_{j=1}^{m} \lambda_j(0)\overline{P_j}(\gamma) = 0\ .$$

Note that $\lambda_j(0) \neq 0$ for some j, since otherwise γ^p would be a common factor of the λ_i's. Hence, $\overline{P_1}, \ldots, \overline{P_m}$ are linearly dependent over $\mathbb{F}_p$, which contradicts the choice of $P_1, \ldots, P_m$ as a minimal spanning set. □

By setting $n = p$ in Construction 4.3.3 it follows that the field size of Construction 4.3.3 is roughly $n^{k^{O(k)}}$ which is much worse than the field size guaranteed by the existential results in Theorem 4.1.2 and Theorem 4.1.3. Note, however, that the construction runs in polynomial time for RS-codes with dimension $O(\log(n)/\log\log(n))$. The proof of Theorem 4.1.4 immediately follows.

4.4 Explicit construction for $k = 2$ with quartic field size

In this section we prove Theorem 4.1.5, which is restated for convenience.

Theorem 4.1.5. *For any $n \geq 4$, there exists an explicit $[n, 2]_q$ RS-code that can correct from $n - 3$ insdel errors, where $q = O(n^4)$.*

The proof of Theorem 4.1.5 requires the notion of Sidon spaces, which were introduced in a work of Bachoc, Serra, and Zémor [BSZ17] in the study of an analogue of Vosper's theorem for finite fields. Later, Roth, Raviv, and Tamo gave an explicit construction of Sidon spaces and used it to provide a construction of cyclic subspace codes [RRT17]. Their construction was also recently used in [RLT21] to construct a public-key cryptosystem. We also rely on their construction. We believe that Sidon spaces in general, and specifically the construction of [RRT17], might find more applications in coding theory and cryptography in the future. We begin with a formal definition of a Sidon space.

Definition 4.4.1. *An $\mathbb{F}_q$ linear subspace $S \subseteq \mathbb{F}_{q^n}$ is called a* Sidon space *if for any nonzero elements $a, b, c, d \in S$ such that $ab = cd$, it holds that*

$$\{a\mathbb{F}_q, b\mathbb{F}_q\} = \{c\mathbb{F}_q, d\mathbb{F}_q\},$$

where $x\mathbb{F}_q = \{x \cdot \alpha : \alpha \in \mathbb{F}_q\}$.

A Sidon space S has the following interesting property, from which it draws its name: Given the product $a \cdot b$ of two nonzero elements $a, b \in S$, one can uniquely factor it to its two factors a, b from S, up to a multiplication by a scalar from the base field. Clearly, this is the best one can hope for, since for any nonzero $\alpha \in \mathbb{F}_q$ the product of the elements $\alpha \cdot a, b/\alpha \in S$ also equals $a \cdot b$. A Sidon space can be viewed as a multiplicative analogue to the well-known notion of *Sidon sets*, which is a common object of study in combinatorics, see e.g. [ET41].

We proceed to present the construction of a Sidon space given in [RRT17].

Theorem 4.4.2 (Construction 15, Theorem 16 in [RRT17])**.** *Let $q \geq 3$ be a prime power, $m \in \mathbb{N}$, and $n = 2m$. Then, there exists an explicit $\gamma \in \mathbb{F}_{q^n}$ such that $S = \{u + u^q \cdot \gamma \mid u \in \mathbb{F}_{q^m}\}$ is an m-dimensional Sidon space over $\mathbb{F}_q$.*

Another component in our construction is the "long" ternary code with minimum distance of at least 5, given in [GS86]. We note that we could also use the codes given in [DD08].

Theorem 4.4.3. *[GS86] For every $m \geq 1$, there exists an explicit $[(3^m + 1)/2, (3^m + 1)/2 - 2m]_3$ linear code with minimum distance at least 5.*

We next combine the above two algebraic objects and construct an RS-code with the desired properties.

Construction 4.4.4. *For $q = 3$ and $m \in \mathbb{N}$. Let $S \subset \mathbb{F}_{3^{4m}}$ be a $2m$-dimensional Sidon space over $\mathbb{F}_3$ as guaranteed by Theorem 4.4.2. Let $s_1, \ldots, s_{2m}$ be a basis of S. Let $H = (h_{i,j})$ be a $(2m) \times ((3^m + 1)/2)$ parity check matrix of the code given in Theorem 4.4.3. Our $[n, 2]_{3^{4m}}$ RS-code of length $n = (3^m + 1)/2$ is defined by the evaluation points*

$$\alpha_j = \sum_{i=1}^{2m} s_i h_{i,j} \text{ for } 1 \leq j \leq (3^m + 1)/2 \, .$$

In other words, we can think of our evaluation points as the n coordinates of the vector $\boldsymbol{\alpha} = (s_1, \ldots, s_{2m}) \cdot H$.

The following property of the evaluation points α_j follows easily from Theorem 4.4.3.

Lemma 4.4.5. *Any four distinct α_j's are linearly independent over $\mathbb{F}_3$.*

Proof. Consider four distinct α_j's, say $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, and assume towards a contradiction that there exist $\beta_1, \ldots, \beta_4 \in \mathbb{F}_3$ not all zero, such that $\sum_{j=1}^{4} \beta_j \alpha_j = 0$. Then

$$0 = \sum_{j=1}^{4} \beta_j \alpha_j = \sum_{j=1}^{4} \beta_j \sum_{i=1}^{2m} s_i h_{i,j} = \sum_{i=1}^{2m} s_j \sum_{j=1}^{4} \beta_j h_{i,j} \ .$$

Since the s_j's are linearly independent over $\mathbb{F}_3$ it follows that $\sum_{j=1}^{4} \beta_j h_{i,j} = 0$ for every $i = 1, \ldots, 2m$. Hence, the four columns h_1, h_2, h_3, h_4 of H are linearly dependent over $\mathbb{F}_3$, which contradicts the fact that the minimum distance of the code checked by H is at least 5. □

We proceed to prove that the constructed RS-code can decode from the maximum number of insdel errors.

Theorem 4.4.6. *The $[n, 2]_{3^{4m}}$ RS-code given in Construction 4.4.4 can correct any $n-3$ worst case insdel errors.*

Proof. Assume towards a contradiction that this is not the case. Proposition 4.2.1 implies that there must exist two triplets of distinct evaluation points $(x_1, x_2, x_3), (y_1, y_2, y_3)$, that agree on at most one coordinate, such that

$$\left| \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix} \right| = 0 \ .$$

Equivalently, $(y_1 - y_2)(x_2 - x_3) = (y_2 - y_3)(x_1 - x_2)$. Since the x_i's are distinct elements of the Sidon space S, $x_2 - x_3$ and $x_1 - x_2$ are *nonzero* elements in S. Similarly, $y_1 - y_2$ and $y_2 - y_3$ are nonzero elements in S. By definition of Sidon spaces, there exists a nonzero $\lambda \in \mathbb{F}_3$ such that

$$\lambda(y_1 - y_2) = y_2 - y_3 \text{ or } \lambda(y_1 - y_2) = x_1 - x_2,$$

which contradicts Lemma 4.4.5. Indeed, each of the equations implies a nontrivial linear dependence over $\mathbb{F}_3$ between at least three and at most four evaluation points (here we used the facts that the elements of each triple are distinct and that the two triples agree on at most one coordinate). □

We conclude this section with a proof of Theorem 4.1.5.

Proof of Theorem 4.1.5. By Theorem 4.4.6, the code given in Construction 4.4.4 is an RS-code of length $n = (3^m + 1)/2$, which is defined over the field $\mathbb{F}_{3^{4m}}$, which is of order $O(n^4)$, as claimed. □

4.4.1 A lower bound on the field size

In Section 4.2.2 we proved the existence of optimal $[n, k]_q$ RS-codes for worst-case insdel errors, over fields of size $q = n^{O(k)}$. This section complements this result by providing a lower bound on the field size for such codes. Specifically, we ask how large must q be in any $[n, k]_q$ RS-code that can correct from $n - 2k + 1$ worst-case insdel errors. We prove the following.

Proposition 4.1.6. *Any $[n, k]_q$ RS-code that can correct from $n - 2k + 1$ worst case insdel errors must satisfy*

$$q \geq \frac{1}{2} \cdot \left(\frac{n}{(2k-1)(k-1)} \right)^{\frac{2k-1}{k-1}} \ .$$

Proof. Consider an $[n,k]_q$ RS-code, defined by evaluation points $\alpha_1,\ldots,\alpha_n$, that can correct any $n-2k+1$ insdel errors. For a *non-constant* polynomial f of degree less than k let $\mathcal{V}_f$ be the set of all subsequences of the codeword corresponding to f, of length $2k-1$:

$$\mathcal{V}_f = \{(f(\alpha_{i_1}),\ldots,f(\alpha_{i_{2k-1}})) : 1 \le i_1 < \ldots < i_{2k-1} \le n\} \subseteq \mathbb{F}_q^{2k-1}.$$

Now, since the code can decode from any $n-2k+1$ insdel errors, by Lemma 3.1.3, the longest common subsequence between any two distinct codewords is of length at most $2k-2$. Therefore, the sets $\mathcal{V}_f$ and $\mathcal{V}_g$ for two distinct polynomials f,g, are disjoint. Therefore,[2]

$$\sum_{1\le \deg(f)<k} |\mathcal{V}_f| \le q^{2k-1}. \tag{4.8}$$

Next, we provide a lower bound on the size of $\mathcal{V}_f$. For any non-constant polynomial f, of degree less than k, and any $a \in \mathbb{F}_q$ there are at most $k-1$ indices i such that $f(\alpha_i)=a$. Thus, for a fixed vector $(a_1,\ldots,a_{2k-1}) \in \mathcal{V}_f$ there are at most $(k-1)^{2k-1}$ increasing vectors of indices $(i_1,\ldots,i_{2k-1})$ such that

$$(f(\alpha_{i_1}),\ldots,f(\alpha_{i_{2k-1}})) = (a_1,\ldots,a_{2k-1}).$$

Therefore $|\mathcal{V}_f| \ge \binom{n}{2k-1}/(k-1)^{2k-1}$. Combined with (4.8) we have

$$\left(\frac{1}{k-1}\right)^{2k-1} \cdot \binom{n}{2k-1} \cdot \left(q^k - q\right) \le \sum_{1\le \deg(f)<k} |\mathcal{V}_f| \le q^{2k-1}\,,$$

By rearranging and the fact that $q^{2k-1}/(q^k-q) \le 2q^{k-1}$ for $q,k \ge 2$, we have

$$\left(\frac{1}{2}\right)^{\frac{1}{k-1}} \left(\frac{n}{(2k-1)(k-1)}\right)^{\frac{2k-1}{k-1}} \le q\,. \qquad \square$$

As one can easily verify, this bound is rather weak, as it provides an improvement over the trivial lower bound of $q \ge n$ only for the vanishing rate regime of $k = O(n^{1/4})$. For codes of dimension 2, the bound implies $q = \Omega(n^3)$, and it slowly degrades as one increases k. Nevertheless, it is always at least $\Omega(n^2)$ for any constant k. It is interesting to note that by combining Proposition 4.1.6 and Theorem 4.4.6 it follows that an $[n,2]_q$ RS-code that can decode from $n-3$ insdel errors requires that $\Omega(n^3) \le q \le O(n^4)$. Determining the minimum possible value of q for this case is an interesting open problem.

4.5 Summary and open questions

This chapter studies the performance of RS-codes against insdel errors. We showed that there are RS-codes that are optimal against insdel errors, i.e., they achieve the half-Singleton bound. We also construct explicit RS-codes that achieve this bound and, as far as we know, this is the first linear code that is shown to achieve this bound.

As discussed, Construction 4.3.3 is not optimal in terms of the field size. It is a fascinating open question to find an RS-code with an optimal field size. Specifically, the challenge is to construct an RS-code that can correct from any $n-2k+1$ insdel errors, over a field of size $O(n^{O(k)})$ (Theorem 4.1.2 and Theorem 4.1.3 prove the existence of such codes).

The lower bound on the field size proved in Proposition 4.1.6 is far from giving a full picture of the tradeoff between dimension and field size. The natural open question is to significantly improve our lower bound or provide a better upper bound.

Finally, another interesting question is to provide an efficient decoding algorithm for our constructions of RS-codes.

[2] This equation remains true even if we include the constant polynomials.

Chapter 5

Constructions of coding schemes for the binary deletion channel and the Poisson repeated channel

5.1 Introduction

The two most studied channels are the Binary Erasure Channel (BEC_p) where each bit is independently replaced by a question mark with probability p and the Binary Symmetric Channel (BSC_p) where each bit is independently flipped with probability p.

In this part, we consider the BDC with parameter p. This channel models the situation where bits of a transmitted message are deleted (i.e. removed) from the message randomly and independently with probability p. In particular, if a message of length n was transmitted on the BDC_p then the length of the received message is concentrated around $(1-p)\cdot n$. We note that the output of the BDC is very different from that of the BEC or the BSC. For example, if we transmit the message 1110101 over each of the channels and corruptions occurred in locations 2 and 5, then the BEC will return the word 1?10?01, the BSC will return 1010001, and the BDC will return 11001. In particular, while the BEC and the BSC do not affect the length of messages transmitted over them, the BDC does exactly that. Thus, unlike the BEC and BSC, the BDC causes synchronization errors. In fact, one of the main reasons for introducing the BDC was to model synchronization errors in communication.

The motivation to study the BDC is obvious. It is not just a theoretical object as it describes a real-life scenario in which there is a loss of information that was sent on some physical layer as well as synchronization errors. Moreover, the surveys [Mit09, MBT10] indicate that tools that were developed in the context of the BDC are useful in the study of other questions. An example of such a question is the trace reconstruction problem, which has applications in computational biology and DNA storage systems [BLC+16]. The problem that we study in this work is the construction of explicit error correcting codes of (relatively) high rate (we will soon explain this notion) for the BDC_p.

One of the most fundamental questions when studying a channel is to determine its capacity, i.e., the maximum achievable transmission rate over the channel that still allows recovering from the errors introduced by the channel, with high probability. Shannon proved in his seminal work [Sha48] that the capacity of the BSC_p is $1-h(p)$, where $h(\cdot)$ is the binary entropy function (for $0<x<1$, $h(x)=-x\log x-(1-x)\log 1-x$). I.e., there are codes with block lengths going to infinity, whose rates converge to $1-h(p)$, that can recover with high probability from the errors inflicted by the channel. Elias [Eli55], who introduced the BEC_p, proved that its capacity is $1-p$.

What about the capacity of the BDC_p? In spite of many efforts (see [Mit09]), the capacity of the BDC_p is still not known and it is an outstanding open challenge to determine it. Yet, for the extremal cases, the asymptotic behavior is somewhat understood. In the regime where $p\to 0$ the capacity approaches to $1-h(p)$

[KMS10], i.e. it approaches the capacity of the BSC_p. In [MD06], the authors showed that the capacity is at least $(1-p)/9$ for all $p \in (0,1)$. In particular, even if p is extremely close to 1, there are codes of positive rate that allow reliable communication over this channel. Another somewhat surprising aspect of this result is that the asymptotic behavior is only a constant off from the capacity of the related BEC_p. In the BEC_p, we know how to build codes that nearly achieve its capacity of $1-p$ for every p. This is not the case for the BDC_p, where the best explicit construction known for the regime $p \to 1$, prior to this work, has rate of $(1-p)/120$ [GL18].

5.1.1 Lower bounds on the capacity of the BDC

The best known lower bound on the capacity is due to Mitzenmacher and Drinea [MD06, DM07] who showed a lower bound of $0.1185 \cdot (1-d)$ for all d, meaning that there are codes of this rate such that every transmitted codeword is decoded correctly with high probability. This lower bound is the best known lower bound for the high deletion probability regime ($d \geq 0.95$). For smaller values of d, Drinea and Mitzenmacher prove stronger bounds in [DM07, Table 1]. Their proof is constructive, but it does not directly yield an efficient decoding algorithm for the family of codes they construct.

5.1.2 Upper bounds on the capacity of the BDC

Fertonani and Duman [FD10] proved several upper bounds on the capacity of the BDC. They do this by providing the transmitter and the receiver with "hints" about the noise of the channel. Adding these hints only increases the information rate, allowing them to bound the capacity of the BDC from above by bounding the capacity of some auxiliary channels using the Blahut-Arimoto algorithm.

Dalai [Dal11] and Rahmati and Duman [RD14] refine this analysis and prove that for any $d \in (0.65, 1)$, the capacity of the BDC is bounded from above by $\mathcal{C}(\mathrm{BDC}_d) \leq 0.4143(1-d)$. Given unlimited computational resources, these methods will converge to the capacity of the channel, but this convergence is extremely slow.

5.1.3 Efficient constructions for the BDC

There are several constructions of efficiently decodable codes for the BDC. Guruswami and Li [GL18] presents a deterministic and efficient code construction for the BDC_p with rate $(1-p)/120$ for all values of p. This rate is smaller than Mitzenmacher's bound, but it is the first construction with rate that scales proportionally to $(1-p)$ for $p \to 1$. Then, Con and Shpilka [CS22], improved the construction of [GL18] and constructed efficient codes for the BDC_p that have rate $(1-p)/16$. This construction and its analysis are presented in this thesis. After our construction was presented, the works [Rub22, PLW22] presented a method to convert any code for the BDC, not necessarily explicit nor efficient, into an explicit and efficient code for the BDC with a negligible decrease to its rate. Therefore, the codes generated using Mitzenmacher and Drinea's construction can be converted to explicit and efficient codes for the BDC. We also mention that in [TPFV21] the authors prove polarization for the deletion channel and show that the capacity can be achieved using their scheme.

5.1.4 The Poisson repeat channel

Another model that we consider is the Poisson repeat channel (PRC) that was first introduced in the work of Mitzenmacher and Drinea [MD06]. In the PRC with parameter λ, each bit of the message is (randomly and independently) replaced with a discrete number of copies of that bit, distributed according to the Poisson distribution with parameter $0 < \lambda$. In particular, with probability $e^{-\lambda}$ the bit is deleted from the message (i.e. this channel can cause synchronization errors similar to the BDC). This channel can model, for example, messages sent using a keyboard that has a tendency to get stuck so a key cannot be pressed or can get stuck and then its symbol is repeated several times. While the PRC is less motivated by practical applications (we are unaware of any applications of this channel besides in the study of the BDC), it is closely related to the BDC as demonstrated in the work of Mitzenmacher and Drinea [MD06], Drinea and Mitzenmacher [DM07] and Cheraghchi [Che18]. In particular, the lower bound on the capacity (a notion that we explain

shortly) of BDC_p of $(1-p)/9$ [MD06] relies on a reduction from the PRC_λ. We too exploit the connection between the BDC and the PRC, and using our construction for the BDC we obtain explicit constructions of error correcting codes for the PRC.

5.1.5 Our Results

In this work, we present and analyze a polynomial time construction of a family of codes for the BDC_p that achieves rate higher than $(1-p)/16$ for every p and have polynomial time encoding and decoding algorithms. We also show that this construction yields a family of codes for PRC_λ of rate $\mathcal{R} > \lambda/17$ for $\lambda \leq 0.5$. This further emphasizes that these channels have much in common.

Specifically, we improve the construction presented in [GL18] and construct an explicit family of codes for the binary deletion channel with rate at least $(1-p)/16$ for any $p \in (0,1)$ that have polynomial time encoding and decoding algorithms.

