Technische Universität München

TUM School of Computation, Information and Technology

Error-Correction for DNA-based Data Storage Systems

Lorenz Welter

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigten Dissertation.

Vorsitz:		Prof. Dr. Reinhard Heckel
Prüfende der Dissertation:	1.	Prof. Dr.-Ing. Antonia Wachter-Zeh
	2.	Prof. Dr. Moshe Schwartz

Die Dissertation wurde am 07.11.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 28.03.2025 angenommen.

Abstract

In the quest for efficient and durable data storage solutions, DNA-based data storage (DNA storage) has emerged as a promising frontier for archival storage. This new storage solution offers many new research problems for the information and coding theory community. Especially the inherent nature of the DNA storage process with the unordered storage fashion, the random number of noisy copies for different data parts, and the occurrence of data-dependent synchronization errors (insertions and deletions) impose challenges and opportunities that have yet to be fully understood to enable its commercial application. This dissertation focuses on taking a step forward in achieving this objective by designing several novel error-correction schemes for subproblems of the overall DNA storage process.

The first part of this dissertation deals with designing codes for insertion and deletion errors for adversarial channels. First, we discuss the setting of codes correcting combinations of insertion/deletion errors with classical substitution errors. In contrast to the design of error-correcting codes for edit errors (insertion, deletion, or substitution error), by fixing the insertion/deletion and substitution errors to different constants, we can leverage this prior knowledge in the design process. We present non-asymptotic bounds on the cardinality of generic codes affected by this error type. For the main result, we focus on the special case of a single insertion/deletion and single substitution error. We discuss properties of generic number-theoretic codes and propose a code construction of a binary single-deletion single-substitution correcting code with low redundancy. Second, we generalize the insertion/deletion correcting problem to two-dimensional arrays, i.e., we study the unique recovery of arrays after insertions or deletions of rows and columns. We extend the well-known insertion and deletion equivalence for vectors to arrays. Additionally, we develop code constructions based on a locate-decode strategy which we can benchmark with our prior derived bounds. We introduce special structures in the code arrays that assist in locating the row and column insertions/deletions, consequently transforming the problem into an erasure-correction problem, which can be solved by classical error-correction mechanisms.

In the second part of the dissertation, we consider coding for the end-to-end DNA storage process comprising the DNA synthesis, the storage, and the DNA sequencing steps. We model the DNA storage channel as a stochastic channel where the input and output are several unordered short DNA strands. In detail, we draw randomly with replacement multiple copies of the input strands, each copy passing through a strand-dependent noise channel inflicting synchronization errors. The set of output DNA strands is obtained by randomly permuting the strands after the noise channel. We propose an index-based concatenated coding scheme together with a low-complexity decoder. We achieve excellent performance on experimental data obtained by nanopore sequencing due to clever channel modeling, tackling negative effects and leveraging inherent opportunities of the DNA storage channel effectively, and soft information processing in the decoder.