Theorem 5.1.1. *Let $p \in (0,1)$. There exist a family of binary error correcting codes $\{C_i\}_{i=1}^{\infty}$ for the BDC_p where the block length of C_i goes to infinity as $i \to \infty$ and*

1. *C_i can be constructed in time polynomial in its block length.*
2. *C_i has rate at least $(1-p)/16$.*
3. *C_i is decodable in quadratic time and encodable in linear time.*

As mentioned earlier, we show that the same construction works for the PRC_λ as well. In particular we prove,

Theorem 5.1.2. *Let $\lambda \leq 0.5$. There exist a family of binary error correcting codes $\{C_i\}_{i=1}^{\infty}$ for PRC_λ where the block length of C_i goes to infinity as $i \to \infty$ and*

1. *C_i can be constructed in time polynomial in its block length.*
2. *C_i has rate $\mathcal{R}_i > \lambda/17$.*
3. *C_i is decodable in quadratic time and encodable in linear time.*

To the best of our knowledge, this is the first explicit construction of an error correcting code for the PRC_λ.

5.1.6 Construction and Proof Overview

Our construction follows the footsteps of the construction of Guruswami and Li [GL18] with some important modifications. We next describe the construction and then its analysis.

Construction: There are several layers to our construction as depicted in Figure 5.1 on page 61. The first two layers come from code concatenation while the third and fourth layers blow-up the code further by repeating symbols and inserting "buffers" between inner codewords. These four layers are similar to those in the construction of [GL18] and the main difference between the constructions is that we use a different inner code and the blow-up in our construction is considerably smaller. We now describe each step in more detail.

Recall that code concatenation is the operation of viewing the message as a shorter message over a larger alphabet, then applying an error correcting code over the large alphabet (the outer code) to the message and, finally, viewing each symbol of the encoded message as a short message over $\{0,1\}$, it is encoded using a binary error correcting code (the inner code).

In our construction, we view the messages as strings of length k over the alphabet $\Sigma = \{0,1\}^{m'}$ (where m' is some constant that we later optimize). As an outer code, we use the code from [HS17], which is an efficient insertion-deletion code with rate close to 1 over Σ. This code returns a word $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \Sigma^n$.

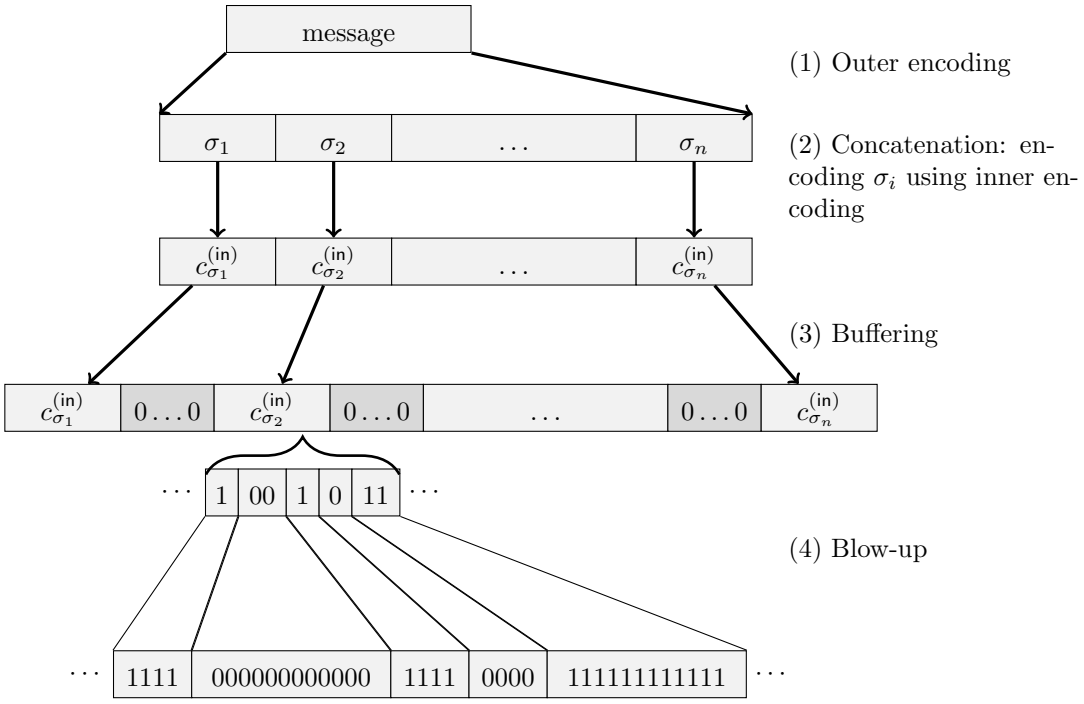

Figure 5.1: The encoding process.

We construct our inner code using a greedy algorithm. First we consider all binary strings of length m which consist of exactly $\beta_1 m$ 1-runs and $0.5(1-\beta_1)m$ 2-runs (i.e. alternating blocks of 0's and 1's where each block length is ≤ 2) where β_1 is a parameter that we will optimize later. Then, we add a codeword to our codebook if it does not contain a subsequence of length $\geq m - \delta m$ that is also a subsequence of any codeword that is already in our codebook. Note that even though the construction time is exponential in m, as $m = O(1)$ in our construction this does not affect the run time by more than a constant factor.

The encoding process is thus as follows (see also Figure 5.1). We first encode the message using the outer code to a codeword of length n over Σ. Then, the concatenation process takes every symbol, σ_i of the outer codeword and maps it to a codeword from the inner code, i.e., a concatenated codeword is of the form $c_1 \circ c_2 \circ \cdots \circ c_n$ where $c_i = \text{ENC}_{\text{in}}(\sigma_i)$, where ENC_{in} is the encoding function of the inner code. This is not the end of the story. In order for the concatenated code to overcome a large amount of deletions caused by the channel we add an additional layer of encoding:

1. We place long buffers of zeros (of length N_B) between inner codewords. This step helps the decoder identify where an inner codeword starts and where it ends.

2. We replace each 1-run with an N_1-run and each 2-run will become an N_2-run (runs of length N_1 and N_2). This helps the decoder identify if the run in the inner codeword was a run of length 1 or 2.

This step is also similar to the construction of [GL18], however, perhaps surprisingly, since we restrict our inner codewords to have a fixed number of 1-runs and 2-runs, this enables us to have N_1 and N_2 considerably smaller than the blow-up parameter used in [GL18]. It is clear that the code construction is efficient as the outer code of [HS17] can be encoded efficiently and the inner code is of constant length and thus can also be encoded efficiently. The last step is clearly efficient (as B, N_1 and N_2 are constants).

Decoding: We now describe our decoding algorithm. First, we identify the buffers in order to divide the string into "decoding windows" that should ideally represent corrupted inner codewords. Second, every decoding window is decoded in the following way: Every run longer than some threshold T is replaced with a 2-run (of the same symbol) and every run of length $\leq T$ is replaced with a 1-run. The third step of the decoding is to use a brute force decoding algorithm on each decoded window to find the closest inner codeword. Since the inner code's block length is constant this step takes constant time for every such window and hence runs in linear time in the length of the word. The last step in the decoding algorithm is to run the decoding algorithm of the outer code as given in [HS17]. This algorithm runs in time quadratic in the outer code's block length. Hence the total run time of our decoder is quadratic in the length of the message.

Analysis: Our analysis classifies errors to three types:

1. Buffer deletions: these are deletions that caused a buffer between inner codewords to completely disappear.
2. Spurious buffers: these are deletions of many 1's that caused the algorithm to mistakenly identify a buffer inside an inner codeword.
3. Wrong decoding of inner codewords: these occur when the algorithm fails to decode correctly a corrupted inner codeword.

The first and second error types can happen in the first stage of the algorithm, i.e., when the decoder identifies the buffers between blown-up inner codewords. First, the decoder might not identify a buffer when a large portion of the buffer was deleted and second, the decoder might mistakenly think that there is a buffer inside an inner codeword if many consecutive runs of the symbol 1 were deleted. We show by using simple concentration bounds that both error types happen with exponentially small probability in m, the inner code block length (as $m = O(1)$ this is a constant probability, but it is still small enough to allow our construction to work). The third error type we consider is when the edit distance between the sent inner codeword and the corresponding string obtained from the second step of the decoding algorithm is greater than $\delta_{\text{in}}m$, the inner code's decoding radius. In this case, the decoding algorithm of the inner code might output a wrong codeword. While this can happen, we show that the *expected* edit distance between the original inner codeword and the decoded inner codeword[1] is smaller than $\delta_{\text{in}}m$, for a large enough m, and furthermore, the edit distance is concentrated around its mean. Hence, we expect to decode successfully most of the inner codewords. Finally, we show that this reasoning implies that the decoding algorithm of the outer code, which is executed at the last step of our decoding algorithm, succeeds with probability $1 - \exp(-\Omega(n))$.

In terms of complexity, we show that even though the construction and decoding of the inner code are exponential in the inner code's block length, the overall complexity (construction, encoding, and decoding) is dominated by the complexity of the outer code which has polynomial time encoding and decoding algorithms thanks to [HS17].

Comparison to [GL18]. We end this high-level summary by elaborating more on the main similarities and differences between our construction and the construction of Guruswami and Li [GL18]. Our scheme, as well as our decoding algorithm, follow closely the scheme and algorithm of Guruswami and Li. In particular, at a high level, the encoding layers are the same as in [GL18], meaning that both constructions use concatenation with the outer code from [HS17], place long buffers between inner codewords and blow-up the code. Since the encoding layers are similar, the decoding steps in both are also similar: first identify the buffers, then use a threshold to distinguish between 1-runs and 2-runs, then use brute force to decode the inner codewords, and finally use the decoder of [HS17]. The main differences between our scheme and the scheme from [GL18] are in the inner code that is used, the blow-up process which is finer in our scheme and our analysis which is more fine-tuned:

[1] In the proof we use the term decoded window as we are never really sure when a codeword started and ended, but this does not affect the intuition.

- The inner code that was used in [GL18] has the property that every codeword consists of 1-runs and 2-runs, but they do not have restriction on the number of 1-runs and 2-runs. In contrast, in our work, all inner codewords have the same number of 1-runs and 2-runs. This property allows us to increase the rate of the inner code compared to [GL18] (See Propositions 5.3.4 and 5.3.5 and the discussion following them), while maintaining its robustness against insertions and deletions.
- In [GL18] the authors blow-up the code by replacing every bit with $60/(1-p)$ copies of that bit. Instead of blowing-up every single bit, we blow-up each 1-run to an N_1-run and each 2-run to an N_2-run where $N_1 \neq N_2$ and both are significantly smaller than $60/(1-p)$ ($N_1 \approx 6/(1-p)$ for example). Thus, the effect of the blow-up on the rate of our code is significantly smaller than in [GL18].
- We improve on the analysis in [GL18], of the edit distance between decoded inner codewords and the original inner codewords, by better accounting the effect of decoding errors on the edit distance. One more improvement lies in our analysis where instead of using the Chernoff bound to upper bound the probability of certain events, we use the fact that binomial distributions with fixed expectations converge to a Poisson distribution. This gives a better upper bound which eventually leads to some saving when optimizing parameters. Our analysis further highlights the tight connection between the BDC and the PRC via the convergence of the binomial distribution to the Poisson distribution.

These modifications, as well as a careful choice of parameters, is the reason for the great saving in the rate compared to [GL18].

5.1.7 Organization

The chapter is organized as follows. In Section 5.2 we introduce the basic notation as well as some well known facts from probability and from previous papers. Section 5.3 contains the construction of our inner code. In Section 5.4 we give our construction and in Section 5.5 we give its analysis. We give slightly improved bounds for fixed values of p in Section 5.6. Finally, Section 5.7 explains how to carry our construction and analysis to the PRC.

5.2 Preliminaries

Throughout this chapter, $\log(x)$ refers to the base-2 logarithm and $h(x)$ denotes the binary entropy function, that is, $h(x) = -x\log(x) - (1-x)\log(1-x)$, for $0 < x < 1$.

Definition 5.2.1. *Let s be a string. A* run *r in s is a single-symbol substring of s such that the symbol before the run and the symbol after the run are different from the symbol of the run. A run of length ℓ will be denoted as ℓ-run.*

For example, consider the string $\langle 0111001 \rangle$. It can be written as the (string) concatenation of the alternating runs $0 \circ 111 \circ 00 \circ 1$. Clearly, every binary string is a concatenation of runs of alternating symbols. The following lemma of Levenshtein will be useful in the analysis of the rate of our inner code.

Lemma 5.2.2. *[Lev66] Let s be a string and let $r(s)$ be the number of runs in s. There are at most*

$$\binom{r(s)+d-1}{d}$$

different subsequences of s of length $|s|-d$.

5.2.1 Facts from Probability

We use two probability distributions in this chapter. The *binomial distribution* with parameters n and p, denoted $\mathrm{Bin}(n,p)$, is the discrete probability distribution of the number of successes in a sequence of n

independent trials, where the probability of success in each trial is p and the probability of failure is $1-p$. The second distribution is the *discrete Poisson distribution* with parameter λ, denoted as Poisson(λ) which is defined with the following probability mass function

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}.$$

A well known fact about Poisson distribution is

Lemma 5.2.3. *[MU05, Lemma 5.2] Let X and Y be two independent Poisson random variables with parameters μ_1 and μ_2. I.e., $X \sim \text{Poisson}(\mu_1)$ and $Y \sim \text{Poisson}(\mu_2)$. Then $Z = X + Y$ is a Poisson random variable with parameter $\mu_1 + \mu_2$.*

We shall use the following simple lemma in our analysis:

Lemma 5.2.4. *Fix T to be a non negative integer and let $Y(\lambda) \sim \text{Poisson}(\lambda)$. Then the function*

$$f(\lambda) := \Pr[Y(\lambda) \leq T] = e^{-\lambda} \sum_{i=0}^{T} \frac{\lambda^i}{i!}$$

is monotonically decreasing in λ.

Proof. It holds that

$$\frac{\mathrm{d}f}{\mathrm{d}\lambda}(\lambda) = -e^{-\lambda} \sum_{i=0}^{T} \frac{\lambda^i}{i!} + e^{-\lambda} \sum_{i=0}^{T-1} \frac{\lambda^i}{i!} = -e^{-\lambda} \frac{\lambda^T}{T!} < 0 \,.$$

□

The next theorem shows that if we let n tend to infinity and p tend to zero under the restriction that $p \cdot n = \lambda$, then the binomial distribution converges to the Poisson distribution with parameter λ.

Theorem 5.2.5. *[MU05, Theorem 5.5] Let $\lambda > 0$ be fixed. Let $\{X_n\}$ be a sequence of binomial random variables such that $X_n \sim \text{Bin}(n, p)$, and $\lim_{n\to\infty} np = \lambda$. Then, for any fixed k,*

$$\lim_{n\to\infty} \Pr[X_n = k] = \frac{e^{-\lambda}\lambda^k}{k!} \,.$$

The next theorem provides more information about the binomial distribution in the regime where $np = \lambda$. Specifically, it tells us when is $\Pr[X \leq T]$ an increasing function of n.

Theorem 5.2.6. *[AS65, Theorem 2.1] Let $\{X_n\}$ be a sequence of binomial random variables with parameters n and $p = \lambda/n$. Let T be some parameter. Set $f(n) := \Pr[X_n \leq T]$.*

1. *If $T \leq \lambda - 1$ then for every $n \geq \lambda$, $f(n)$ is monotonically increasing in n.*
2. *If $\lambda \leq T$ then for every $n \geq T$, $f(n)$ is monotonically decreasing in n.*

For concentration bounds, we will use the following versions of the Chernoff bounds.

Lemma 5.2.7. *[MU05, Theorems 4.4 and 4.5] Suppose $X_1, \ldots, X_n$ are independent identically distributed random variables taking values in $\{0, 1\}$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X_i]$. Then, for any $0 < \alpha < 1$:*

$$\Pr[X \geq (1+\alpha)n\mu] \leq e^{-\frac{\mu n \alpha^2}{3}}$$

and

$$\Pr[X \leq (1-\alpha)n\mu] \leq e^{-\frac{\mu n \alpha^2}{2}} \,.$$

When we have a Poisson random variable we shall use the following Chernoff bound

Lemma 5.2.8. *[MU05, Theorem 5.4] Let X be a Poisson random variable with parameter μ.*

1. *If $x > \mu$, then*
$$\Pr(X \geq x) \leq \frac{e^{-\mu}(e\mu)^x}{x^x} .$$
2. *If $x < \mu$,*
$$\Pr(X \leq x) \leq \frac{e^{-\mu}(e\mu)^x}{x^x} .$$

Another concentration bound we use is Hoeffding's inequality

Theorem 5.2.9. *[Hoe94, Theorem 2] If $X_1, X_2, \ldots, X_n$ are independent random variables with finite first and second moment and $a_i \leq X_i \leq b_i$ for $1 \leq i \leq n$. Let $X = \sum_{i=1}^{n} X_n$ and $\mu = \mathbb{E}[X]$ then for $t > 0$*
$$\Pr[X - \mu > t] < \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) .$$

To approximate binomial coefficients we shall use the following lemma

Lemma 5.2.10. *For any $n, k \in \mathbb{N}$ such that $k/n \leq 1/2$ we have,*
$$2^{nh\left(\frac{k}{n}\right) - O(\log n)} \leq \binom{n}{k} \leq 2^{nh\left(\frac{k}{n}\right)} .$$

The proofs of the bounds follow from Stirling's formula, e.g., see [GRS12, Section 3.2]

5.2.2 The Code of Haeupler and Shahrasbi [HS17]

Our construction relies on the following code of Haeupler and Shahrasbi [HS17].

Theorem 5.2.11 ([HS17, Theorem 1.1])**.** *For every $\varepsilon_{\text{out}} > 0$ and $\delta_{\text{out}} \in (0, 1)$ there exists n_0 so that for every $n > n_0$ there is an integer k satisfying $k/n > 1 - \delta_{\text{out}} - \varepsilon_{\text{out}}$, an alphabet Σ of size $O_{\varepsilon_{\text{out}}}(1)$ and encoding and decoding maps $E : \Sigma^k \mapsto \Sigma^n$, $D : \Sigma^* \mapsto \Sigma^k$, respectively, such that if $ED(E(x), y) \leq \delta_{\text{out}} n$ then $D(y) = x$. Further E and D are explicit and can be computed in linear and quadratic time in n, respectively.*

We shall denote with $\mathcal{R}_{\text{out}} := k/n$ the rate of this code, which will be used as the outer code in our construction.

5.3 The Inner Code

In this section we describe the construction of our inner code. Before giving the construction we define a set of strings from which we shall pick our codewords.

Definition 5.3.1. *We denote with $S \subset \{0, 1\}^*$ the set containing all binary strings s that start and end with the symbol 1 and that contain only 1-runs and 2-runs.*

Let $\beta_1 \in [0, 1]$. Define $S_{m,\beta_1} \subset S$ to be the set of all $s \in S$ of length m, such that the number of 1-runs in s is exactly $\beta_1 m$ and the number of 2-runs in s is exactly $\beta_2 m = (1 - \beta_1)m/2$. Denote $\beta := \beta_1 + \beta_2$.

Remark 5.3.2. *We observe that as every string in S begins and ends with the same symbol, the number of runs in it is odd. Accordingly, βm, in the definition of S_{m,β_1}, is an odd integer.*

Our goal in this section is to construct a code $C \subset S_{m,\beta_1}$ such that the length of a longest common subsequence of any two different codewords is $< m - \delta m$.

Remark 5.3.3. *Observe that the deletion channel is likely to output a word that is not in S. However, Algorithm 5 (given in Section 5.5), for decoding the final code, contains a threshold decoding step (Step 2 in Algorithm 5) that always returns a binary string in S. We further restrict our attention to S_{m,β_1} as it somewhat simplifies the analysis.*

We construct this code using the natural greedy algorithm: We consider all strings in $S_{m,\beta_1} \subset S$ and greedily choose strings that are far from each other. To reason about the parameters of the code, we need the following propositions.

The first proposition gives an upper bound on the size of the "deletion ball", i.e., given a string $s \in S_{m,\beta_1}$ it upper bounds the number of different subsequences of s of length $m - \delta m$, that belong to S.

Proposition 5.3.4. *Let $s \in S_{m,\beta_1}$. It holds that*

$$\#\left\{s' \in S \mid s' \text{ is a subsequence of } s \text{ and } |s'| = m - \delta m\right\} \leq \binom{(\beta+\delta)m}{\delta m}.$$

Proof. By definition, s is a binary string that contains exactly βm runs. Let H be the set of all the subsequences obtained from s by applying δm deletions. According to Lemma 5.2.2, the size of H is at most $\binom{\beta m+\delta m-1}{\delta m} < \binom{\beta m+\delta m}{\delta m}$. Clearly if we restrict further and consider only those strings in $H \cap S$ we can only decrease the size of the set. □

The second proposition upper bounds the size of the "insertion ball", i.e., given a string $s' \in S$ of length $m - \delta m$, it gives an upper bound on the number of strings $s \in S_{m,\beta_1}$ that can be obtained from s' by performing δm insertions. The proof of this proposition is considerably more elaborate.

Proposition 5.3.5. *Let $0 < \delta < \beta_1/3$. Fix $s_{\text{sub}} \in S$ such that $|s_{\text{sub}}| = m - \delta m$. The number of binary strings in S_{m,β_1} that contain s_{sub} as a subsequence is at most $\binom{\beta m}{\delta m}$.*

We note that the equivalent propositions from [GL18] gave upper bounds of $\binom{m}{\delta m}$ and $O(\delta m) \cdot \binom{m}{\delta m}$ respectively. The main reason for our saving is that we restrict our codewords to have exactly $\beta_1 m$ 1-runs and $\beta_2 m$ 2-runs. This saving is one of the places where we improve upon [GL18]. This improvement affects the rate of the inner code as we shall later see.

The proof of the proposition relies on an algorithm for generating all strings $s \in S_{m,\beta_1}$ such that s_{sub} is a subsequence of s. As in [GW17a, Lemma 2.3], in order to avoid over counting, we will generate all such s by finding the lexicography first occurrence of s_{sub} in s. We first explain the idea behind the algorithm and then prove Proposition 5.3.5.

Denote $s_{\text{sub}} = \langle b_1 b_2 \dots b_{m-\delta m} \rangle$, where $b_i \in \{0,1\}$. In order to obtain a string $s \in S_{m,\beta_1}$ from s_{sub} we need to choose indices $1 \leq n_1 < n_2 < \dots < n_{m-\delta m} \leq m$ for the bits of s_{sub} in s. Moreover, to make sure that the locations chosen are indeed the lexicography first occurrence of s_{sub} in s, the entries between n_i and n_{i+1} (for $1 \leq i \leq m - \delta m - 1$) must contain the opposite bit of the symbol in location n_{i+1}.

Since both s_{sub} and s consist of just 1-runs and 2-runs, this puts some restrictions on the embedding of s_{sub} in s, e.g., we cannot have $n_{i+1} - n_i > 3$ (all locations between them (and maybe longer) are identical and hence give a run that is too long). In particular, and more formally, we have the following restrictions

1. The first bit in s_{sub} must be located as the first bit in s. This is because the first bit in s must be a 1 bit. I.e., $n_1 = 1$.

2. Let b_i be a 1-run in s_{sub} and assume w.l.o.g. that it is a 0 bit. Its location, n_i, must be chosen such that the location of the next bit in s_{sub}, b_{i+1}, is either

   (a) $n_{i+1} = n_i + 1$. I.e., $\langle b_i, b_{i+1} \rangle = \langle 01 \rangle$ in s_{sub} is mapped to $\langle 01 \rangle$ in s, or

   (b) $n_{i+1} = n_i + 2$. I.e., $\langle 01 \rangle$ in s_{sub} is mapped to $\langle 001 \rangle$ in s.

   The case where $b_i = 1$ is completely analogous.

3. Let b_i, b_{i+1} be a 2-run in s_{sub} (i.e., the symbols of b_i and b_{i+1} are the same) and assume w.l.o.g. that both symbols are 0. The locations n_i, n_{i+1}, n_{i+2} of b_i, b_{i+1}, b_{i+2} (b_{i+2} is a 1 bit) in s must be chosen in accordance with one of the following cases:

   (a) $n_{i+1} = n_i + 1$ and $n_{i+2} = n_i + 2$. I.e., $\langle 001 \rangle$ in s_{sub} is mapped to $\langle 001 \rangle$ is s.

   (b) $n_{i+1} = n_i + 2$ and $n_{i+2} = n_i + 3$. I.e., $\langle 001 \rangle$ in s_{sub} is mapped to $\langle 0101 \rangle$ in s.

   (c) $n_{i+1} = n_i + 2$ and $n_{i+2} = n_i + 4$. I.e., $\langle 001 \rangle$ in s_{sub} is mapped to $\langle 01001 \rangle$ in s.

   (d) $n_{i+1} = n_i + 3$ and $n_{i+2} = n_i + 4$. I.e., $\langle 001 \rangle$ in s_{sub} maps to $\langle 01101 \rangle$ in s.

   (e) $n_{i+1} = n_i + 3$ and $n_{i+2} = n_i + 5$. I.e., $\langle 001 \rangle$ in s_{sub} is mapped to $\langle 011001 \rangle$ in s.

   The case where $b_i = 1$ is completely analogous.

4. If $n_{m-\delta m} < m$, then the remaining bits of s must be filled with 1-runs and 2-runs such that the total number of 1-runs and 2-runs is exactly $\beta_1 m$ and $\beta_2 m$, respectively.

It is not hard to verify that any arrangement that does not follow the restrictions above will either contain a run of length 3 or more, will not have the right number of 1-runs, or will not correspond to the lexicographically first embedding of s_{sub} in s.

We shall think of the cases above as describing operations that can be performed on a string s'. E.g. if $s' = \langle 101001 \rangle$ and we apply 3e to the last three bits in s' then we will get the string $\langle 1010\mathbf{11}0\mathbf{0}1 \rangle$, where the bold symbols are the symbols that were added from the application of 3e (in other words, the symbols that are not bold are the embedding of the original string). If we then apply, say, 2b to the second and third bits of the new string then we will get the string $\langle 10\mathbf{0}10\mathbf{11}0\mathbf{0}1 \rangle$ etc.

To simplify matters note that if we consider a 2-run in s', say $\langle 001 \rangle$ and we wish to apply 3c on it, i.e. map it to $\langle 01001 \rangle$ in s, then we can think about this as first applying 3b to $\langle 001 \rangle$, obtaining the string $\langle 0101 \rangle$ and then applying to the last two bits 2b, getting the string $\langle 01001 \rangle$. I.e. we can simulate 3c by first applying 3b and then applying 2b. Similarly, we can simulate each of the operations 3d and 3e using 3b and then applying 2b to the appropriate bits (for 3e we need to apply 3b and then 2b to two different locations).

Using the above terminology, we next describe an algorithm that given a string s_{sub} generates $s \in S_{m,\beta_1}$ such that s_{sub} is a subsequence of s. The algorithm will first select a subset of the 2-runs in s_{sub} and apply 3b to them. Then it will add more 1-runs to the resulting string, locating them to the right of the last bit. Finally, it will apply 2b to several 1-runs.

There is a delicate point that we wish to stress before giving the algorithm. In this last step we restrict the 1-runs to which we can apply 2b. To illustrate why the restriction is needed, consider the following example: Consider the string $\langle 001 \rangle$ and apply 3b to it. This generates the string $\langle 0\mathbf{1}01 \rangle$, where, as before, the bold symbols represent the symbols that were added in the embedding. If we now apply 2b to the first two bits then we would get $\langle 0\mathbf{01}01 \rangle$. This however, is not the first lexicographical embedding of $\langle 001 \rangle$ in $\langle 00101 \rangle$ (which is $\langle 001\mathbf{01} \rangle$). Thus, if we wish to construct a lexicographically first embedding of s_{sub} in the resulting string s then in the last step, where we apply 2b to several runs, we should never apply 2b to the first bits resulting from the application of 3b in the first step.

In view of the above discussion we say that a 1-run is *frozen* if it is the first bit of a substring that resulted from applying 3b. In other words, a 1-run is not frozen if it is either an original 1-run of s_{sub}, a 1-run that was added in the second step, or if it is the 2nd or 3rd bits generated by applying 3b (i.e. if we had $\langle 001 \rangle \to \langle 0101 \rangle$ then the non-frozen 1-runs are the bold bits $\langle 0\mathbf{1}0\mathbf{1} \rangle$, and the last bit may also be non-frozen).

Let r_1 and r_2 be the number of 1-runs and 2-runs in s_{sub} and let x be an integer such that $0 \leq x \leq \delta m$ and $r_1 + r_2 + 2x \leq \beta m$.

Algorithm 4: Embed

input : $s_{\text{sub}} \in S$ such that $|s_{\text{sub}}| = m - \delta m$,
and $0 \leq x \leq \delta m$ where $r_1 + r_2 + 2x \leq \beta m$

output: A string $s \in S_{m,\beta_1}$ such that s_{sub} is a subsequence of s

[1] Select x 2-runs in s_{sub} and apply 3b to them.

`/* total number of 1-runs is` $r_1 + 3x$ `and of 2-runs is` $r_2 - x$ `*/`

`/* total number of non-frozen 1-runs is` $r_1 + 2x$ `*/`

[2] Add $\beta m - r_1 - r_2 - 2x$ many 1-runs to the right of the string

`/* total number of runs is` βm `and number of 1-runs is` $\beta m - r_2 + x$ `*/`

`/* total number of non-frozen 1-runs is */`

`/*` $\beta m - r_1 - r_2 - 2x + r_1 + 2x = \beta m - r_2$ `*/`

[3] Select $\delta m - (\beta m - r_1 - r_2 - x)$ non-frozen 1-runs and apply 2b to each of them

`/* length of resulting string is exactly` m `*/`

Claim 5.3.6. *Algorithm 4 returns a string in* S_{m,β_1}.

Proof. Step 1 turns each of the chosen x 2-runs into three 1-runs, only two of which are non-frozen. Hence, the number of 2-runs is $r_2 - x$, the number of 1-runs is $r_1 + 3x$ and the number of non-frozen 1-runs is $r_1 + 2x$.

Step 2 completes the number of runs to βm by introducing $\beta m - r_1 - r_2 - 2x$ new 1-runs[2]. The total number of 2-runs did not change, the total number of 1-runs is now

$$(r_1 + 3x) + (\beta m - r_1 - r_2 - 2x) = \beta m - r_2 + x$$

and the number of non-frozen 1-runs is

$$(r_1 + 2x) + (\beta m - r_1 - r_2 - 2x) = \beta m - r_2 \ .$$

Step 3 turns $\delta m - (\beta m - r_1 - r_2 - x)$ 1-runs into 2-runs. We now show that this gives a string in S_{m,β_1}. For this we need to show that it has only 1-runs and 2-runs and the correct number of runs of each type. The fact that we only get 1- and 2-runs follows from the definition of our operations. Now, the resulting number of 1-runs is

$$\begin{aligned}(\beta m - r_2 + x) - (\delta m - (\beta m - r_1 - r_2 - x)) &= 2\beta m - \delta m - 2r_2 - r_1 \\ &= 2\beta m - \delta m - (m - \delta m) \\ &= 2\beta m - m \\ &= \beta_1 m \ ,\end{aligned}$$

where we have used the facts that $m - \delta m = |s_{\text{sub}}| = 2r_2 + r_1$, that $\beta = \beta_1 + \beta_2$ and that $m = \beta_1 m + 2\beta_2 m$. Similarly, the number of 2-runs is

$$\begin{aligned}(r_2 - x) + (\delta m - (\beta m - r_1 - r_2 - x)) &= r_1 + 2r_2 + \delta m - \beta m \\ &= m - \delta m + \delta m - (\beta_1 + \beta_2)m \\ &= \beta_2 m \ .\end{aligned}$$

Note that by our construction, the string begins with a 1 and it also ends with a 1 as the total number of runs is odd (recall Remark 5.3.2). Thus, the resulting string is in S_{m,β_1} as claimed. □

The next claim shows that any $s \in S_{m,\beta_1}$ that contains s_{sub} as a subsequence can be obtained from the algorithm for an appropriate choice of $0 \leq x \leq \delta m$.

[2]Note that since $r_1 + r_2 + 2x \leq \beta m$, this operation is well defined.

Claim 5.3.7. *For any $s \in S_{m,\beta_1}$ that contains s_{sub} as a subsequence, there exists an $0 \le x \le \delta m$ and appropriate choices for the different steps of the algorithm so that the resulting string is s.*

Proof. As the proof contains many tedious details, we leave some of the arguments to the reader. Let $s \in S_{m,\beta_1}$ and denote by $1 = n_1 < n_2 < \ldots < n_{m-\delta m} \le [n]$ the embedding of s_{sub} in s that corresponds to the first lexicographic appearance of s_{sub} in s. By the discussion above, there are restrictions on the values n_i. Specifically, when we embed a 1-run from s_{sub} in s, then n_i and n_{i+1} must satisfy one of the rules 2a or 2b and when we embed a 2-run, we must follow one of the rules 3a–3e.

Observe that the locations $n_1, \ldots, n_{m-\delta m}$ determine how to embed s_{sub} into s, simply by going over all the runs in s_{sub} from left to right. This can be easily proved by induction. For example, assume that the first run of s_{sub} is $\langle 110 \rangle$ and s starts with $\langle 10010 \rangle$. Then $n_1 = 1$, $n_2 = 4$, and $n_3 = 5$ and this implies that we need to perform 3d to the first run in s_{sub}. Now, our algorithm, when going over the first 2-run in Step 1, returns the string $\langle 1010 \rangle$. Then, in Step 3 the first 0 is a non-frozen 1-run and therefore the algorithm performs 2b to this 0. This combination of 3b and 2b is equivalent to 3d, as required. Thus, the algorithm correctly embeds the first run of s_{sub} into s. We now move to the next run etc.

Notice that by the description above, after embedding all the runs, it may be the case that $n_{m-\delta m} < m$. However, looking back at the algorithm, in Step 2 we add several 1-runs to the embedding. Then, in Step 3, it may be the case that some of these 1-runs will be turned to 2-runs by the algorithm (all the added 1-runs to the right of the string are non-frozen). This ensures that the number of runs in the resulting string after Step 3 is exactly βm and that we still have a valid embedding of s_{sub}.

Finally, note that there is a value of x that will make the algorithm embed s_{sub} successfully. The proof in the previous paragraph determines x exactly. It is the number of times that we applied 3b in the embedding. Clearly, this number is at most δm as otherwise the size of the resulting string will be larger than m. Also note that $r_1 + r_2 + 2x \le \beta m$ since otherwise we create a string that contains more than βm runs. Thus, x as we just defined satisfies both requirements $0 \le x \le \delta m$ and $r_1 + r_2 + 2x \le \beta m$. □

To conclude, if we consider all possible values x can take, and all the possibilities to perform the choices in the algorithm we get an upper bound on the number of strings $s \in S_{m,\beta_1}$ which contain s_{sub} as a subsequence. We are now ready to prove Proposition 5.3.5.

Proof of Proposition 5.3.5. By the argument above it is enough to count the number of possibilities for x and the number of possible choices made by the algorithm. For any choice of x, there are exactly $\binom{r_2}{x}$ ways of selecting x many 2-runs in Step 1 of Algorithm 4. In Step 2 of the algorithm we have no freedom since we add the new 1-runs at the end of the string. Finally, in Step 3 we have $\binom{\beta m - r_2}{\delta m - (\beta m - r_1 - r_2 - x)}$ many ways to choose $\delta m - (\beta m - r_1 - r_2 - x)$ many 1-runs among the non-frozen 1-runs.

We first note that

$$\binom{\beta m - r_2}{\delta m - (\beta m - r_1 - r_2 - x)} \le \binom{\beta m - r_2}{\delta m - x} .$$

Indeed, as $r_2 \le \beta_2 m + \delta m$ it follows that $\beta m - r_2 \ge \beta_1 m + \delta m \ge 2\delta m$. In addition, observe that $\delta m - (\beta m - r_1 - r_2 - x) \le \delta m - x$ as otherwise Algorithm 4 performs more than δm operations, in contradiction. As the binomial coefficients are monotonically increasing up to $(\beta m - r_2)/2 \ge \delta m$ the inequality follows.

Hence, the total number of strings that can be obtained from the algorithm is upper bounded by

$$\sum_{x=0}^{\delta m} \binom{r_2}{x} \binom{\beta m - r_2}{\delta m - (\beta m - r_1 - r_2 - x)} \le \sum_{x=0}^{\delta m} \binom{r_2}{x} \binom{\beta m - r_2}{\delta m - x} = \binom{\beta m}{\delta m} ,$$

where the equality follows by Vandermonde's identity (E.g., [Tuc94, Chapter 5.5, Identity (10)]). □

Armed with Propositions 5.3.4 and 5.3.5 we now show the existence of an appropriate inner code.

Proposition 5.3.8. *Let $0 \le \beta_1, \delta \le 1$ be parameters. Let $\beta = \frac{1+\beta_1}{2}$. For every $\varepsilon > 0$ there is M_ε so that for every $m > M_\varepsilon$ there is a set $C \subseteq S_{m,\beta_1}$ of size $|C| = 2^{\lfloor m \mathcal{R}_{\text{in}} \rfloor}$ where*

$$\mathcal{R}_{\text{in}} = \beta h\left(\frac{\beta_1}{\beta}\right) - (\delta + \beta) h\left(\frac{\delta}{\delta + \beta}\right) - \beta h\left(\frac{\delta}{\beta}\right) - \varepsilon ,$$

such that for every $c \neq c' \in C$ it holds that any string $s_{\text{sub}} \in S$ that is a subsequence of both c and c' is of length $|s_{\text{sub}}| < m - \delta m$.

Proof. We first note that the number of binary strings in S_{m,β_1} is exactly $\binom{\beta m}{\beta_1 m}$ as we have $\binom{\beta m}{\beta_1 m}$ ways to arrange the $\beta_1 m$ 1-runs and the $\beta_2 m$ 2-runs.