Contents

II Coding Schemes for the DNA Storage Channel with a Probabilistic Decoder

Appendix

Chapter 1

Introduction and Overview

The start of the digital age enabled us to communicate and store massive amounts of data. According to current projections, the demand for digital data storage is exponentially increasing from current amounts in the magnitude of Zettabytes (one trillion Terabyte) to Yottabytes (one quadrillion Terabyte) in the future, requiring innovative storage ideas [ZZS+16]. A storage solution is mainly benchmarked by four key aspects: density (bits per volume), durability (long retention period), access speed (latency and bandwidth), and energy-efficiency (while storing and during read/write process) [CNS19]. A vast majority of the data subject to be stored is aimed for *archival storage*, i.e., data that is not accessed for a longer time span; thus, the crucial factors are high density, long durability, and energy efficiency during the storage process for this application. Under the data growth predictions, current storage solutions such as magnetic (e.g., hard disk drives and tape), optical (e.g., CDs and Blu-ray), and solid-state (e.g., flash) memories are not able to keep up with the demand for archival storage even when reaching their predicted fundamental storage limits. In fact, their utilization would lead to immense maintenance costs, mainly produced by the vast space required due to their limited storage density and the necessity to rewrite due to their limited life span. Hence, researchers have pinpointed molecular data storage, especially deoxyribonucleic acid (DNA), as a potential candidate to solve the world's future storage problem. According to DNA's projected theoretical storage density, today's storage demand could fit in a 10 cm^3 box full of DNA, and the world's projected storage demand in 2040 could be satisfied by simply one kilogram of DNA [ZZS+16]. To turn this dream into reality, many challenges have to be overcome, and research advances have yet to be made. Especially due to the success of large-scale proof-of-concepts experiments in the early 2010s, e.g., [CGK12; GBC+13], DNA-based data storage (in short DNA storage) has attracted considerable attention not only in the biotechnology field but also in the information and coding theory community. The unique storage flow and noise sources impose interesting research problems on a theoretical and practical level. It is evident that error-correction schemes are necessary to ensure reliable data storage.

In this dissertation, we deal with information- and coding-theoretic problems related to DNA storage. First, we discuss error-correction schemes for specific adversarial channels with insertion and deletion errors, which are two prominent error types occurring in the DNA storage process. Second, we propose a practical and flexible end-to-end coding scheme for the DNA storage channel, which is a close abstraction of the full storage process.

1.1 Outline

We briefly discuss the structure of the dissertation and provide a short summary for each chapter. The dissertation is split into three parts: *Preliminaries* (Chapter 2), *Codes on Adversarial Insertion/Deletion Channels* (Part I, Chapters 3 to 5), and *Coding Schemes for the DNA Storage Channel with a Probabilistic Decoder* (Part II, Chapters 6 and 7).

Chapter 2 highlights the principles and challenges of DNA storage. We also provide a historical review of DNA storage experiments with a coding-theoretic perspective. Further, we introduce the mathematical notation used throughout the dissertation. Then, we discuss components and terminology for reliable data storage. We distinguish the two channel model approaches: adversarial and stochastic model. Finally, we discuss the different error types treated in this dissertation and introduce their associated distance metrics.

Chapter 3 discusses the principles of error-correction for insertion and deletion errors in the adversarial setting. First, we provide a historical review of insertion and deletion error correction. Then, we discuss the properties of codes correcting insertion and deletions, mainly their error-correction equivalence and bounds on the redundancy of such codes. Finally, we present code constructions of well-known insertion/deletion error-correcting codes, e.g., the Varshamov-Tenengolts codes.

Chapter 4 considers the correction of fixed combinations of insertion/deletion and substitution errors. We discuss bounds on the cardinality of error-correcting codes for this type of error. Then, we analyze the properties of number-theoretic codes when affected by a single deletion and single substitution error. Building on these results, we construct single-deletion single-substitution error-correcting codes.

Chapter 5 generalizes the insertion and deletion coding problem to the two-dimensional space. In particular, we consider the problem of insertion or deletion of rows and columns in arrays. We extend the insertion/deletion equivalence to arrays and also derive bounds for codes under this error model. Then, we present two code constructions: One deals with a simplified channel where the code array is affected by a fixed single-row and single-column insertion/deletion error. Then, this construction is generalized to tackle combinations of row and column insertion/deletions.

Chapter 6 introduces channel coding for insertion, deletion, and substitution errors assuming a stochastic model for single sequence and multiple sequence transmission. Further, we provide a historical review for error-correction on such channels and introduce information-theoretic tools to analyze the performance of channel codes with decoders unaware of the true channel law.

Chapter 7 treats the problem of building a practical end-to-end DNA storage coding solution. Further, we discuss the DNA storage channel model with its opportunities and challenges. In particular, we model error events as strand-dependent errors to narrow the gap of the true underlying error channel explicitly when applying nanopore sequencing. Additionally, we take into account the unordered storage fashion and the random copy effect of the DNA storage channel. We propose a practical coding scheme together with the design of a low-complexity decoder. We evaluate this scheme on experimental data.