The construction of C is done greedily. We go over all strings $s \in S_{m,\beta_1}$ and add them to C one by one as long as they do not share a common subsequence that is too long (from S) with any string that is already in C. Propositions 5.3.4 and 5.3.5 imply that any $s \in S_{m,\beta_1}$ contains at most $\binom{\beta m+\delta m}{\delta m}$ many subsequences of length $m - \delta m$ from S, and each such string is a subsequence of at most $\binom{\beta m}{\delta m}$ strings in S_{m,β_1}. Thus, whenever we add a string to C we exclude at most

$$\binom{\beta m + \delta m}{\delta m} \cdot \binom{\beta m}{\delta m}$$

other strings from being in C. Therefore, our codebook contains at least

$$|C| \geq \frac{\binom{\beta m}{\beta_1 m}}{\binom{\beta m+\delta m}{\delta m}\binom{\beta m}{\delta m}} \geq 2^{m\left(\beta h\left(\frac{\beta_1}{\beta}\right)-(\delta+\beta)h\left(\frac{\delta}{\delta+\beta}\right)-\beta h\left(\frac{\delta}{\beta}\right)\right)-O(\log m)}$$

codewords, where the inequality follows by Lemma 5.2.10. Thus, for every $\varepsilon > 0$ there exists large enough $m > 0$ such that the constructed set $C \subset S_{m,\beta_1}$ is of size $2^{\lfloor m\mathcal{R}_{\text{in}} \rfloor}$ where $\mathcal{R}_{\text{in}} = \beta h\left(\frac{\beta_1}{\beta}\right) - (\delta+\beta)h\left(\frac{\delta}{\delta+\beta}\right) - \beta h\left(\frac{\delta}{\beta}\right) - \varepsilon$. □

By construction, the code C can handle an adversary that, given a codeword $c \in C$, returns a subsequence $s_{\text{sub}} \in S$ of c where $|s_{\text{sub}}| \geq m - \delta m$. That is, we can uniquely identify the original codeword c from s_{sub}. Our next goal is to show that our code can handle the usual edit distance adversary. In other words, it can handle an adversary that performs any δm insertion and deletion (and hence it is not bound to return a string in S). The key observation is that if we look at two different codewords $c, c' \in C$ and denote by s a longest common subsequence between c and c' then it must be that there exists $s' \in S$ that is also a subsequence of c and c' and $|s| = |s'|$.

Proposition 5.3.9. *Let C be the code constructed in Proposition 5.3.8. For any two codewords $c, c' \in C$ it holds that $ED(c, c') > 2\delta m$.*

Proof. Let $c \neq c' \in C$. Lemma 3.1.3 gives

$$\text{ED}(c, c') = |c| + |c'| - 2\,|\text{LCS}(c, c')| = 2m - 2\,|\text{LCS}(c, c')| \ .$$

Observe that if s is a longest common subsequence of c and c' then there is a string $s_{\text{sub}} \in S$, such that $|s_{\text{sub}}| = |s|$ and s_{sub} is also a common subsequence of c and c'. Indeed, let s be a longest common subsequence of c and c'. Since both c and c' start and end with 1, s also starts and ends with 1. Moreover, for every three consecutive, equal bits in s, we can flip the second bit and the resulting string will still be a common subsequence of maximal length (as neither c nor c' contain a run of length 3 or more). Repeating this we will get a string only containing 1-runs and 2-runs, i.e. a string in S.

As C was constructed so that not two codewords in C share a common subsequence (from S) of length larger or equal to $m - \delta m$, it follows that

$$\text{ED}(c, c') = 2m - 2\,|\text{LCS}(c, c')| > 2m - 2(m - \delta m) = 2\delta m \ .$$

□

Remark 5.3.10. *Note that C can be constructed in time at most $O\left(2^{2m} \cdot m^2\right)$ as in the worst case we compute the edit distance between any two possible strings.*

5.4 Construction

In this section we give a construction of a code for the BDC_p. Throughout this section we fix p.

We repeat the high level description of the construction from Section 5.1.6 (and as depicted in Figure 5.1 on page 61). We first do code concatenation. As outer code we use the one given in [HS17, Theorem 1.1] (restated as Theorem 5.2.11 here). As the inner code we use the code constructed in Proposition 5.3.8. Then, in order to protect the concatenated codeword from a large number of deletions, we first place a *buffer* of zeros between every two consecutive inner codewords. Since the decoder first looks for the buffers in order to identify where an inner code starts and where it ends, this step helps to reduce the amount of synchronization errors in the outer code. Secondly, we *blow-up* the inner codewords by replacing every run of length 1 with a run of length N_1 and every run of length 2 with a run of length N_2, where the symbols of the runs are preserved. For example, $\langle 11 \rangle$ turns into $\langle 1^{N_2} \rangle$ and $\langle 0 \rangle$ is replaced with $\langle 0^{N_1} \rangle$. If we choose N_1 and N_2 appropriately, then (with high probability) the decoder will identify the original run length.

We now give a formal description of our construction.

The parameters: At this point, we do not specify the parameters explicitly. We prefer to first present the scheme and analyze it before optimizing the parameters. However, the order by which we choose the parameters is important as there are some dependencies among them. First, we choose $M_1, M_2, \beta_1, M_B, \delta_{\mathrm{out}}$ to be fixed constants. One should have in mind that $M_1 < M_2$ are the quantities through which N_1 and N_2 are determined, i.e., they determine how we blow-up the different types of runs. Then we choose δ_{in} to be larger than some quantity $\gamma = \gamma(M_1, T, M_2, \beta_1)$ that we later define (see Proposition 5.5.1). At this point, we can compute the value of $\mathcal{R}_{\mathrm{in}}$, the rate of the inner code, using Proposition 5.3.8. Then, we choose a small enough $\varepsilon_{\mathrm{out}}$ that determines the alphabet size of the outer code C_{out}. Denote with $\mathcal{R}_{\mathrm{out}}$ the rate of C_{out} and by n its block length. Finally, we pick m, the block length of the inner code, to satisfy[3] $\Sigma = \{0,1\}^{m \cdot \mathcal{R}_{\mathrm{in}}}$.

While this may seem a bit confusing the main thing to remember is that $\varepsilon_{\mathrm{out}}$ that was picked at the end, can be taken to be as small a constant as we wish, or, in other words, we can pick m to be as large a constant as we wish. This is important as we will bound the probabilities of several bad events by expressions of the form $\exp(-\Omega(m))$ and it will be important for us to be able to pick m large enough as to make all our estimates small.

Encoding: The process of encoding starts with the outer code. Given as input a message $x \in \Sigma^{\mathcal{R}_{\mathrm{out}} n}$, we encode it with the code given in Theorem 5.2.11 to obtain an outer codeword $c^{(\mathrm{out})} = (\sigma_1, \ldots, \sigma_n) \in C_{\mathrm{out}} \subset \Sigma^n$. Then, every symbol in $c^{(\mathrm{out})}$, $\sigma_i \in \Sigma = \{0,1\}^{m \cdot \mathcal{R}_{\mathrm{in}}}$, is encoded using the inner code to a codeword that we denote $c^{(\mathrm{in})}_{\sigma_i}$. We thus get a codeword in the concatenated code

$$\left(c^{(\mathrm{in})}_{\sigma_1}, \ldots, c^{(\mathrm{in})}_{\sigma_n}\right) \in C_{\mathrm{out}} \circ C_{\mathrm{in}} \ .$$

Now that we have a codeword in the concatenated code we add additional layers of encoding that are crucial for the decoding algorithm to succeed.

1. Every two adjacent inner codewords are separated by a buffer of zeros of length $N_B = \lceil M_B \cdot m/(1-p) \rceil$.
2. In every inner codeword, we replace every 1-run with a run of length $N_1 = \lceil M_1/(1-p) \rceil$ where the symbol of the run is preserved.
3. In every inner codeword, we replace every 2-run with a run of length $N_2 = \lceil M_2/(1-p) \rceil$ where the symbol of the run is preserved.

After the buffering and blow-up process we have three different run lengths $\lceil M_B \cdot m/(1-p) \rceil, \lceil M_1/(1-p) \rceil$ and $\lceil M_2/(1-p) \rceil$. Note that the buffer's length is much larger than $\lceil M_1/(1-p) \rceil$ and $\lceil M_2/(1-p) \rceil$ since it grows with m.

[3] When we choose parameters we make sure that $m \cdot \mathcal{R}_{\mathrm{in}}$ is an integer.

Block length and rate: Note that as C_{in} contains strings in S_{m,β_1}, each of the n inner codewords becomes of length $\lceil M_1/(1-p)\rceil \cdot \beta_1 m + \lceil M_2/(1-p)\rceil \cdot \beta_2 m$. As we have $n-1$ buffers between codewords the total block length is

$$(\lceil M_1/(1-p)\rceil \cdot \beta_1 m + \lceil M_2/(1-p)\rceil \cdot \beta_2 m) \cdot n + \lceil M_B \cdot m/(1-p)\rceil \cdot (n-1) \ .$$

Since the input to the encoding is a string in $\Sigma^{\mathcal{R}_{\text{out}} n}$ (and recall that $|\Sigma| = 2^{\mathcal{R}_{\text{in}} m}$) the rate $\mathcal{R}$ of the construction is given by

$$\begin{aligned}
\mathcal{R} &= \frac{\log\left(|\Sigma|^{\mathcal{R}_{\text{out}} n}\right)}{\beta_1 \lceil M_1/(1-p)\rceil mn + \beta_2 \lceil M_2/(1-p)\rceil mn + \lceil M_B m/(1-p)\rceil (n-1)} \\
&\geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}}{\beta_1 \lceil M_1/(1-p)\rceil + \beta_2 \lceil M_2/(1-p)\rceil + M_B/(1-p) + 1/m} \\
&\geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}}{\beta_1 M_1/(1-p) + \beta_2 M_2/(1-p) + \beta + M_B/(1-p) + 1/m} \\
&= \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}(1-p)}{\beta_1 M_1 + \beta_2 M_2 + \beta(1-p) + M_B + (1-p)/m} \ .
\end{aligned} \tag{5.1}$$

We can avoid the ceilings if we consider values of p such that $\lceil M_1/(1-p)\rceil$, $\lceil M_2/(1-p)\rceil$ and $\lceil M_B m/(1-p)\rceil$ are integers. In this case, the rate is

$$\mathcal{R} \geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}(1-p)}{\beta_1 M_1 + \beta_2 M_2 + M_B} \ . \tag{5.2}$$

Run time analysis: By Theorem 5.2.11, the outer code can be constructed in linear time. Constructing the inner code requires time at most $O\left(2^{2m} \cdot m^2\right)$ (see Remark 5.3.10). As $m = \log|\Sigma| / \mathcal{R}_{\text{in}}$, we get that constructing the inner code takes time $O(m^2 \cdot 2^{2m}) = |\Sigma|^{O(1)} = O_{\varepsilon_{\text{out}}}(1)$, which is constant. Thus, as all encoding steps are done in linear time, the encoding time complexity is $O(n)$.

5.5 Correctness and Analysis

We first present the decoding algorithm and then prove its correctness. After that, we show how to choose the parameters to obtain Theorem 5.1.1.

Let y be the binary string received after transmitting $\text{Enc}(x)$. The decoding procedure is given in Algorithm 5 in page 73. Observe that the algorithm depends on some integral parameter T. When analyzing the algorithm we will see what T has to satisfy in order for the algorithm to decode successfully with high probability. For the time being it is enough to remember that $M_1 < T < M_2$.

Before proving the correctness of the algorithm we give its run time analysis.

Run time analysis of Algorithm 5. It is clear that Steps 1 and 2 take linear time. Step 3 runs the inner decoding algorithm n times. As the inner decoding algorithm is a brute force operation that is run on strings of constant length it takes constant time. Thus, the first three steps of the decoding algorithm require linear time. In Step 4 we run the decoding algorithm of [HS17] (recall Theorem 5.2.11), that requires $O(n^2)$ time. Therefore, the entire decoding procedure is dominated by the last step which runs in time $O(n^2)$.

5.5.1 Correctness of Decoding Algorithm

In this section, we prove that Algorithm 3 succeeds with high probability.

Algorithm 5: Decode with threshold T

input : Binary string y which is the output of the BDC_p on $\mathrm{ENC}(x)$
output: A message $\tilde{x} \in \Sigma^k$

[1] `/* Identifying buffers Step:` $M_B \cdot m/2$ `*/`
Every run of zeros of length longer than $M_B \cdot m/2$ is identified as a buffer.
`/* Denote by` $s_1, \ldots, s_t$ `the strings between the identified buffers. */`
[2] `/* Threshold decoding step: */`
for *every* s_i **do**
  for *every run in* s_i **do**
    if *the length of the run is longer than* T **then**
      Decode it to a run of length 2
    else
      Decode it to a run of length 1
    end
  end
end
`/* Let` $\tilde{c}_1, \ldots, \tilde{c}_t$ `be the strings obtained in this step. */`
[3] `/* Inner code decoding step: */`
Use brute-force decoding to decode each $\tilde{c}_i$ to get $\tilde{\sigma}_i$. Denote $\tilde{\sigma}^{\mathsf{out}} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n) \in \Sigma^n$
[4] `/* Outer code decoding step: */`
Run the decoding algorithm of the outer code on $(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_t)$ to obtain $\tilde{x}$
Output $\tilde{x}$

Figure 5.2: Algorithm for decoding our code over BDC_p. The algorithm is assumed to know the parameters k, n, m, M_B, T as well as C_{in} and C_{out}.

Proposition 5.5.1. *Given* $M_1, T, M_2, M_B, \beta_1, \delta_{\mathrm{in}}, \varepsilon_{\mathrm{in}}, \delta_{\mathrm{out}}$ *(as described in Section 5.4) let* $Z_1 \sim \mathrm{Bin}(\lceil M_1/(1-p) \rceil, 1-p)$ *and* $Z_2 \sim \mathrm{Bin}(\lceil M_2/(1-p) \rceil, 1-p)$*. Denote*

$$\begin{aligned} P^{(1)\to(2)} &:= \Pr[Z_1 \geq T+1], \\ P^{(1)\to(0)} &:= \Pr[Z_1 = 0], \\ P^{(2)\to(1)} &:= \Pr[Z_2 \leq T], \\ P^{(2)\to(0)} &:= \Pr[Z_2 = 0], \end{aligned}$$

and define

$$\gamma := \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2) P^{(1)\to(0)} + 4\beta_2 P^{(2)\to(0)} \tag{5.3}$$

(the reason for this definition of γ *is revealed later). If* $\gamma < \delta_{\mathrm{in}}$*, then there exists* $\epsilon_0 = \epsilon_0(M_1, T, M_2, M_B, \beta_1, \delta_{\mathrm{in}}, \delta_{\mathrm{out}})$ *such that for every* $\varepsilon_{\mathrm{out}} < \epsilon_0$ *the following holds. Let* $x \in \Sigma^{\mathcal{R}_{\mathrm{out}} n}$ *(where* $|\Sigma| = O_{\varepsilon_{\mathrm{out}}}(1)$*) be a message and let* y *be the string obtained after encoding* x *using our code and transmitting it through the* BDC_p*. Then, Algorithm 5 returns* x *with probability* $1 - \exp(-\Omega(n))$ *when given* y *as input.*

Observe that as $\epsilon_0 = \epsilon_0(M_1, T, M_2, M_B, \beta_1, \delta_{\mathrm{in}}, \delta_{\mathrm{out}})$, it does not depend on m and n. The rest of Section 5.5.1 is devoted to proving Proposition 5.5.1. We first discuss the structure of the proof and prove relevant lemmas. The actual proof is given at the end of this section.

Let $\sigma^{\mathsf{out}} = (\sigma_1, \ldots, \sigma_n) \in \Sigma^n$ be the result of encoding x with the outer code. I.e. the first step before concatenating with our inner code. Let $c_{\sigma_i}^{(\mathsf{in})}$ be the result of encoding σ_i with the inner code.

The decoding algorithm succeeds if the decoding procedure of the outer code, which is executed in Step 4 of the algorithm, outputs the correct message. This happens if $\mathrm{ED}(\sigma^{(\mathsf{out})}, \tilde{\sigma}^{(\mathsf{out})}) \leq \delta_{\mathrm{out}} n$. To prove that this

holds with high probability, we classify the errors that can be introduced at each step of the algorithm and bound the probability that we get too many of them.

There are three error types that increase the edit distance between $\sigma^{(\mathsf{out})}$ and $\tilde{\sigma}^{(\mathsf{out})}$:

1. *Deleted buffer:* This happens when the channel deleted too many bits from a buffer so that less than $M_B m/2$ bits survived the channel, and we did not identify this buffer in Step 1 of the algorithm.

2. *Spurious buffer:* In this case the algorithm mistakenly identifies a buffer inside an inner codeword. This might happen if there are many consecutive runs of the symbol 1 that were deleted by the channel. As a result, a long run of the symbol 0 is created and the algorithm will mistakenly identify it as a buffer in Step 1.

3. *Wrong inner decoding:* Here the decoding of the inner code returns a different inner codeword. This error happens if the edit distance between an inner codeword $c_{\sigma_i}^{(\mathsf{in})}$ and the corresponding $\tilde{c}_j$ is larger[4] than $\delta_{\text{in}} m$.

In the following subsections, we analyze each error type separately and show that each happens with probability $\exp(-\Omega(m))$ per inner codeword. Our analysis of the first two error types is similar to [GL18], but our analysis of the third case is different.

Deleted Buffer

Proposition 5.5.2. *Let r_B be a buffer in Enc(x). The probability that the decoding algorithm fails to identify it as a buffer in Step 1 is at most* $\exp(-\Omega(m))$.

Proof. Recall that the length of a buffer is $\lceil M_B m/(1-p) \rceil$. Therefore the expected number of bits that survive the transmission through the BDC_p is at least $M_B m$. The decoder misses a buffer if the number of buffer bits that survived the transmission is smaller than $M_B m/2$. Let Z denote the random variable that corresponds to the number of bits that survived the transmission of r_B through the BDC_p. Clearly, $Z \sim \text{Bin}(\lceil M_B m/(1-p) \rceil, (1-p))$. By using the Chernoff bound given in Lemma 5.2.7, we get that this error happens with probability

$$\Pr\left[Z < \frac{M_B m}{2}\right] = \Pr\left[Z < \left(1 - \frac{1}{2}\right) M_B m\right] < \exp\left(-\frac{1}{8} M_B m\right) .$$

□

Spurious Buffer

Recall that this can happen if many consecutive runs of the symbol 1 were deleted by the channel, so a long run of the symbol 0 is created. If the length of this long run is longer than $M_B m/2$ then the decoder mistakenly identifies it as a buffer.

Proposition 5.5.3. *Let $c_{\sigma_i}^{(\mathsf{in})}$ be an inner codeword. Denote by* $\text{Blow}(c_{\sigma_i}^{(\mathsf{in})})$ *the string obtained by blowing up the runs in $c_{\sigma_i}^{(\mathsf{in})}$ according to the encoding procedure. The probability that the decoder in Step 1 identifies a buffer inside the string obtained by transmitting Blow$(c_{\sigma_i}^{(\mathsf{in})})$ through the BDC$_p$ is at most* $\exp(-\Omega(m))$.

Proof. We first compute the probability that a run is deleted. Recall that after encoding the message we transmit runs of length $\lceil M_1/(1-p) \rceil$ or $\lceil M_2/(1-p) \rceil$. The probability that all the bits from a run of length $\lceil M_1/(1-p) \rceil$ are deleted by the BDC_p is

$$p^{\lceil M_1/(1-p) \rceil} \leq p^{M_1/(1-p)} \leq e^{-M_1}.$$

[4]Note that it may be the case that due to decoding errors, the ith inner codeword was interpreted as the jth codeword by the decoder (e.g. if a buffer was deleted or a spurious buffer was introduced).

Equivalently, the probability that all the bits from a run of length $\lceil M_2/(1-p) \rceil$ are deleted by the BDC_p is

$$p^{\lceil M_2/(1-p) \rceil} \leq p^{M_2/(1-p)} \leq e^{-M_2}.$$

Suppose that ℓ consecutive runs of the bit 1 are deleted. We consider two cases.

First, consider the case where $\ell > mM_B/4M_2$. The probability that exactly ℓ runs of the symbol 1 are deleted is at most (the highest probability is obtained when all the ℓ runs are $\lceil M_1/(1-p) \rceil$-runs)

$$p^{\lceil M_1/(1-p) \rceil \ell} \leq \exp\left(-M_1 \ell\right) \leq \exp\left(-M_1 M_B m/4M_2\right) = \exp\left(-\Omega\left(m\right)\right).$$

The probability that there exist $\geq mM_B/(4M_2)$ consecutive runs of the symbol 1 that are deleted in a word of length m is at most $O(m^2) \cdot \exp\left(-\Omega\left(m\right)\right) = \exp(-\Omega(m))$ (we just need to pick the start and end point of the consecutive runs).

Now, if $\ell \leq mM_B/(4M_2)$ consecutive runs of 1's are deleted, then there are $\ell + 1$ runs of zeros that are merged to a single run. Suppose that all the merged runs were 2-runs (so that the length of the run of the symbol 0 that was created is maximized). Denote by Z the random variable that corresponds to the number of bits that survived the transmission of these $\ell+1$ runs. It holds that $Z \sim \mathrm{Bin}\left((\ell+1)\left\lceil M_2/\left(1-p\right)\right\rceil, 1-p\right)$ and

$$\begin{aligned}
\mathbb{E}\left[Z\right] &= (\ell+1)\left\lceil M_2/(1-p)\right\rceil(1-p) \\
&\leq (\ell+1)(M_2+1) \\
&\leq (mM_B/(4M_2)+1)(M_2+1) \\
&\overset{(*)}{\leq} \frac{M_B m + 4M_2}{3} \\
&< \frac{2}{5} M_B m
\end{aligned}$$

where inequality $(*)$ holds for $M_2 \geq 3$ and the last inequality holds for large enough[5] m. Thus, we get by the Chernoff bound that the probability that $Z \geq M_B m/2$ is

$$\Pr\left[Z \geq \frac{M_B m}{2}\right] = \Pr\left[Z \geq \left(1+\frac{1}{4}\right)\frac{2}{5} M_B m\right] \leq \exp\left(-\frac{1}{120} M_B m\right).$$

Hence, the probability that specific $\ell \leq mM_B/(4M_2)$ consecutive runs of the symbol 1 were deleted and a spurious buffer was created is at most $\exp(-M_B m/120)$. Therefore, for such an ℓ, the probability that there exists a spurious buffer in an inner codeword of length m is at most $m^2 \cdot \exp(-M_B m/120) \leq \exp(-M_B m/240)$, for large enough m. □

Remark 5.5.4. *Proposition 5.5.3 upper bounds the probability that a spurious buffer is identified in a inner codeword. However, it may be the case that the decoder identifies two or more spurious buffers inside a single inner codeword. This is not an issue as the maximal number of spurious buffer inside an inner codeword is $\leq 2/M_B$, and therefore the expected number of spurious buffers in an inner codeword is at most $(2/M_B) \cdot \exp(-\Omega(m)) = \exp(-\Omega(m))$.*

Wrong Inner Decoding

This is the most difficult case to analyze. The inner decoding procedure might output a wrong codeword when the edit distance between an inner codeword $c_{\sigma_i}^{(\mathsf{in})}$ and the corresponding word that was obtained at Step 2 of the algorithm, $\tilde{c}_j$, is larger than $\delta_{\mathrm{in}} m$. The next proposition shows that the probability of this event is exponentially small in m.

[5]Recall that by the way that we choose our parameters we pick m at the end so that we can make it as large a constant as we wish.

Proposition 5.5.5. *Assume the setting of Proposition 5.5.1. Let $c_{\sigma_i}^{(\mathsf{in})}$ be an inner codeword. Assume that the buffers before and after $c_{\sigma_i}^{(\mathsf{in})}$ were detected correctly and that there were no spurious buffers in between. Suppose that $\tilde{c}_j$ is the corresponding string obtained at Step 2 of the decoding algorithm on s_j. Then,*[6]

$$\Pr\left[\mathrm{ED}\left(c_{\sigma_i}^{(\mathsf{in})}, \tilde{c}_j\right) > \delta_{\mathrm{in}} m\right] \leq \exp(-\Omega(m)) \ .$$

We prove this claim in the remainder of this subsection, but first we give some intuition and introduce some important notions. Recall that a run r_j in an inner codeword is replaced with a run of length $N_1 = \lceil M_1/(1-p) \rceil$ or $N_2 = \lceil M_2/(1-p) \rceil$. Let Z_j be the random variable corresponding to the number of bits from this blown-up run that survived the transmission through the BDC_p. If $|r_j| = 1$ then, $Z_j \sim \mathrm{Bin}\left(\lceil M_1/(1-p) \rceil, 1-p\right)$. If $|r_j| = 2$ then, $Z_j \sim \mathrm{Bin}\left(\lceil M_2/(1-p) \rceil, 1-p\right)$. Intuitively, in Step 2 the algorithm reads every Z_j and decides according to the threshold T if Z_j corresponds to a run of length 1 or 2. However, it may be the case that, say, $Z_{j+1} = 0$ and then the algorithm will mistakenly base its decision according to the value of $Z_j + Z_{j+2}$, etc. For example, consider an initial string $\langle 00100 \rangle$. After the blow-up, we transmit the string $\langle 0^{N_2} 1^{N_1} 0^{N_2} \rangle$. Suppose that the middle run (the run consisting of the symbol 1) was deleted by the channel. The decoder then faces a long run of 0's and treats it as a single run and in particular, it will decode it as $\langle 0 \rangle$ or $\langle 00 \rangle$, or even as a spurious buffer. This motivates the following definitions.

Definition 5.5.6. *When $Z_j = 0$ we say that r_j was* deleted by the channel.

Remark 5.5.7. *We shall make a distinction between runs that were deleted by the channel and those that our algorithm "deleted" so whenever we refer to a deleted bit we will stress which process caused the deletion.*

Definition 5.5.8. *For every $j \in [\beta m]$, let b_j be the bit appearing in r_j. We denote*

$$r'_j = \begin{cases} \langle b_j b_j \rangle & \text{if } Z_j > T \\ \langle b_j \rangle & \text{if } 0 < Z_j \leq T \\ \langle \rangle & \text{if } Z_j = 0 \end{cases} \ .$$

In other words, r'_j is what Step 2 of our decoding algorithm would output when given Z_j as input. In particular, $\left|r'_j\right|$ can be $0, 1, 2$, depending on Z_j. Note that if $\left|r'_j\right| = 0$ then it means that the channel deleted the run.

For the next definition, we remind the reader that in our setting the total number of runs in an inner codeword (and hence also in a blown-up word) is $\beta_1 m + \beta_2 m = \beta m$.

Definition 5.5.9. *A set $I \subset [\beta m], |I| \geq 2$, is called a* maximal merged set *if the following conditions hold:*

1. *For every $i \in I$ it holds that $Z_i > 0$.*
2. *All the bits from I are merged into one run.*
3. *There is no set J such that $I \subsetneq J$ and the bits from J are merged into one run.*

For example, consider the following consecutive runs that were sent through the channel $\langle 0^{N_2} 1^{N_1} 0^{N_1} 1^{N_2} 0^{N_1} 1^{N_1} 0^{N_2} \rangle$. Suppose that the third run and the fifth run were deleted by the channel and the rest of the runs were not deleted by the channel. The maximal merged set corresponding to this deletion pattern is $I = \{2, 4, 6\}$.

Claim 5.5.10. *Let $I \subset [\beta m]$ be a maximal merged set. Denote $j = \min I$ and $k = \max I$. Then, all the runs $r_{j+1}, r_{j+3}, \ldots, r_{k-1}$ were* deleted by the channel.

Proof. Assume w.l.o.g. that r_j and r_k are runs of the symbol 0. For every $i \in \{j+1, j+3, \ldots, k-1\}$, r_i is a run of symbol 1 and must be deleted by the channel. Otherwise, I will not be a merged set. □

[6] Recall that we use a different index j to indicate that it may be the case that spurious buffers were found earlier, in some other inner codeword, or that some earlier buffers were mistakenly deleted.

Definition 5.5.11. *Let $I \subset [\beta m]$ be a maximal merged set and set $j = \min I$. We denote by $\tilde{r}_j$ the result of Step 2 of our decoding algorithm on this merged run.*

Remark 5.5.12. *It is important to remember that r_j is the original run, r'_j is what the algorithm would return when given Z_j as input, and $\tilde{r}_j$ is what the algorithm actually returns when reading the bits of the merged run.*

We can now see that some bits that survived the channel were deleted by our algorithm as it failed to realize that they came from different runs. This is captured by the next definition.

Definition 5.5.13. *Let $I \subset [\beta m]$ be a maximal merged set. We say that the* decoding algorithm deleted $\left|r'_j\right| - |\tilde{r}_j| + \sum_{i \in I \setminus \{j\}} |r_i|$ *bits in the set I.*

As $\left|r'_j\right| \leq |\tilde{r}_j|$ the following claim is obvious.

Claim 5.5.14. *Let I be a maximal merged set and set $j = \min I$. The number of bits deleted by the decoding algorithm in the merged set I is at most $\sum_{i \in I \setminus \{j\}} |r_i|$.*

We next extend Claim 5.5.14 and bound the total number of bits that our algorithm deletes in an inner codeword. We assume that the buffers before and after the word were correctly identified by the decoding algorithm in Step 1.

Claim 5.5.15. *Let $D \subset [\beta m]$ be the indices of the runs that were deleted by the channel. If the last run was not deleted, i.e., $\beta m \notin D$, then the number of bits that were deleted by the decoding algorithm is at most $\sum_{i \in D} |r_{i+1}|$.*

If the last run was deleted by the channel, i.e., $\beta m \in D$, then the number of bits deleted by the algorithm is at most $\sum_{i \in D \setminus \{\beta m\}} |r_{i+1}| + 2$.

Proof. We first deal with the case where some runs were merged with the bits in the buffers (before or after the word). This happens if the first or the last run were deleted by the channel. If $1 \in D$ then let $r_{i'}$ to be the first run of the symbol 1 that was not deleted by the channel. Then, all runs of the symbol 0 before $r_{i'}$ were merged to the left buffer. Therefore, $D_L := \{1, 3, \ldots, i' - 2\} \subseteq D$ and the decoding algorithm deleted exactly $|r_2| + \ldots + |r_{i'-1}| = \sum_{\ell \in D_L} |r_{\ell+1}|$ bits (since all these runs were considered as part of the buffer).

Similarly, if $\beta m \in D$ define $r_{i'}$ to be the last run of the symbol 1 that was not deleted by the channel. In this case all the runs of 0's after $r_{i'}$ were merged to the right buffer. In this case, $D_R := \{i' + 2, i' + 4, \ldots, \beta m\} \subseteq D$ and the decoding algorithm deleted exactly $|r_{i'+1}| + \ldots + |r_{\beta m - 1}| \leq 2 + \sum_{\ell \in D_R \setminus \{\beta m\}} |r_{\ell+1}|$ bits.

We now account for inner deletions (i.e., those that did not cause runs to merge with buffers). These deletions may generate what we called maximal merged sets. Let $I_1, \ldots, I_t$ be all maximal merged sets, excluding those that were merged with buffers. Denote $j_i = \min I_i$ and $k_i = \max I_i$ and let[7] $D_i := D \cap [j_i + 1, k_i - 1]$ for $i \in [t]$.

According to Claim 5.5.10 it holds that $\{j_i + 1, j_i + 3, \ldots, k_i - 1\} \subseteq D_i$. Thus, $I_i \subseteq \{j_i, j_i + 2, \ldots, k_i\}$. Claim 5.5.14, implies that the number of bits deleted by the algorithm in I_i is at most $\sum_{\ell \in I_i \setminus \{j_i\}} |r_\ell|$. Thus, the total number of bits deleted by the algorithm, excluding those bits from $D_L \cup D_R$, is bounded from above by

$$\sum_{i=1}^{t} \sum_{\ell \in I_i \setminus \{j_i\}} |r_\ell| \leq \sum_{i=1}^{t} \sum_{\ell \in D_i} |r_{\ell+1}| \leq \sum_{i \in D \setminus (D_L \cup D_R)} |r_{i+1}| \,.$$

Taking into account the deleted bits from $D_L \cup D_R$ the claim follows. □

We now use concentration bounds to argue about the expected number of bits that were deleted and the effect on the edit distance between the original inner codeword and the one returned by the algorithm in Step 2.

We first study the probability that $r_j \neq r'_j$ (recall Definition 5.5.8):

[7] Note that it may be the case that the set $D' := D \setminus (\cup_i D_i)$ is not the empty set. In this case the indices in D' correspond to runs that were deleted by the channel but did not cause a merge. E.g., if two consecutive runs are deleted by the channel and the runs before and after were not deleted, then this does not make our algorithm to delete additional bits.

1. If $|r_j| = 1$ then there are two possible types of errors:

   (a) $\left|r'_j\right| = 2$: We denote the probability for this to happen by

   $$P^{(1)\to(2)} := \Pr[Z_j \geq T+1] \,.$$

   We next give two estimates of this probability, one is an exact calculation and the other is an upper bound. Direct calculation gives

   $$P^{(1)\to(2)} = \Pr[Z_j \geq T+1] = \sum_{i=T+1}^{\lceil \frac{M_1}{1-p} \rceil} \binom{\lceil \frac{M_1}{1-p} \rceil}{i} (1-p)^i \cdot p^{\lceil \frac{M_1}{1-p} \rceil - i} \,. \tag{5.4}$$

   We next would like to use the Poisson distribution to give a simpler bound. For this we would like to use Theorems 5.2.5 and 5.2.6.

   Lemma 5.5.16. *Let $q \geq 1-p$ and $T \geq M_1 + q$. It holds that*

   $$P^{(1)\to(2)} \leq 1 - e^{-M_1 - q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!} \tag{5.5}$$

   Moreover, the function $f(q) := 1 - e^{-M_1-q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!}$ is monotonically increasing in q.

   Proof. Define $Y(j,x) \sim \mathrm{Bin}(j, (M_1+x)/j)$. Observe that $\mathbb{E}[Y(j,x)] = M_1 + x$. Denote $n' = \lceil M_1/(1-p) \rceil$. First note that

   $$\Pr[Z_j \geq T+1] \leq \Pr[Y(n', 1-p) \geq T+1]$$

   since the expectation of $Y(n',(1-p))$ is $M_1+(1-p)$ whereas the expectation of Z_j is $\leq M_1+(1-p)$ and they are both binomial distributions on n' trials. By the same reasoning we have that for every $j \geq n'$

   $$\Pr[Y(j,(1-p)) \geq T+1] \leq \Pr[Y(j,q) \geq T+1] \,.$$

   Let $P(x) \sim \mathrm{Poisson}(x)$. Theorem 5.2.5 implies that $\lim_{j\to\infty} Y(j,x) = P(M_1+x)$. Therefore,

   $$\begin{aligned} P^{(1)\to(2)} = \Pr[Z_j \geq T+1] &\leq \Pr[Y(n',q) \geq T+1] \\ &= 1 - \Pr[Y(n',q) \leq T] \\ &\leq 1 - \lim_{j\to\infty} \Pr[Y(j,q) \leq T] \\ &= 1 - \Pr[P(M_1+q) \leq T] \\ &= 1 - e^{-M_1-q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!} \,, \end{aligned}$$

   where the second inequality follows from Theorem 5.2.6 due to monotonicity for $T \geq M_1 + q$. Note that the monotonicity of $f(q) = 1 - e^{-M_1-q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!}$ follows from Lemma 5.2.4. □

   (b) $\left|r'_j\right| = 0$: Here the blown-up run was completely deleted by the channel. The probability for this to happen is $P^{(1)\to(0)} := \Pr[Z_j = 0]$. It holds that,

   $$P^{(1)\to(0)} = \Pr[Z_j = 0] = p^{\lceil \frac{M_1}{1-p} \rceil}. \tag{5.6}$$

   It also holds that for any $p \in (0,1)$,

   $$P^{(1)\to(0)} = \Pr[Z_j = 0] \leq e^{-M_1}. \tag{5.7}$$

2. Similarly, when $|r_j| = 2$ there are two cases to consider:

   (a) $|r'_j| = 1$: The probability for this to happen is $P^{(2)\to(1)} := \Pr[Z_j \leq T]$. As before, the exact calculation is

   $$P^{(2)\to(1)} = \Pr[Z_j \leq T] = \sum_{i=0}^{T} \binom{\left\lceil \frac{M_2}{1-p} \right\rceil}{i} (1-p)^i \cdot p^{\left\lceil \frac{M_2}{1-p} \right\rceil - i} . \tag{5.8}$$

   Similarly to the calculations for $P^{(1)\to(2)}$, we would like to upper bound $P^{(2)\to(1)}$ using a simpler expression coming from the Poisson distribution.

   Lemma 5.5.17. *For every p and for $T \leq M_2 - 1$, it holds that*

   $$P^{(2)\to(1)} \leq e^{-M_2} \sum_{i=0}^{T} \frac{M_2^i}{i!} . \tag{5.9}$$

   Moreover, for every $q \geq p$ such that $M_2/(1-q)$ is an integer, it holds that

   $$P^{(2)\to(1)} \leq \sum_{i=0}^{T} \binom{\frac{M_2}{1-q}}{i} (1-q)^i \cdot q^{\frac{M_2}{1-q} - i} \tag{5.10}$$

   Proof. For a natural number $1 \leq i$, let $Y(i) \sim \mathrm{Bin}\,(i, M_2/i)$. Let $P \sim \mathrm{Poisson}(M_2)$. Observe that $\Pr[Z_j \leq T] \leq \Pr[Y(\lceil M_2/(1-p) \rceil) \leq T]$ as the latter can only have smaller expectation. Since $\lim_{i\to\infty} Y(i) \sim P$ and due to the monotonicity implied by Theorem 5.2.6 we get that when $T \leq M_2 - 1$, for every p it holds that:

   $$\begin{aligned} P^{(2)\to(1)} = \Pr[Z_j \leq T] &\leq \Pr[Y(\lceil M_2/(1-p) \rceil) \leq T] \\ &\leq \Pr[Y(\lceil M_2/(1-q) \rceil) \leq T] \\ &= \Pr[Y(M_2/(1-q)) \leq T] \\ &\leq \lim_{i\to\infty} \Pr[Y(i) \leq T] \\ &= \Pr[P \leq T] \\ &= e^{-M_2} \sum_{i=0}^{T} \frac{M_2^i}{i!} , \end{aligned}$$

   where the second and the third inequalities hold due to Theorem 5.2.6 for $T \leq M_2 - 1$. Note that the second inequality proves the second statement in the lemma. □

   (b) $|r'_j| = 0$: The probability for this to happen is $P^{(2)\to(0)} := \Pr[Z_j = 0]$. It holds that,

   $$P^{(2)\to(0)} = \Pr[Z_j = 0] = p^{\left\lceil \frac{M_2}{1-p} \right\rceil} \tag{5.11}$$

   and for every $p \in (0,1)$ we have

   $$P^{(2)\to(0)} = \Pr[Z_j = 0] \leq e^{-M_2} . \tag{5.12}$$

Recall that $c^{(\mathsf{in})}_{\sigma_i}$ is an inner codeword that consists of exactly $\beta_1 m$ 1-runs and $\beta_2 m$ 2-runs. Also recall that we blow-up an inner codeword, $c^{(\mathsf{in})}_{\sigma_i}$, and send it through the BDC_p. Suppose that Step 1 of the algorithm identified the $i-1$'th and the i'th buffer and that there were no spurious buffers in between. Let s_j be the binary string corresponding to this decoding window obtained in Step 1, and let $\tilde{c}_j$ be the result of Step 2 of the algorithm on s_j.