Chapter 8 concludes the dissertation.

Chapter 2

Preliminaries

The dissertation treats topics related to error-correction mechanisms in DNA storage. Hence, we highlight the main principles and challenges in DNA storage in Section 2.1. Subsequently, we introduce the mathematical notation used throughout the dissertation in Section 2.2. Afterwards, we briefly discuss the fundamental principles of error-correction for this dissertation in Section 2.3.

2.1 DNA-based Data Storage

Natural evolution selected genes as the fundamental elements that carry the biological information of living organisms. Genes are structured DNA sequences that are used to store and communicate over time, e.g., over generations, the information of functional products of the body, i.e., primarily recipes of proteins, which are an essential fundamental building entity of living organisms. Inspired by nature, researchers are pushing towards utilizing DNA as a medium to store artificial data, i.e., translating binary digital data into DNA sequences or vice versa. DNA is a double-helix formed by two complementary, directional strands twisted around each other which are composed of the four essential molecules called nucleotides: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) [MK05]. The respective complement relation of the nucleotides is defined by the Watson-Crick pairing A – T and C – G. Due to this fixed mapping, we describe a DNA strand as a single chain of nucleotides throughout this dissertation.

DNA storage offers four main advantages [MGK$^+$18; PMB$^+$18] solving the high storage demand in the future:

1. **Energy-efficient long-term storage:**
   DNA is very stable compared to other storage media. It can be retrieved after thousands of years with minimal maintenance efforts if stored in an appropriate medium. The work [GHP$^+$15] showed error-free DNA storage over 2000 years at 10°C and 2 000 000 years at −18°C (simulated via heating process) in Sicilia containers.

2. **Ultra-dense storage capacity:**
   It offers a theoretical storage capacity of 100 trillion (10^{12}) Gigabyte per gram. The work in [OCA$^+$20] achieved already a storage density of 17 Exabytes (10^9 Gigabyte) per gram.

3. **Easily replicable DNA strands:**
   Using polymerase chain reaction (PCR), DNA strands can be easily and rapidly replicated, enabling large-scale data amplification, e.g., for data backups or creating physical redundancy for better data readout.

4. **Non-obsolete technology:**
   Due to DNA's fundamental role in nature and researchers' curiosity to understand and learn from nature, technologies involved with DNA will always be accessible and improved, e.g., for writing and reading purposes.

As these key points seem promising, many challenges must be overcome such that DNA storage can be practical, i.e., commercially in use [MGK+18; DPG+22]:

1. **Cost-effectiveness:**
   Using current technology, it is estimated to cost 800 million US dollars per Terabyte; in contrast, storage using magnetic tape costs around 15 US dollars per Terabyte.

2. **High latency for data writing and retrieval:**
   The current writing and reading speeds are both in the order of kilobytes per second (to be competitive, it is expected to be in the order of gigabytes per second) [GMM16].

3. **Large-scale rewritability and random-access:**
   Due to the expected high storage capacity, it is inherent that rewriting or random-access data retrieval needs to be scaled up, i.e., made practical, since mainly only a tiny fraction of the data is desired to be altered or read.

4. **Strand data errors:**
   Due to natural or technological impairments in writing, storing, and reading processes, different error types occur, which require the development of new (or more efficient) error-handling mechanisms [HMG19]. Further, the errors statistically depend on the DNA strands and the system components used. For example, insertion and deletion errors appear within a single DNA strand. Moreover, specific DNA structures, e.g., multiple consecutive equal nucleotides (homopolymers), cause higher (mostly deletion) error rates [HMG19]. Strand losses and breaks can also occur within the storage processes [GHP+15; GSG+24].

5. **Unordered and short sequence storage design:**
   Due to current synthesis limitations, DNA strands are presently around 100–200 nucleotides long and are stored in an unordered fashion. This requires a practical way of reordering the strands [GHP+15] and is mainly tackled using addresses in the individual strands.