For every $j \in [\beta m - 1]$, Let X_j be the random variable defined by

$$X_j = \begin{cases} 0 & \text{if } |r_j| = \left|r'_j\right| \\ 1 & \text{if } Z_j > 0 \text{ and } |r_j| \neq \left|r'_j\right| \\ |r_j| + |r_{j+1}| & \text{if } Z_j = 0 \end{cases} .$$

Similarly define $X_{\beta m}$ to be

$$X_{\beta m} = \begin{cases} 0 & \text{if } |r_{\beta m}| = \left|r'_{\beta m}\right| \\ 1 & \text{if } Z_{\beta m} > 0 \text{ and } |r_{\beta m}| \neq \left|r'_{\beta m}\right| \\ |r_{\beta m}| + 2 & \text{if } Z_{\beta m} = 0 \end{cases} .$$

Claim 5.5.18. *Let $c^{(\text{in})}_{\sigma_i}$ be an inner codeword. Assume that the buffers before and after $c^{(\text{in})}_{\sigma_i}$ were detected correctly and assume that there were no spurious buffers in between. Suppose that $\tilde{c}_j$ is the corresponding string obtained at Step 2 of the decoding algorithm on s_j. Then,*

$$\text{ED}\left(c^{(\text{in})}_{\sigma_i}, \tilde{c}_j\right) \leq \sum_{j=1}^{\beta m} X_j .$$

Proof. If r_j is a 1-run and r'_j is a 2-run then there was an insertion. Equivalently, if r_j is a 2-run and r'_j is a 1-run then there was a deletion. If a run was completely deleted by the channel then according to Claim 5.5.15, at the worst case scenario, the following run is also deleted by the algorithm. The definition of the X_j's accounts for all that. □

Note that we may do over counting in some scenarios, e.g., if $r_{j+1} \neq r'_{j+1}$ and r_j was deleted by the channel then $X_j + X_{j+1} = |r_j| + |r_{j+1}| + 1$ but the edit distance is at most $|r_j| + |r_{j+1}|$. This over counting makes the upper bound less tight.

Set $X = \sum_{j=1}^{\beta m} X_j$. We next upper bound and lower bound $\mathbb{E}[X]$.

Claim 5.5.19. *It holds that*

$$\mathbb{E}[X] \geq \xi m ,$$

where

$$\xi = \beta_1 \left(P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)}\right) + \beta_2 \left(P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)}\right) .$$

Proof. For every X_j such that r_j is a 1-run we have

$$\mathbb{E}[X_j] \geq 1 \cdot P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)} ,$$

where we used the fact that $|r_j| + |r_{j+1}| \geq 2$. Similarly, for every X_j such that r_j is a 2-run we have

$$\mathbb{E}[X_j] \geq 1 \cdot P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)} .$$

As there are exactly β_1 1-runs and β_2 2-runs, the claim follows. □

Claim 5.5.20. *It holds that*

$$\mathbb{E}[X] \leq \gamma m + P^{(1)\to(0)} ,$$

where,

$$\gamma = \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2) \cdot P^{(1)\to(0)} + 4\beta_2 \cdot P^{(2)\to(0)} , \tag{5.13}$$

is the same γ as in Proposition 5.5.1.

For the proof we shall denote with $X_j^{i,k}$ the random variable X_j when r_j is an i-run and r_{j+1} is a k-run.

Proof. Suppose that $r_{\beta m}$ is a 1-run. As will be explained later, this is the worst case, i.e., the upper bound that we prove on $\mathbb{E}[X]$ is largest in this case. Denote by $Y^{i,k} \subseteq [\beta m - 1]$ the set of indices $j \in [\beta m - 1]$ such that r_j is an i-run and r_{j+1} is a k-run. From linearity of expectation it follows that

$$\mathbb{E}[X] = \sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] + \mathbb{E}\left[X_{\beta m}\right] .$$

Let λ_1 be such that $\left|Y^{1,2}\right| = \lambda_1 m$. Thus, $\left|Y^{1,1}\right| = (\beta_1 - \lambda_1)m - 1$ (where the 1 is subtracted because of the last run, which we assumed is a 1-run). By definition we have that

$$\sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] = \left(1 \cdot P^{(1)\to(2)} + 3 \cdot P^{(1)\to(0)}\right) \cdot \lambda_1 m$$

and

$$\sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] = \left(1 \cdot P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)}\right) \cdot ((\beta_1 - \lambda_1)m - 1) .$$

Observe that

$$\begin{aligned}
&\sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \mathbb{E}[X_{\beta m}] \\
= \;& \left(1 \cdot P^{(1)\to(2)} + 3 \cdot P^{(1)\to(0)}\right) \cdot \lambda_1 m + \left(1 \cdot P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)}\right) \cdot ((\beta_1 - \lambda_1)m - 1) \\
& + \left(1 \cdot P^{(1)\to(2)} + 3 \cdot P^{(1)\to(0)}\right) \\
= \;& P^{(1)\to(2)} \cdot \beta_1 m + P^{(1)\to(0)} \cdot (2\beta_1 m + \lambda_1 m + 1) .
\end{aligned}$$

As there are exactly $\beta_2 m$ 2-runs, it holds that $0 \le \lambda_1 \le \beta_2$. Hence, this sum is maximized for $\lambda_1 = \beta_2$. We thus have that

$$\begin{aligned}
&\sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \mathbb{E}[X_{\beta m}] \\
\le\; & \beta_1 m P^{(1)\to(2)} + (2\beta_1 + \beta_2) m P^{(1)\to(0)} + P^{(1)\to(0)} .
\end{aligned} \tag{5.14}$$

Similarly, let λ_2 be such that $\left|Y^{2,1}\right| = \lambda_2 m$. Thus, $\left|Y^{2,2}\right| = (\beta_2 - \lambda_2)m$. It holds that

$$\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] = \left(1 \cdot P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)}\right) \cdot \lambda_2 m$$

and

$$\sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] = \left(1 \cdot P^{(2)\to(1)} + 4 \cdot P^{(2)\to(0)}\right) \cdot (\beta_2 - \lambda_2) m .$$

Since the sum $\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right]$ is maximized for $\lambda_2 = 0$ we get,

$$\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] \le \beta_2 m P^{(2)\to(1)} + 4\beta_2 m P^{(2)\to(0)} . \tag{5.15}$$

Combining (5.14) and (5.15) we obtain

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \mathbb{E}[X_{\beta m}] + \sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] \\
&\le \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2) \cdot P^{(1)\to(0)} + 4\beta_2 \cdot P^{(2)\to(0)} + P^{(1)\to(0)} \\
&= \gamma m + P^{(1)\to(0)} ,
\end{aligned} \tag{5.16}$$

as claimed.

Note that if $r_{\beta m}$ was a 2-run, then $\left|Y^{1,2}\right| + \left|Y^{1,1}\right| = \beta_1 m$ (no need to subtract 1 since the last run is now a 2-run) and we have,

$$\sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] \leq \beta_1 m P^{(1)\to(2)} + (2\beta_1 + \beta_2) m P^{(1)\to(0)} \ .$$

In this case, we have that $\left|Y^{2,1}\right| + \left|Y^{2,2}\right| = \beta_2 m - 1$. Thus, if we let λ_2 be such that $\left|Y^{2,1}\right| = \lambda_2 m$ and $\left|Y^{2,2}\right| = (\beta_2 - \lambda_2)m - 1$ then

$$\begin{aligned}
&\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] + \mathbb{E}[X_{\beta m}] \\
= \ & \left(1 \cdot P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)}\right) \cdot \lambda_2 m + \left(1 \cdot P^{(2)\to(1)} + 4 \cdot P^{(2)\to(0)}\right) \cdot ((\beta_2 - \lambda_2)m - 1) \\
& + \left(1 \cdot P^{(2)\to(1)} + 4 \cdot P^{(2)\to(0)}\right) \\
= \ & P^{(2)\to(1)} \cdot \beta_2 m + P^{(2)\to(0)} \cdot (4\beta_2 m - \lambda_2 m) \ ,
\end{aligned}$$

and this sum is maximized for $\lambda_2 = 0$. We thus have that

$$\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] + \mathbb{E}[X_{\beta m}] \leq P^{(2)\to(1)} \beta_2 m + 4P^{(2)\to(0)} \beta_2 m$$

Then, if $r_{\beta m}$ is a 2-run we have

$$\begin{aligned}
\mathbb{E}[X] &\leq \beta_1 m P^{(1)\to(2)} + (2\beta_1 + \beta_2) m P^{(1)\to(0)} + P^{(2)\to(1)} \beta_2 m + 4P^{(2)\to(0)} \beta_2 m \\
&= \gamma m < \gamma m + P^{(1)\to(0)} \ .
\end{aligned}$$

□

Thus, for any constant $\gamma' > \gamma$ there exist a constant $M_{\gamma'}$ such that for all $m > M_{\gamma'}$ it holds that

$$\mathbb{E}[X] \leq \gamma m + P^{(1)\to(0)} < \gamma' m \ .$$

In the following claim we use concentration bound to show that the probability that X is greater than $\gamma' m$, for $\gamma' > \gamma$, is exponentially small in m and then we conclude that decoding of an inner codeword succeeds with high probability.

Claim 5.5.21. *For any $\gamma' > \gamma$ and for every constant $\nu > 0$ it holds that for a large enough m,*

$$\Pr[X > (1+\nu)\gamma' m] < \exp\left(-\frac{\nu^2 \xi^2 m}{8\beta}\right) = \exp(-\Omega(m)) \ ,$$

where ξ is as in Claim 5.5.19.

Proof. First note that

$$\Pr[X > (1+\nu)\gamma' m] \leq \Pr[X > (1+\nu)\mathbb{E}[X]] \ ,$$

where by Claim 5.5.20 the inequality holds for large enough m. The delicate point is to notice that the X_j's are independent. This is because each X_j is determined solely according to the value of Z_j (indeed, its value only depends on whether $Z_j = 0$, $Z_j \leq T$ or $Z_j > T$), and the random variables Z_j's are independent by the definition of the binary deletion channel. For every X_j it holds that $0 \leq X_j \leq 4$ and if we set $t = \nu\mathbb{E}[X]$ and apply Theorem 5.2.9 then we get that

$$\begin{aligned}
\Pr\left[X > (1+\nu)\mathbb{E}[X]\right] &< \exp\left(-\frac{2\nu^2(\mathbb{E}[X])^2}{\beta m \cdot 4^2}\right) \\
&\leq \exp\left(-\frac{2\nu^2(\xi m)^2}{16\beta m}\right) \\
&= \exp\left(-\frac{\nu^2 \xi^2 m}{8\beta}\right) \ ,
\end{aligned}$$

where the second inequality follow from Claim 5.5.19.

□

We are now ready to prove the main claim of this subsection, Proposition 5.5.5.

Proof of Proposition 5.5.5. By Claim 5.5.18 X is an upper bound on $\mathrm{ED}(c_{\sigma_i}^{(\mathrm{in})}, \tilde{c}_j)$. Thus,

$$\Pr\left[\mathrm{ED}\left(c_{\sigma_i}^{(\mathrm{in})}, \tilde{c}_j\right) > \delta_{\mathrm{in}} m\right] \leq \Pr[X > \delta_{\mathrm{in}} m] \ .$$

By the assumption in Proposition 5.5.1 we have that $\delta_{\mathrm{in}} > \gamma$. We thus get that

$$\Pr\left[X > \delta_{\mathrm{in}} m\right] = \Pr\left[X > \left(1 + \frac{\delta_{\mathrm{in}} - \gamma}{\delta_{\mathrm{in}} + \gamma}\right) \frac{\delta_{\mathrm{in}} + \gamma}{2} m\right] \leq \exp\left(-\left(\frac{\delta_{\mathrm{in}} - \gamma}{\delta_{\mathrm{in}} + \gamma}\right)^2 \frac{\xi^2}{8\beta} m\right) \ ,$$

where the last inequality follows from Claim 5.5.21 by plugging $\nu = \frac{\delta_{\mathrm{in}} - \gamma}{\delta_{\mathrm{in}} + \gamma}$ and $\gamma' = \frac{\delta_{\mathrm{in}} + \gamma}{2}$. This completes the proof of Proposition 5.5.5. □

Remark 5.5.22. *Observe that all the parameters involved in the upper bound in Proposition 5.5.5, namely, γ, ξ, β are independent of m. That is, they only depend on $\delta_{\mathrm{in}}, M_1, M_2, T$ and β_1.*

We are now ready to prove Proposition 5.5.1.

Proof of Proposition 5.5.1. We would like to show that with high probability, the edit distance between the original outer codeword $\sigma^{(\mathrm{out})}$ and the string $\tilde{\sigma}^{(\mathrm{out})}$, obtained after Step 3 of the decoding algorithm, is smaller than $\delta_{\mathrm{out}} n$. To prove this we shall analyze the contribution of each of the error types (deleted buffer, spurious buffer and wrong inner decoding) on the edit distance.

A deleted buffer causes two inner codewords to merge and thus be decoded incorrectly by the inner code's decoding algorithm. When considering the effect of this on the edit distance between $\sigma^{(\mathrm{out})}$ and $\tilde{\sigma}^{(\mathrm{out})}$, this introduces two deletions and one insertion. Similarly, a single spurious buffer introduces one deletion and two insertions, since an inner codeword is split into two parts. Likewise, ℓ spurious buffers inside an inner codeword introduce 1 deletion and possibly $\ell + 1$ insertions. A wrong inner decoding causes just one deletion and one insertion. Therefore, every error type increases the edit distance between the original outer codeword $\sigma^{(\mathrm{out})}$ and $\tilde{\sigma}^{(\mathrm{out})}$ by at most three.

As mentioned, the outer decoding algorithm fails if $\mathrm{ED}\left(c^{(\mathsf{out})}, \tilde{c}^{(\mathsf{out})}\right) > \delta_{\mathrm{out}} n$. Thus, for this to happen, at least one of the following bad events must occur:

1. There were at least $\delta_{\mathrm{out}} n/9$ deleted buffers.
2. There were at least $\delta_{\mathrm{out}} n/9$ spurious buffers.
3. There were at least $\delta_{\mathrm{out}} n/9$ inner codewords that were decoded incorrectly even though they did not have spurious buffers and their buffers were identified.

We first treat events (1) and (3). We saw in Propositions 5.5.2 and 5.5.5 that for every inner codeword, each error type happens with probability $\exp(-\Omega(m))$. Since δ_{out} is a fixed constant, there exists a large enough m so that $\exp(-\Omega(m)) \leq \delta_{\mathrm{out}}/10$ for each error type. An important observation is that, similarly to Remark 5.5.22, the constants in the $\exp(-\Omega(m))$ in the different propositions depend only on $\delta_{\mathrm{in}}, \beta_1, M_1, T, M_2, M_B$ which are fixed constants and are not related to the outer code. Thus, we can choose a small enough $\varepsilon_{\mathrm{out}}$, which determines a large enough m, so that the probability for each error type is $\leq \delta_{\mathrm{out}}/10$. By the Chernoff bound given in Lemma 5.2.7, for a large enough n, each of the two bad events happens with probability $\exp(-\Omega(n))$.

We now turn to event (2). By Proposition 5.5.3 and Remark 5.5.4, the number of spurious buffers in every inner codewrod is between 0 and $2/M_B$ and the expected number of spurious buffers in an inner codeword is $\exp(-\Omega(m))$. Again, we can choose a small enough $\varepsilon_{\mathrm{out}}$, which determines a large enough m, so that the expected number of spurious buffers in an inner codeword is $\leq \delta_{\mathrm{out}}/10$. By the independence of the BDC_p, we can apply the Hoeffding bound (Theorem 5.2.9) and get that the probability that there are more than $\delta_{\mathrm{out}} n/9$ spurious buffers is $\exp(-\Omega(n))$.

In conclusion, Algorithm 5 succeeds with probability $1 - \exp(-\Omega(n))$ as claimed. □

5.5.2 Proof of Theorem 5.1.1

We now prove our main theorem.

Proof of Theorem 5.1.1. Our goal is to maximize the rate given in Equation (5.1) while assuring that the parameters that we pick guarantee successful decoding with high probability. Recall that the order by which we choose the parameters in our construction is the following. First, we choose $M_1, T, M_2, \beta_1, M_B, \delta_{\text{out}}$ to be fixed constants. Then, we compute upper bounds on $P^{(1)\to(2)}$, $P^{(1)\to(0)}$, $P^{(2)\to(1)}$, $P^{(2)\to(0)}$. Plugging these upper bounds to Equation (5.3), we get an upper bound[8] on γ which we denote by $\tilde{\gamma}$. Note that $\tilde{\gamma}$ depends only on M_1, T, M_2, β_1, and p. Then we choose δ_{in} to be larger than $\tilde{\gamma}$, and in particular we have $\gamma \leq \tilde{\gamma} < \delta_{\text{in}}$. Proposition 5.5.1 guarantees that if we choose a small enough ε_{out}, then our decoding algorithm will succeed with high probability. Thus, we only have to make sure that the rate that we get satisfies the statement in the theorem. We calculate the value of $\mathcal{R}_{\text{in}}$ using Proposition 5.3.8 and then use it to calculate the overall rate.[9]

We consider several regimes of p and for each regime we choose suitable parameters.

Case $p \geq 0.9$: In this case we choose:

$$M_1 = 5.41, M_2 = 22.8, \beta_1 = 0.522, M_B = 10^{-5}, \delta_{\text{out}} = 2^{-20} \quad \text{and } \delta_{\text{in}} = 0.01052 \,,$$

and set $T = 12$. From Proposition 5.3.8 we get that, for our choice of parameters, the rate of the inner code is $\mathcal{R}_{\text{in}} = 0.5229$. The upper bounds for $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ are computed using Equations (5.7), (5.9), and (5.12). To upper bound $P^{(1)\to(2)}$, we use Equation (5.5) given in Lemma 5.5.16 with $q = 0.1$. Observe that as we assume $p \geq 0.9$ it follows that $q \geq 1 - p$.

One can plug in the upper bounds to Equation (5.3) and observe that $\tilde{\gamma} < \delta_{\text{in}}$. Proposition 5.5.1 guarantees that for a small enough ε_{out} our decoding algorithm succeeds with high probability. To calculate the rate we use Equation (5.1). For a large enough m (e.g. $m > 10^5$) we obtain

$$\frac{0.5229(1-p)}{8.27323 + 0.761(1-p) + (1-p)/m} \geq \frac{0.5229(1-p)}{8.34933} > \frac{(1-p)}{16} \,.$$

Case $0.57 < p < 0.9$: For this regime we use the parameters

$$M_1 = 5.59, M_2 = 23.5, \beta_1 = 0.53, M_B = 10^{-5}, \delta_{\text{out}} = 2^{-20} \quad \text{and } \delta_{\text{in}} = 0.008013 \,,$$

and set $T = 13$. We get that the rate of the inner code is $\mathcal{R}_{\text{in}} = 0.55224$. We first note that the calculations used to upper bound $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ were obtained by using Equations (5.6), (5.8) and (5.11) with $p = 0.9$. This can be done since Equations (5.6) and (5.11) are clearly monotonically increasing in p and we are considering smaller values of p. Also, observe that since $M_2/(1 - 0.9) = 235$ is an integer, then by Equation (5.10) given in Lemma 5.5.17, for every $p \leq 0.9$,

$$P^{(2)\to(1)} \leq \sum_{i=0}^{T} \binom{\frac{M_2}{1-0.9}}{i} (1-0.9)^i \cdot (0.9)^{\frac{M_2}{1-0.9}-i} \,,$$

which is exactly what we get from Equation (5.8) with $p = 0.9$. Now, to upper bound $P^{(1)\to(2)}$ we use Equation (5.5) with $q = 1 - 0.57$, which is fine as $p > 0.57$ and thus $q > 1 - p$. As before, calculations show that $\tilde{\gamma} < \delta_{\text{in}}$. Hence for a small enough ε_{out} our decoding algorithm succeeds with high probability by Proposition 5.5.1. Plugging the parameters into Equation (5.1) and letting m be large enough we get

$$\frac{0.55224(1-p)}{8.48521 + 0.765(1-p) + (1-p)/m} > \frac{0.55224(1-p)}{8.81416} > \frac{1-p}{16} \,.$$

[8] We do not compute the value of γ exactly as it is too difficult to do parametrically.

[9] When applying Proposition 5.3.8, we set $\delta = \delta_{\text{in}}$.

Case $0 < p \leq 0.57$: The parameters we choose for this regime are

$$M_1 = 5.59, M_2 = 20.21, \beta_1 = 0.53, M_B = 10^{-5}, \delta_{\text{out}} = 2^{-20} \quad \text{and } \delta_{\text{in}} = 0.006147 \ ,$$

and set $T = 13$. Using Proposition 5.3.8, we get that $\mathcal{R}_{\text{in}} = 0.577475$. As in the previous case, the upper bounds to $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ were obtained by using Equations (5.6), (5.8) and (5.11), this time with $p = 0.57$ (observe that $M_2/(1-0.57)$ is an integer). In this case $13 = T \geq \lceil M_1/(1-p) \rceil$. For a random variable Z distributed as $Z \sim \text{Bin}\left(\lceil M_1/(1-p) \rceil, 1-p\right)$, it holds that

$$P^{(1)\to(2)} = \Pr[Z \geq T+1] = 0$$

since bits can only be deleted by the BDC_p.

One can simply verify that $\tilde{\gamma} < \delta_{\text{in}}$ and hence for a small enough ε_{out} our decoding algorithm succeeds with high probability by Proposition 5.5.1. Plugging the parameters into Equation (5.1) and letting m be large enough we get that for the case $p \leq 0.57$, the rate of the construction is

$$\frac{0.57747(1-p)}{7.71206 + 0.765(1-p) + (1-p)/m} > \frac{0.57747(1-p)}{8.47706} > \frac{1-p}{16} \ .$$

This completes the proof of Theorem 5.1.1 □

5.6 Rates For Fixed Values of Deletion Probabilities

In Theorem 5.1.1 we constructed codes of rate larger than $(1-p)/16$ for the BDC_p that can be used for reliable communication. Note that even if $p \to 1$ our construction gives codes of positive rate. Now, we wish to fix p (and thus leave the regime $p \to 1$) and instead of using the bounds given in Equations (5.5), (5.7), (5.9) and (5.12), we can use the exact direct calculations given in Equations (5.4), (5.6), (5.8) and (5.11), respectively. Using the exact bounds we can improve, for any fixed value of p, the rate of the code compared to what we obtained in Theorem 5.1.1. The reason that we can improve the bound is that in the proof of Theorem 5.1.1 we looked for a relatively simple argument that should work for every value of p. When p is fixed, we can use more direct calculations to get a better bound. For example, we can get significant improvement by using Equation (5.6) instead of Equation (5.7). E.g., for $p = 0.8$ there is a relatively large difference between $p^{M_1/(1-p)}$ and e^{-M_1}. E.g., for $M_1 = 5$ we have that $e^{-5} = 0.00673$ and $0.8^{5/(0.2)} = 0.00377$. Such savings allow us to choose smaller value of M_1 for the case $p = 0.8$. Then, by reducing the value of M_1 we reduce also the values of T and M_2 which eventually lead to an improved rate.

The reason that we do not optimize the calculation using these equations for every p is that the optimization involves complex expressions involving all our parameters and it is not clear how to optimize it and get a closed formula for the rate for arbitrary p.

In [DM07], the authors gave constructions of probabilistic codes for the binary deletion channel. They derived lower bounds on the capacity of the BDC_p that are the best lower bounds as far as we know for fixed values of p.

In Table 5.1 we compare our results to the ones obtained in [DM07]. One can see that our rates are smaller by approximately a factor of 2. Yet, the construction presented in this chapter is deterministic, has polynomial time complexity and has a simpler analysis.

Note that as p tends to 1 the rate that we achieve approaches $(1-p)/15.7$ as can be seen in Figure 5.3.

Remark 5.6.1. *The calculations in Table 5.1 are done only for $p \geq 0.5$ as the focus of this chapter is on large values of p. When p is small we know that the rate is better than $(1-p)/9$ (it approaches $1 - h(p)$) and one has to take a different approach in order to obtain good constructions.*

Remark 5.6.2. *As noted in the introduction of this chapter, the work by Rubinstein [Rub22] improves upon this result by showing how one can convert any code (even non explicit) for the BDC into an explicit and efficient code for the BDC with a negligible lost to the rate. Rubinstein achieves this assuming no structure on the inner code (note that the structure of our inner code clearly affects the final rate of our construction).*

p	$(\beta_1, N_1, T, N_2, \mathcal{R}_{\text{in}}, \delta_{\text{in}})$	Final rate	[DM07]
0.50	(0.497, 8, 7, 27, 0.5456, 0.00922)	0.050682	0.10186
0.55	(0.519, 9, 8, 34, 0.5525, 0.00825)	0.043005	0.084323
0.60	(0.508, 10, 8, 38, 0.5184, 0.01120)	0.035935	0.069564
0.65	(0.519, 13, 9, 49, 0.5545, 0.00810)	0.029926	0.056858
0.70	(0.509, 15, 9, 57, 0.5267, 0.01051)	0.024353	0.045324
0.75	(0.524, 20, 10, 75, 0.5400, 0.00910)	0.019420	0.035984
0.80	(0.514, 24, 10, 96, 0.5289, 0.01022)	0.014830	0.027266
0.85	(0.526, 34, 11, 138, 0.5413, 0.00895)	0.010701	0.019380
0.90	(0.537, 54, 12, 224, 0.5534, 0.00773)	0.006845	0.012378
0.95	(0.53, 108, 12, 452, 0.5402, 0.00893)	0.003305	0.005741
0.99	(0.52, 541, 12, 2280, 0.5318, 0.00985)	0.000641	-

Table 5.1: Rates for fixed values of p. N_1 and N_2 are the lengths of the inner codeword runs after the blow-up. I.e., $N_1 = \lceil M_1/(1-p) \rceil$ and $N_2 = \lceil M_2/(1-p) \rceil$.

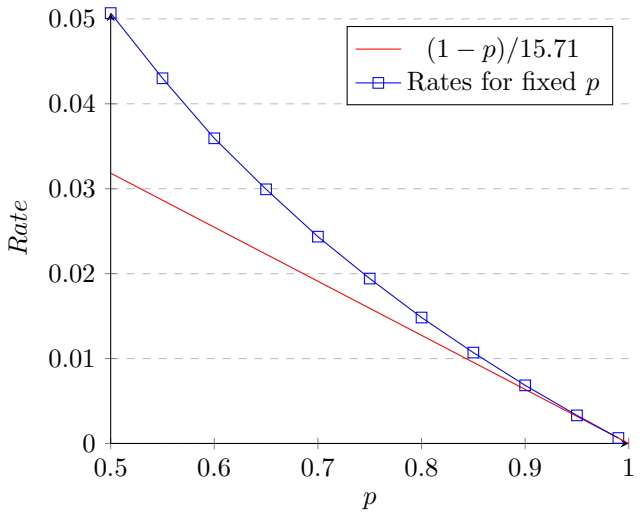

Figure 5.3: Rates for fixed values of p.

Removing this structure, he needed to design clever buffers (as opposed to our large chucks of zeros) and instead of classical code concatenations, he uses a recursive construction with a careful and tight analysis. In [PLW22], Pernice et al. improved this result and showed that it holds for any "general repeat channel" which is a broad family of synchronization channels that capture also the BDC and the Poisson repeat channel.

5.7 Poisson Repeat Channel

We first recall the definition of the PRC_λ.

Definition 5.7.1. *Let $\lambda > 0$. The* Poisson repeat channel with parameter λ (PRC_λ) *replaces each transmitted bit randomly (and independently of other transmitted bits), with a discrete number of copies of that bit, distributed according to the Poisson distribution with parameter λ.*

This channel was first defined by Mitzenmacher and Drinea in [MD06] who used it to prove a lower bound of $(1-p)/9$ on the rate of the BDC. More recently, Cheraghchi [Che18] gave an upper bound on its capacity and showed further connections to the BDC.

Before proceeding, let us describe the connection between the PRC and the BDC discovered by Mitzenmacher and Drinea. What they observed is that a code for the PRC_λ having rate $\mathcal{R}$, yields a code for the BDC_p of rate $(1-p)\cdot\mathcal{R}/\lambda$. The reduction is via a probabilistic argument – from each codeword in the code for the PRC_λ we generate a codeword for the BDC_p as follows: we replace each of the bits in the codeword by a discrete number of copies of those bits, distributed according to the Poisson distribution with parameter $\lambda/(1-p)$. The intuition for the construction is that now, when we send the codeword through the BDC_p, the resulting word is distributed as if we had sent the original codeword through the PRC_λ.

To the best of our knowledge, prior to this work there were no explicit deterministic constructions of coding schemes for the PRC_λ. In this section, we prove that the scheme that we constructed for the BDC can also be used for PRC (with slightly different parameters). We note that one can also use the construction given in [GL18] to obtain a deterministic construction for the PRC, yet our construction yields better rates in this case as well.

We focus on the regime where $\lambda \leq 0.5$, as, in some sense, the PRC behaves like the BDC for small values of λ – intuitively, the smaller λ is the more likely deletions are.

We now describe the construction for this channel. Note that most of the details are identical to our construction for the BDC_p. Therefore, in order not to repeat the entire proof, we focus on the differences and leave the details to the reader.

5.7.1 Construction

We use the same inner and outer codes defined in Proposition 5.3.8 and Theorem 5.2.11. For parameters $M_1 < M_2$ and M_B our construction is as follows:

Encoding. The only differences in the encoding procedure are the length of the buffers and the blow-up of the runs:

- We place a buffer of 0's between every two inner codewords, where the buffers length is $\lceil M_B m/\lambda \rceil$.
- Every run of length 1 is replaced with a run of length $\lceil M_1/\lambda \rceil$.
- Every run of length 2 is replaced with a run of length $\lceil M_2/\lambda \rceil$.

Remark 5.7.2. *We must choose $M_2 > \lambda$ since otherwise all runs in the inner code will be replaced with a run of length* 1*.*

Decoding. Since the inner and outer codes are the same we use the decoding algorithm given in Algorithm 5.

Rate. Similar to the calculations yielding Equation (5.1), the rate of this construction is

$$\begin{aligned}
\mathcal{R} &= \frac{\log\left(|\Sigma|^{\mathcal{R}_{\text{out}} n}\right)}{\beta_1 \lceil M_1/\lambda \rceil nm + \beta_2 \lceil M_2/\lambda \rceil nm + \lceil M_B m/\lambda \rceil (n-1)} \\
&\geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}}{\beta_1 \lceil M_1/\lambda \rceil + \beta_2 \lceil M_2/\lambda \rceil + M_B/\lambda + 1/m} \\
&\geq \frac{\mathcal{R}_{\text{out}}\mathcal{R}_{\text{in}} \cdot \lambda}{\beta_1 M_1 + \beta_2 M_2 + \beta\lambda + M_B + \lambda/m} \,.
\end{aligned} \tag{5.17}$$

As before, we can avoid the ceilings if we consider values of λ such that $\lceil M_1/\lambda \rceil$, $\lceil M_2/\lambda \rceil$ and $\lceil M_B m/\lambda \rceil$ are integers. In this case, the rate of the construction is given by

$$\mathcal{R} \geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}} \cdot \lambda}{\beta_1 M_1 + \beta_2 M_2 + M_B} \,. \tag{5.18}$$

5.7.2 Correctness of Decoding Algorithm

Since we use the same inner and outer codes in our encoding and the same decoding algorithm, the analysis performed in Section 5.5 can be repeated to this case as well with some minor modifications. We will briefly mention these modifications and leave the proofs to the reader.

We start by formally stating an analogous version of Proposition 5.5.1 to this setting.

Proposition 5.7.3. *Given $M_1, T, M_2, M_B, \beta_1, \delta_{\text{in}}, \varepsilon_{\text{in}}, \delta_{\text{out}}$ (as described in Section 5.7.1) let $Z_1 \sim \text{Poisson}(\lambda \lceil M_1/\lambda \rceil)$ and $Z_2 \sim \text{Poisson}(\lambda \lceil M_2/\lambda \rceil)$. Denote*

$$\begin{aligned} P^{(1)\to(2)} &:= \Pr[Z_1 \geq T+1] \,, \\ P^{(1)\to(0)} &:= \Pr[Z_1 = 0] \,, \\ P^{(2)\to(1)} &:= \Pr[Z_2 \leq T] \,, \\ P^{(2)\to(0)} &:= \Pr[Z_2 = 0] \,, \end{aligned}$$

and define

$$\gamma := \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2)\, P^{(1)\to(0)} + 4\beta_2 P^{(2)\to(0)} \,. \tag{5.19}$$

Let $x \in \Sigma^{\mathcal{R}_{\text{out}} n}$ be a message and let y be the string obtained after encoding x using our code and transmitting it through the PRC$_\lambda$. If $\gamma < \delta_{\text{in}}$, then there exists $\epsilon_0 = \epsilon_0(M_1, T, M_2, M_B, \beta_1, \delta_{\text{in}}, \delta_{\text{out}})$ such that for every $\varepsilon_{\text{out}} < \epsilon_0$ it holds that Algorithm 5 returns x with probability $1 - \exp(-\Omega(n))$.

Note that the only difference between this proposition and Proposition 5.5.1 is in the definitions of Z_1 and Z_2. Recall that the proof of Proposition 5.5.1 heavily relies on Propositions 5.5.2, 5.5.3, and 5.5.5. Therefore, to prove Proposition 5.7.3, one needs to formally state and prove analogous versions of Propositions 5.5.2, 5.5.3, and 5.5.5 in the setting of the PRC.

We first observe that it is very simple to prove the analogous claims to Propositions 5.5.2 and 5.5.3 by using Lemma 5.2.8 instead of Lemma 5.2.7 (since our random variables are now distributed according to the Poisson distribution). Hence we omit the details. We thus have that the probability of each error type is $\exp(-\Omega(m))$ per inner codeword. We focus on analyzing the case where we might output a wrong inner codeword in Step 3 of Algorithm 5 (i.e. the case analyzed in Proposition 5.5.5).

Wrong Inner Decoding

Note that as we consider the same threshold decoding step for decoding the inner windows (i.e., Step 2 in Algorithm 5) and the same inner code, the claims of Section 5.5.1 apply here as well. The difference from Section 5.5.1 is in the computations of the probabilities $P^{(1)\to(2)}$, $P^{(1)\to(0)}$, $P^{(2)\to(1)}$, $P^{(2)\to(0)}$. We focus on these computations as they play a significant role in computing the rate in Theorem 5.1.2.

Recall that in the encoding process, a run r_j is replaced with a run of length $\lceil M_1/\lambda \rceil$ or $\lceil M_2/\lambda \rceil$ depending on r_j's length. As in Section 5.5.1, define Z_j to be the random variable corresponding to the number of bits from this blown-up run that survived the transmission through the PRC$_\lambda$. According to Lemma 5.2.3, $Z_j \sim \text{Poisson}(\lambda \lceil M_1/\lambda \rceil)$ if $|r_j| = 1$ and $Z_j \sim \text{Poisson}(\lambda \lceil M_2/\lambda \rceil)$ if $|r_j| = 2$. Let r'_j be exactly as defined in Definition 5.5.8. As before, we study the probability that $r_j \neq r'_j$:

1. If $|r_j| = 1$ then there are two possible types of errors:
   (a) $|r'_j| = 2$: The probability for this to happen is $P^{(1)\to(2)} := \Pr[Z_j \geq T+1]$. We next give two estimates, one is an exact calculation and the other is an upper bound. For every λ we have

$$\begin{aligned} P^{(1)\to(2)} &= \Pr[Z_j \geq T+1] \\ &= 1 - \Pr[Z_j \leq T] \\ &= 1 - e^{-\lambda \lceil \frac{M_1}{\lambda} \rceil} \sum_{i=0}^{T} \frac{(\lambda \lceil \frac{M_1}{\lambda} \rceil)^i}{i!} \,. \end{aligned} \tag{5.20}$$

Let Y be a random variable distributed as $Y \sim \text{Poisson}(M_1 + \lambda)$. We can upper bound $P^{(1)\to(2)}$ by

$$\begin{aligned} P^{(1)\to(2)} = \Pr[Z_j \geq T+1] &= 1 - \Pr[Z_j \leq T] \\ &\leq 1 - \Pr[Y \leq T] \\ &= 1 - e^{-M_1-\lambda} \sum_{i=0}^{T} \frac{(M_1+\lambda)^i}{i!} \ . \end{aligned} \tag{5.21}$$

where the inequality follows from Lemma 5.2.4 by noting that $\lambda \lceil M_1/\lambda \rceil \leq M_1 + \lambda$.