Significant efforts and scientific research have recently been made to understand the opportunities and overcome the challenges. The dissertation focuses on error handling (the last two challenges), considering easy replicability. Specifically, the design of error-correction codes and decoding algorithms will be discussed in various scenarios. Note that constrained

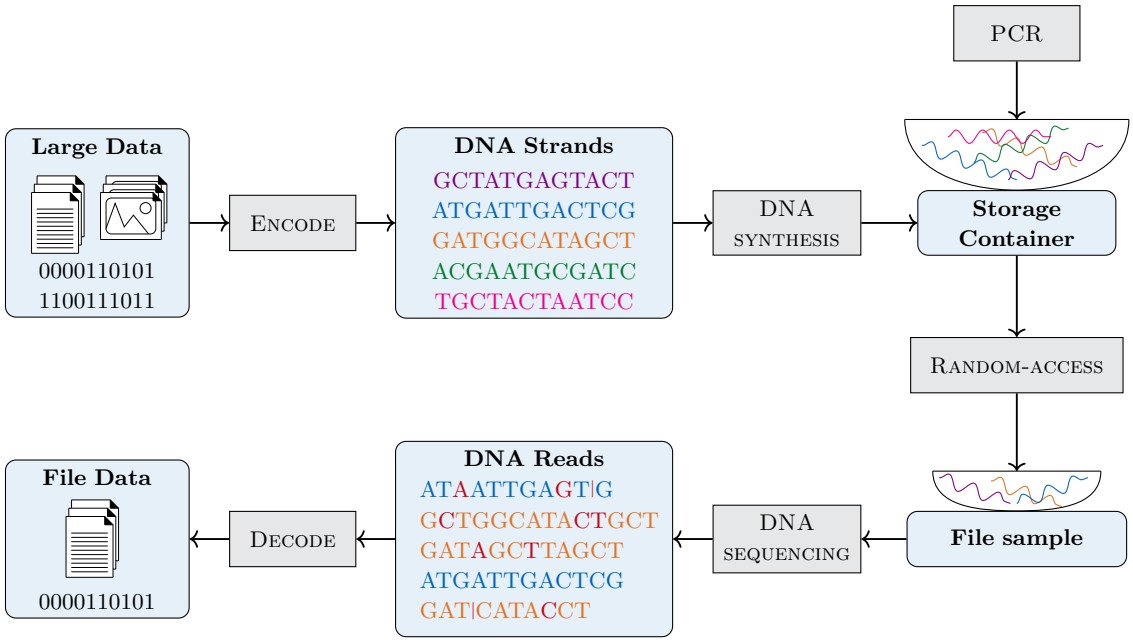


Figure 2.1: Flow together with the components of a generic DNA storage system.

coding, i.e., error avoidance by shaping the stored DNA strands such that error-prone structures are avoided, may also be an option to handle errors [SKS+24]. We refer to [DPG+22] for a deeper view of other efforts of the challenges. A historical review on DNA storage with a coding-theoretic view is given in Section 2.1.2.

Upcoming, the basic principles and components of a DNA storage system will be discussed, providing an overview of the process of the initial data storage until the final act of data retrieval.

2.1.1 Components of a DNA-Based Data Storage System

The overall task of conventional data storage is retaining information over time. A usual storage system consists of four essential components: writing, storage, retrieval, and reading. In the following, a typical DNA storage system flow is described and depicted in Figure 2.1 [CNS19; DPG+22]. The described flow is generic and may be adapted to specific system targets or technologies. Note that rewriting particular data in an existing storage is not discussed.

Writing process: A large amount of user data, commonly seen as binary data, is subject to be stored as DNA. Hence, an algorithm (*encoder*) maps the binary string into several short quaternary sequences, comprised by the symbols $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ representing the different nucleotides, since current synthesis technologies provide only high-accuracy results for short DNA strands. In most practical systems, the encoder adds index information to each DNA strand to enable reordering of the data. Most importantly, the encoder strategically inserts logical redundancy using error-correcting codes to enable recovery even in the presence of errors (strand losses, insertion/deletion/substitution errors, permutation noise, strand breaks).

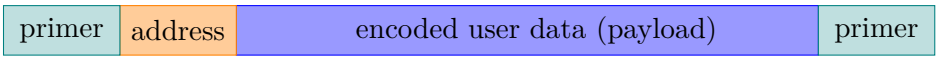


Figure 2.2: Typical structure of a stored DNA strand.

Additional logical redundancy may be used for constrained coding, avoiding certain error-prone structures. In the next step, primer strands, i.e., biological marker sequences, are added at the beginning and end of the strands, which are necessary for certain biological processes, e.g., for PCR and sequencing. We illustrate the typical structure of a DNA strand in Figure 2.2. The encoded sequences are then physically created by the chemical process of *DNA synthesis*. Currently, phosphoramidite-based synthesis is most commonly applied, which, in short, stitches the DNA strands together nucleotide by nucleotide. Moreover, array-based systems enable the creation of many physical copies (in the order of thousands or millions) of each DNA strand in parallel, creating extra physical redundancy. However, the amount of copies statistically varies and depends on different factors, e.g., strand structure or position on the array [DPG+22].

Storing process: The DNA strands are stored as a pool in an unordered fashion either by injecting them into living cells (in vivo storage) or in a physical storage container (in vitro storage), protecting it against possible external harms and slowing down the decay of the DNA. When performing in vitro storage, PCR is typically applied to further amplify the number of DNA strands available in the pool. This can be performed at any time, e.g., directly after writing, during storage, or simply before reading to facilitate data access. Due to PCR's statistical nature, this exhibits different amounts of amplification across the individual strands, which, in extreme cases, results in the total dilution of a particular DNA strand (simply by not getting copied enough).

Retrieval process: A likely scenario is that only a tiny fraction of the data needs to be accessed, e.g., a specific book from an extensive library. Hence, random-access techniques are required to facilitate the efficient retrieval of arbitrary data parts. The most common techniques use specifically designed primer sequences to differentiate files but use different techniques for retrieval. One method uses primer-targeted PCR to amplify the targeted DNA strands and dilute undesired DNA strands, making it unlikely to read non-file-related DNA strands. Other methods physically separate the selected DNA strands based on magnetism or fluorescence-sorting methods to enable correct file retrieval to circumvent the PCR-induced bias in strand distribution.

Reading process: For data readout, a portion of the DNA strands is given to a *DNA sequencing* machine, which translates the available DNA strands to noisy reads, i.e., noisy quaternary sequences comprised again by symbols of $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, of the original written sequences. There are two leading sequencing technologies by *Illumina* and *Oxford Nanopore*. Illumina is a "sequencing by synthesis" method, where the DNA strands get fixed to a board. Then, fluorescent markers cyclically attach and detach, revealing, subsequently, the nucleotides of the DNA strands at their respective positions. This process's main advantage is its high accuracy and the fact that it can be highly parallelized. The disadvantages are the cost of the equipment and latency since the reads get assembled in a non-streamline fashion. Oxford Nanopore is based on nanopore sequencing, where each DNA strand to be translated is pulled through a tiny pore. At the pore, fluctuations of electronic current are measured,