(b) $\left|r'_j\right| = 0$: In this case, r_j was completely deleted by the channel. The probability for this to happen is

$$P^{(1)\to(0)} = \Pr[Z_j = 0] = e^{-\lambda \lceil M_1/\lambda \rceil} \leq e^{-M_1} \ . \tag{5.22}$$

2. If $|r_j| = 2$ then one of the following cases hold:

- $\left|r'_j\right| = 1$: The probability for this to happen is $P^{(2)\to(1)} := \Pr[Z_j \leq T]$. As before, the exact probability calculation is

$$P^{(2)\to(1)} = \Pr[Z_j \leq T] = e^{-\lambda \lceil \frac{M_2}{\lambda} \rceil} \sum_{i=0}^{T} \frac{(\lambda \lceil \frac{M_2}{\lambda} \rceil)^i}{i!} \ . \tag{5.23}$$

  Let Y be a random variable distributed as $Y \sim \text{Poisson}(M_2)$ then it holds that

$$P^{(2)\to(1)} = \Pr[Z_j \leq T] \leq \Pr[Y \leq T] = e^{-M_2} \sum_{i=0}^{T} \frac{M_2^i}{i!} \ , \tag{5.24}$$

  where the inequality follows from Lemma 5.2.4 by noting that $M_2 \leq \lambda \lceil M_2/\lambda \rceil$.

- $\left|r'_j\right| = 0$. The probability for this to happen is

$$P^{(2)\to(0)} = \Pr[Z_j = 0] = e^{-\lambda \lceil M_2/\lambda \rceil} \leq e^{-M_2} \ . \tag{5.25}$$

By using these estimates and proceeding exactly as in the proof of Proposition 5.5.5 one gets that the probability of error in this case as well is $\exp(-\Omega(m))$. Combining everything together the proof of Proposition 5.7.3 follows similarly to the proof of Proportion 5.5.1 . In particular, Algorithm 5 decodes correctly in this setting as well.

Proof of Theorem 5.1.2

As in the proof of Theorem 5.1.1, we first compute an upper bound on γ (recall its definition in Proposition 5.7.3) that holds for all $\lambda \leq 0.5$, then we compute the rate of the inner code by using Proposition 5.3.8 and finally we compute the rate of our code using Equation 5.17.

The parameters we use for our construction are

$$M_1 = 5.49, M_2 = 24.2, \beta_1 = 0.532, M_B = 10^{-5} \quad \text{and } \delta_{\text{out}} = 2^{-20} \ .$$

We pick $T = 13$ and set $\delta_{\text{in}} = 0.00954$.

First observe that for every $\lambda > 0$, we can upper bound $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ using Equations (5.22), (5.24), and (5.25) respectively. As we assume $\lambda \leq 0.5$, we can upper bound $P^{(1)\to(2)}$ using Equation (5.21) with $\lambda = 0.5$ (due to monotonicity implied by Lemma 5.2.4). Plugging these upper bounds to Equation (5.19), we get an upper bound on γ which, as before, we denote by $\tilde{\gamma}$. Calculating, it is simple to verify that $\gamma \leq \tilde{\gamma} < \delta_{\text{in}}$. Therefore, for a small enough ε_{out}, our decoding algorithm succeeds with high probability. Applying Proposition 5.3.8 we get an inner code of rate $\mathcal{R}_{\text{in}} = 0.53186$ and by letting m be large enough, the rate of our concatenated code according to Equation (5.17) is

$$\mathcal{R} = \frac{0.5318\lambda}{8.58349 + 0.766\lambda + \lambda/m} > \frac{\lambda}{17} \ .$$

Bibliography

[AG19] Omar Alrabiah and Venkatesan Guruswami. An exponential lower bound on the sub-packetization of MSR codes. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 979–985, 2019.

[AGFC07] Khaled AS Abdel-Ghaffar, Hendrik C Ferreira, and Ling Cheng. On linear and cyclic codes for correcting deletions. In *2007 IEEE International Symposium on Information Theory*, pages 851–855. IEEE, 2007.

[AS65] Theodore W Anderson and Stephen M Samuels. Some inequalities among binomial and poisson probabilities. In *Proc. Fifth Berkeley Symp. Math. Statist. Probab*, volume 1, pages 1–12, 1965.

[BBD+22] Amit Berman, Sarit Buzaglo, Avner Dor, Yaron Shany, and Itzhak Tamo. Repairing reed–solomon codes evaluated on subspaces. *IEEE Transactions on Information Theory*, 2022.

[BDIR18] Fabrice Benhamouda, Akshay Degwekar, Yuval Ishai, and Tal Rabin. On the local leakage resilience of linear secret sharing schemes. In *Annual International Cryptology Conference*, pages 531–561. Springer, 2018.

[BGZ17] Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Transactions on Information Theory*, 64(5):3403–3410, 2017.

[BLC+16] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A DNA-based archival storage system. *ACM SIGARCH Computer Architecture News*, 44(2):637–649, 2016.

[BLOS+21] Daniella Bar-Lev, Itai Orr, Omer Sabary, Tuvi Etzion, and Eitan Yaakobi. Deep dna storage: Scalable and robust dna storage via coding theory and deep learning. *arXiv preprint arXiv:2109.00031*, 2021.

[BSZ17] Christine Bachoc, Oriol Serra, and Gilles Zémor. An analogue of Vosper's theorem for extension fields. *Mathematical Proceedings of the Cambridge Philosophical Society*, 163(3):423–452, 2017.

[Cau12] Augustin Louis Baron Cauchy. *Recherches sur les nombres*. 1812.

[CCS+18] Douglas Carmean, Luis Ceze, Georg Seelig, Kendall Stewart, Karin Strauss, and Max Willsey. Dna data storage and hybrid molecular–electronic computing. *Proceedings of the IEEE*, 107(1):63–72, 2018.

[CGHL21] Kuan Cheng, Venkatesan Guruswami, Bernhard Haeupler, and Xin Li. Efficient linear and affine codes for correcting insertions/deletions. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1–20. SIAM, 2021.

[Che18] Mahdi Cheraghchi. Capacity upper bounds for deletion-type channels. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 493–506. ACM, 2018.

[CHL+19] Kuan Cheng, Bernhard Haeupler, Xin Li, Amirbehshad Shahrasbi, and Ke Wu. Synchronization strings: Highly efficient deterministic constructions over small alphabets. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2185–2204. SIAM, 2019.

[CJLW18] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 200–211. IEEE, 2018.

[CR03] Maxime Crochemore and Wojciech Rytter. *Jewels of stringology: text algorithms*. World Scientific, 2003.

[CR20] Mahdi Cheraghchi and João Ribeiro. An overview of capacity results for synchronization channels. *IEEE Transactions on Information Theory*, 67(6):3207–3232, 2020.

[CS22] Roni Con and Amir Shpilka. Improved constructions of coding schemes for the binary deletion channel and the poisson repeat channel. *IEEE Trans. Inf. Theory*, 68(5):2920–2940, 2022.

[CST22] Roni Con, Amir Shpilka, and Itzhak Tamo. Explicit and efficient constructions of linear codes against adversarial insertions and deletions. *IEEE Transactions on Information Theory*, 68(10):6516–6526, 2022.

[CST23] Roni Con, Amir Shpilka, and Itzhak Tamo. Reed–solomon codes against adversarial insertions and deletions. *IEEE Transactions on Information Theory*, 2023.

[CT22] Roni Con and Itzhak Tamo. Nonlinear repair of reed-solomon codes. *IEEE Transactions on Information Theory*, 68(8):5165–5177, 2022.

[CZ21] Bocong Chen and Guanghui Zhang. Improved Singleton bound on insertion-deletion codes and optimal constructions. *arXiv preprint arXiv:2105.02004*, 2021.

[Dal11] Marco Dalai. A new bound on the capacity of the binary deletion channel with high deletion probabilities. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 499–502. IEEE, 2011.

[Dav35] Harold Davenport. On the addition of residue classes. *Journal of the London Mathematical Society*, 1(1):30–32, 1935.

[DD08] Danyo Danev and Stefan Dodunekov. A family of ternary quasi-perfect BCH codes. *Designs, Codes and Cryptography*, 49(1-3):265–271, 2008.

[DDKM18] Hoang Dau, Iwan M Duursma, Han Mao Kiah, and Olgica Milenkovic. Repairing Reed-Solomon codes with multiple erasures. *IEEE Transactions on Information Theory*, 64(10):6567–6582, 2018.

[DGW+10] 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.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[DLTX19] Tai Do Duc, Shu Liu, Ivan Tjuawinata, and Chaoping Xing. Explicit Constructions of Two-Dimensional Reed-Solomon Codes in High Insertion and Deletion Noise Regime. *arXiv preprint arXiv:1909.03426*, 2019.

[DM07] Eleni Drinea and Michael Mitzenmacher. Improved lower bounds for the capacity of iid deletion and duplication channels. *IEEE Transactions on Information Theory*, 53(8):2693–2714, 2007.

[EL73] Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite and Finite Sets, Keszthely (Hungary)*. Citeseer, 1973.

[Eli55] Peter Elias. Coding for noisy channels. *IRE Convention Record*, 4:37–46, 1955.

[ERR10] Salim El Rouayheb and Kannan Ramchandran. Fractional repetition codes for repair in distributed storage systems. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1510–1517. IEEE, 2010.

[ET41] Paul Erdös and Pál Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, 1(4):212–215, 1941.

[FD10] Dario Fertonani and Tolga M Duman. Novel bounds on the capacity of the binary deletion channel. *IEEE Transactions on Information Theory*, 56(6):2753–2765, 2010.

[GERCP13] Sreechakra Goparaju, Salim El Rouayheb, Robert Calderbank, and H Vincent Poor. Data secrecy in distributed storage systems under exact repair. In *2013 International Symposium on Network Coding (NetCod)*, pages 1–6. IEEE, 2013.

[GFV17] Sreechakra Goparaju, Arman Fazeli, and Alexander Vardy. Minimum storage regenerating codes for all parameters. *IEEE Transactions on Information Theory*, 63(10):6318–6328, 2017.

[GH21] Venkatesan Guruswami and Johan Håstad. Explicit two-deletion codes with redundancy matching the existential bound. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 21–32. SIAM, 2021.

[GK] Venkatesan Guruswami and Cheng Kuan. personal communication.

[GL18] Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 2018.

[GRS12] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at http://www. cse. buffalo. edu/ atri/courses/coding-theory/book*, 2012.

[GS86] Igor Borisovich Gashkov and Vladimir Michilovich Sidel'nikov. Linear ternary quasi-perfect codes correcting double errors. *Problemy Peredachi Informatsii*, 22(4):43–48, 1986.

[GS95] Arnaldo Garcia and Henning Stichtenoth. A tower of artin-schreier extensions of function fields attaining the drinfeld-vladut bound. *Inventiones mathematicae*, 121(1):211–222, 1995.

[GS96] Arnaldo Garcia and Henning Stichtenoth. On the asymptotic behaviour of some towers of function fields over finite fields. *Journal of number theory*, 61(2):248–273, 1996.

[GW17a] Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.

[GW17b] Venkatesan Guruswami and Mary Wootters. Repairing Reed-Solomon codes. *IEEE transactions on Information Theory*, 63(9):5684–5698, 2017.

[Hae19] Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 334–347. IEEE, 2019.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[HMG19] Reinhard Heckel, Gediminas Mikutis, and Robert N Grass. A characterization of the DNA data storage channel. *Scientific reports*, 9(1):1–12, 2019.

[Hoe94] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[HS17] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the Singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 33–46. ACM, 2017.

[HS21] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings and codes for insertions and deletions - A survey. *IEEE Trans. Inf. Theory*, 67(6):3190–3206, 2021.

[HSRD17] Reinhard Heckel, Ilan Shomorony, Kannan Ramchandran, and NC David. Fundamental limits of dna storage systems. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 3130–3134. IEEE, 2017.

[KMS10] Adam Kalai, Michael Mitzenmacher, and Madhu Sudan. Tight asymptotic bounds for the deletion channel with small deletion probabilities. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 997–1001. IEEE, 2010.

[Kot96] Ralf Kotter. Fast generalized minimum-distance decoding of algebraic-geometry and Reed-Solomon codes. *IEEE Transactions on Information Theory*, 42(3):721–737, 1996.

[Lev66] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[LSWZY19] Andreas Lenz, Paul H Siegel, Antonia Wachter-Zeh, and Eitan Yaakobi. Coding over sets for dna storage. *IEEE Transactions on Information Theory*, 66(4):2331–2351, 2019.

[LT21] Shu Liu and Ivan Tjuawinata. On 2-dimensional insertion-deletion Reed-Solomon codes with optimal asymptotic error-correcting capability. *Finite Fields and Their Applications*, 73:101841, 2021.

[LWJ19] Weiqi Li, Zhiying Wang, and Hamid Jafarkhani. On the sub-packetization size and the repair bandwidth of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 65(9):5484–5502, 2019.

[LX21] Shu Liu and Chaoping Xing. Bounds and constructions for insertion and deletion codes. *arXiv preprint arXiv:2111.14026*, 2021.

[Mas84] Richard C. Mason. *Diophantine Equations over Function Fields*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1984.

[Mas85] David W. Masser. Open problems. In *Chen, W.W.L. (ed.). Proceedings of the Symposium on Analytic Number Theory. Imperial College, London*, 1985.

[MBT10] Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials*, 12(1):87–96, 2010.

[MBW18] Jay Mardia, Burak Bartan, and Mary Wootters. Repairing multiple failures for scalar MDS codes. *IEEE Transactions on Information Theory*, 65(5):2661–2672, 2018.

[MD06] Michael Mitzenmacher and Eleni Drinea. A simple lower bound for the capacity of the deletion channel. *IEEE Transactions on Information Theory*, 52(10):4657–4660, 2006.

[Mit09] Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

[MS81] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[Oes88] Joseph Oesterlé. Nouvelles approches du "théoreme" de Fermat. *Astérisque*, 161(162):165–186, 1988.

[PDC13] Dimitris S Papailiopoulos, Alexandros G Dimakis, and Viveck R Cadambe. Repair optimal erasure codes through Hadamard designs. *IEEE Transactions on Information Theory*, 59(5):3021–3037, 2013.

[PLW22] Francisco Pernice, Ray Li, and Mary Wootters. Efficient capacity-achieving codes for general repeat channels. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 3097–3102. IEEE, 2022.

[RD14] Mojtaba Rahmati and Tolga M Duman. Upper bounds on the capacity of deletion channels using channel fragmentation. *IEEE Transactions on Information Theory*, 61(1):146–156, 2014.

[RLT21] Netanel Raviv, Ben Langton, and Itzhak Tamo. Multivariate Public Key Cryptosystem from Sidon Spaces. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 242–265. Springer, 2021.

[RRT17] Ron M. Roth, Netanel Raviv, and Itzhak Tamo. Construction of Sidon spaces with applications to coding. *IEEE Transactions on Information Theory*, 64(6):4412–4422, 2017.

[RSE17] Netanel Raviv, Natalia Silberstein, and Tuvi Etzion. Constructions of high-rate minimum storage regenerating codes over small fields. *IEEE Transactions on Information Theory*, 63(4):2015–2038, 2017.

[RSG+14] KV Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A" hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 331–342, 2014.

[RSK11] Korlakai Vinayak Rashmi, Nihar B Shah, and P Vijay Kumar. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. *IEEE Transactions on Information Theory*, 57(8):5227–5239, 2011.

[RSRK12] KV Rashmi, Nihar B Shah, Kannan Ramchandran, and P Vijay Kumar. Regenerating codes for errors and erasures in distributed storage. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 1202–1206. IEEE, 2012.

[Rub22] Ittai Rubinstein. Explicit and efficient construction of nearly optimal rate codes for the binary deletion channel and the poisson repeat channel. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 105:1–105:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[SAK+01] Kenneth W Shum, Ilia Aleshnikov, P Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. A low-complexity algorithm for the construction of algebraic-geometric codes better than the gilbert-varshamov bound. *IEEE Transactions on Information Theory*, 47(6):2225–2241, 2001.

[SAP+13] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris S. Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: Novel erasure codes for big data. *Proc. VLDB Endow.*, 6(5):325–336, 2013.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[SH+22] Ilan Shomorony, Reinhard Heckel, et al. Information-theoretic foundations of dna data storage. *Foundations and Trends® in Communications and Information Theory*, 19(1):1–106, 2022.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[SNW02] Reihaneh Safavi-Naini and Yejing Wang. Traitor tracing for shortened and corrupted fingerprints. In *ACM workshop on Digital Rights Management*, pages 81–100. Springer, 2002.

[Soo08] Tan Jin Soon. QR code. *Synthesis Journal*, 2008:59–78, 2008.

[SPDC14] Karthikeyan Shanmugam, Dimitris S Papailiopoulos, Alexandros G Dimakis, and Giuseppe Caire. A repair framework for scalar MDS codes. *IEEE Journal on Selected Areas in Communications*, 32(5):998–1007, 2014.

[SRV15] Natalia Silberstein, Ankit Singh Rawat, and Sriram Vishwanath. Error-correcting regenerating and locally repairable codes via rank-metric codes. *IEEE Transactions on Information Theory*, 61(11):5765–5778, 2015.

[Sti09] Henning Stichtenoth. *Algebraic function fields and codes*, volume 254. Springer Science & Business Media, 2009.

[Sto81] Walter W. Stothers. Polynomial identities and Hauptmoduln. *The Quarterly Journal of Mathematics*, 32(3):349–370, 1981.

[SV90] Alexei N Skorobogatov and Serge G Vladut. On the decoding of algebraic-geometric codes. *IEEE Transactions on Information Theory*, 36(5):1051–1060, 1990.

[TPFV21] Ido Tal, Henry D Pfister, Arman Fazeli, and Alexander Vardy. Polar codes for the deletion channel: Weak and strong polarization. *IEEE Transactions on Information Theory*, 68(4):2239–2265, 2021.

[TSN07] Dongvu Tonien and Reihaneh Safavi-Naini. Construction of deletion correcting codes using generalized Reed–Solomon codes and their subcodes. *Designs, Codes and Cryptography*, 42(2):227–237, 2007.

[Tuc94] Alan Tucker. *Applied combinatorics*. John Wiley & Sons, Inc., 1994.

[TVZ82] Michael A Tsfasman, Serge Vlădutx, and Thomas Zink. Modular curves, Shimura curves, and Goppa codes, better than Varshamov-Gilbert bound. *Mathematische Nachrichten*, 109(1):21–28, 1982.

[TWB12] Itzhak Tamo, Zhiying Wang, and Jehoshua Bruck. Zigzag codes: MDS array codes with optimal rebuilding. *IEEE Transactions on Information Theory*, 59(3):1597–1616, 2012.

[TYB17] Itzhak Tamo, Min Ye, and Alexander Barg. Optimal repair of Reed-Solomon codes: Achieving the cut-set bound. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 216–227. IEEE, 2017.

[Vos56] Alan Gordon Vosper. The critical pairs of subsets of a group of prime order. *Journal of the London Mathematical Society*, 1(2):200–205, 1956.

[VW03] Leonid N Vaserstein and Ethel R Wheland. Vanishing polynomial sums. *Communications in Algebra*, 31(2):751–772, 2003.

[WB99] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[WMSN04] Yejing Wang, Luke McAven, and Reihaneh Safavi-Naini. Deletion correcting using generalized Reed-Solomon codes. In *Coding, Cryptography and Combinatorics*, pages 345–358. Springer, 2004.

[WTB16] Zhiying Wang, Itzhak Tamo, and Jehoshua Bruck. Explicit minimum storage regenerating codes. *IEEE Transactions on Information Theory*, 62(8):4466–4480, 2016.

[YB17a] 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.

[YB17b] Min Ye and Alexander Barg. Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization. *IEEE Transactions on Information Theory*, 63(10):6307–6317, 2017.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, pages 216–226, 1979.