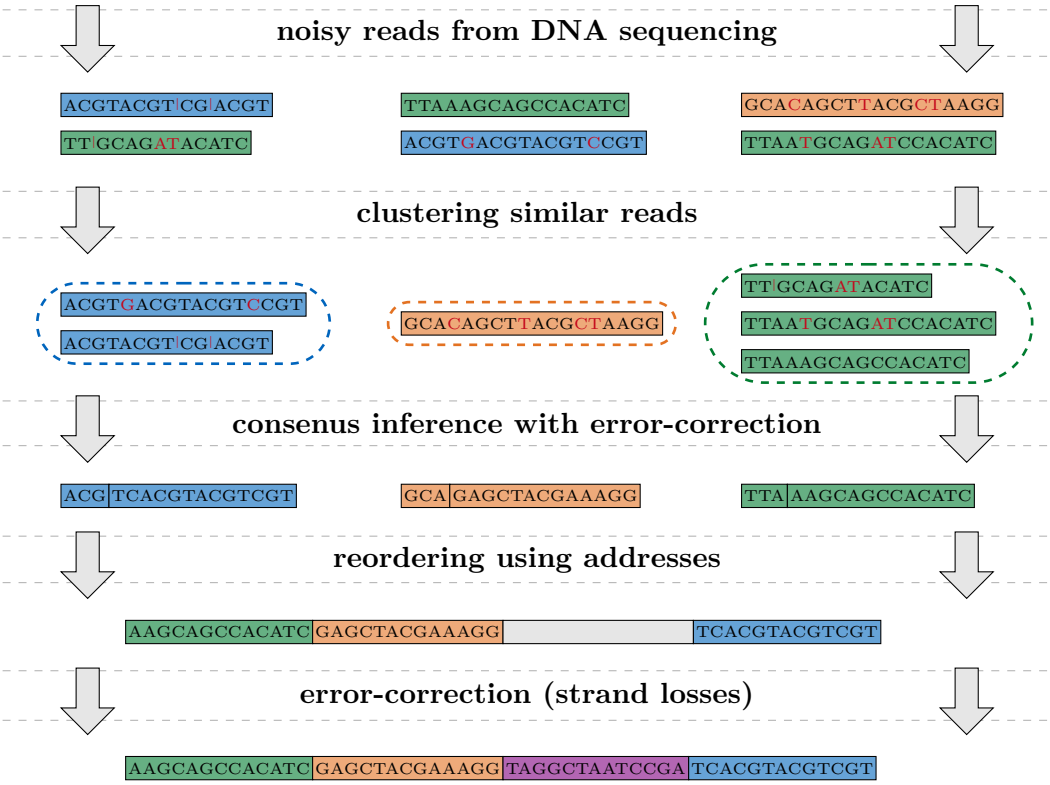


Figure 2.3: Typical decoder flow.

which depend on the present nucleotide (and unintentionally on the surround nucleotides) pushing through the pore, indicating the sequence of nucleotides. In contrast to the other method, this facilitates online reading. Moreover, the equipment is cheap; however, the technique suffers from lower accurate reads, i.e., higher error rates. Nonetheless, the DNA reads contain varying numbers of noisy copies of the original stored DNA strands after DNA sequencing, e.g., a fraction of stored strand is not even within the retrieved portion. Subsequently, the *decoder*'s task is using an algorithm with feasible complexity to retrieve the original data. The decoder leverages the structure introduced by the encoder and the availability of physical redundancy due to the multiple copies to tackle the negative effects of the storage channel. A typical decoder flow is illustrated in Figure 2.3. Given multiple noisy reads affected by insertion and deletion errors, the decoder might first cluster DNA reads based on their similarity. Then, for each cluster, the original stored DNA strand is inferred given the (multiple) noisy reads belonging to the cluster. Subsequently, the data region of the estimated DNA strands are stitched together according to their address information. Strand losses, meaning the original stored strands which do not appear in the portion of drawn noisy reads, are handled by leveraging the logical redundancy incorporated in the data part.

2.1.2 Review of DNA-Based Data Storage Experiments

This section briefly reviews DNA storage experiments with a coding-theoretic view. A more comprehensive version is given in [YKGR$^+$15; CNS19; DSP$^+$20; HSA23; XLC$^+$23; SKS$^+$24].

The discovery of the molecular DNA structure in 1953 [WC53] initiated the first visions of using DNA as a storage medium guided by three scientists Feynman, Wiener, and Neiman [Fey59; Wie64; Nei64] shortly after in 1959 and early 1960s, respectively. Later, [Bau95] manifests the theory of DNA storage, e.g., by introducing relations between information bits and DNA nucleotides, and discussing the availability of extra physical redundancy by having multiple copies of DNA strands to retrieve the stored information. Nonetheless, DNA storage was first demonstrated experimentally in 1988 in the *Microvenus* bio-artwork in which synthetic DNA was implemented in living bacteria [Dav96]. Several other small-scale (storage amount in the order of bytes) experiments demonstrated *in vivo* DNA storage, i.e., DNA storage within living organisms, e.g., [Kac00; WWF03; MKG03; AO04; YSS$^+$07; Gus09; GGL$^+$10]. That being said, in this review, we focus on *in vitro* DNA storage, i.e., stored within an appropriate physical medium. Motivated by a cryptographic application, i.e., hiding secret messages by making them very small compared to a larger carrier (microdots), [RB99] conducted 1999 the first proof-of-concept experiment without living cells. A short secret message was encoded in DNA, camouflaged within the human genome, and then wrapped in an emulsion and droplet onto paper as a microdot. In 2001, the work in [BBB$^+$01] addressed the longevity of DNA if stored in an appropriate medium, that research will always be concerned with DNA since it is the human genetic material (hence DNA technology never gets redundant), and that reading errors can be mitigated by easy replicability of DNA nucleotides. They accompanied these claims with a prototype experiment, storing the two opening lines of the novel *A Tale of Two Cities*. Moreover, the authors pointed out that DNA storage's potential is restricted more by practical challenges than theoretical limits, which is still valid in the 2020s today, though indeed highly advanced. The first advanced error-detection (not correction) methods specially designed for DNA storage (without experiment) were discussed in [SFH$^+$03] introducing a special Huffman source coding tailored to DNA storage (e.g., avoiding specific nucleotide orders), comma codes which are marker codes (one nucleotide is reserved for synchronization), and alternating codes to avoid specific error-prone structures.

In the following decade, the 2010s, large-scale experiments emerged, kick-started by "next-generation" (at the time) DNA synthesis and sequencing technologies. This is mainly driven by the two experiments [CGK12] and [GBC$^+$13] showing the pure potentials of DNA storage. In 2012, [CGK12] stored 0.66 Megabyte (MB) consisting of text, images, and JavaScript source code and retrieved only with 10 bits of error. The highest amount of data stored until then has been approximately 1 kilobyte (kB) in [GGL$^+$10] (in vitro). The authors split the data bitstream into 54 898 chunks. They appended an index represented in bits to each chunk, allowing the synthesis of short strands yielding high-quality DNA strands while ensuring reordering when reading. Then, each bit is encoded into a nucleotide, i.e., for a bit 0 encoding randomly to A or C and bit 1 randomly to G or T. This incurs a significant redundancy cost; however, it allows for constraining the DNA strands to avoid error-prone nucleotide sequences. Due to the attachment of primer pairs to the stored strands, the whole pool of strands could be amplified by PCR, yielding, on average, 3000 copies per

stored strand. Reading uses only the correct length of DNA reads and generates consensus estimates of the stored DNA strands with multiple sequence alignment (MSA) tools. Shortly after, in 2013, [GBC+13] reported the first successful error-free storage of 0.75 MB consisting of text, PDF documents, audio, and image files, albeit with a significant increase in logical redundancy due to implemented error-correction mechanism. The authors used a Huffman-like source coding approach to translate the data into a ternary sequence, which is directly translated by a fixed look-up table to a nucleotide data sequence, both steps specifically designed to avoid error-prone DNA sequences. This sequence is split into overlapping DNA strands so that each data nucleotide will be stored four times, mirroring a repetition code spread over multiple DNA strands. Moreover, the authors are the first to employ an error-detection mechanism by adding a parity symbol to each stored DNA strand's index.

The first sophisticated error correction was employed in [GHP+15] using a concatenated coding system of Reed-Solomon codes. The inner code was used to correct several local errors in the individual DNA strands, and the outer code was used to correct residual errors and lost strands (erasure decoding). The authors proposed silica as a storage medium and simulated 2000 years of storage (aging by heating the DNA), demonstrating error-free storage retrieval thanks to the error-correction mentioned above. In 2015, [YYM+15] addressed two major challenges: random-access and rewritability. Both techniques prevent the need to process all stored DNA strands since random-access enables access to specific data segments, and rewriting enables directly altering segments. Concurrently, random-access is also discussed in [BLC+16]. Both works solve the task similarly by flanking DNA strands belonging to individual data segments, e.g., different files, with specifically tailored primer strands. Recall that by applying primer-targeted amplification of the selected strands, the non-interested strands get diluted, being subsequently unlikely to be read. In 2016, [BGH+16] presented a sophisticated concatenated coding scheme. The main contribution is to protect the index with a low-rate code and an error-detection on the DNA strand using a cyclic-redundancy-check (CRC), emphasizing the reordering of the strands. Additionally, they are the first to deal with insertion/deletion errors employing a modulation code in contrast to simple MSA decoding tools. Inspired by the usage exclusive-or parity strands as error-correction/detection in [BLC+16], the work [EZ17] stored 2.14 MB using Fountain codes as outer codes achieving high information density. However, this was attainable due to low channel noise in the experiment. Later, in 2017, [YGM17] presented the first successful experiment using nanopore sequencing. This technology is prospected to provide lower-cost sequencing, better portability (as the device technology is smaller), and enables long reads at the expense of 10-20 times higher channel noise (at the time). A massive increase in storage amount was achieved by [OAC+18] in 2018, storing 200.2 MB using around 13 million DNA strands, remaining the largest experiment today. The authors also present a method to construct primer pairs for random-access using primer-targeted PCR. Moreover, they focus on minimizing the number of DNA reads to ensure error-free decoding. Expanding on this, [CTL+19] connects the cost of writing, i.e., the additional redundant nucleotides needed (and thus to be synthesized) to encode ensuring error-detection/-correction/-avoidance, and the cost of reading, i.e., the amount of DNA reads needed to be read (providing physical redundancy at the cost of sequencing) to ensure error-free decoding. Further, they use low-density parity-check codes instead of Reed-Solomon or Fountain codes as outer codes and a simple inner marker code. The work

of [LCA+19] uses chemical attachment strategies to glue stored DNA strands together before reading to enhance the efficiency of the nanopore sequencer by using longer reads. The work of [AVA+19] leveraged the array-synthesis methodology, which simultaneously writes thousands of the same DNA strands. Instead of synthesizing the same nucleotide within an array cell each time, they allow different nucleotides, albeit in fixed compositions, to attach in each synthesis cycle. Consequently, they effectively encode into a higher-order alphabet, increasing the theoretical limit of storage capacity by cleverly leveraging the array-synthesis procedure. Focused on leveraging the nanopore sequencing output, [CNT+20] presents an approach to use the soft information on the nucleotides provided by the nanopore basecaller within a Viterbi decoder of the employed inner convolutional code. A low-cost synthesis method is investigated in [ALD+20], which comes at the price of high channel noise incurred at the writing step. Thus, it requires higher error-protection schemes. Random-access based on primer-targeted PCR requires additional nucleotide sequences to be attached to each DNA strand. In 2021, [BSB+21] proposes a method that relies on file-specific encapsulation of the DNA strands, which additionally protects the DNA, together with fluorescence-activated sorting for file access using boolean-search logic. Due to the highly stochastic nature of the DNA storing process (writing, PCR, sequencing, ...), the present decoders are limited due to a model mismatch. More precisely, the decoder assumes an auxiliary channel model instead of the true unknown underlying model due to impracticability or intractability of channel parameters, or simply due to high resulting decoding complexity when decoding optimally. Hence, data-driven machine learning algorithms for decoding have been proposed and experimentally validated in [BLOS+21] and in [PTY+22], the latter purely focused on image storage. In 2023, [LCR+23] presented a random-access method where DNA strands are separated using magnetic beads, which can also enable repurification, thus providing repeated high-quality reads of the original stored DNA.

2.2 Notation

This section formally defines the general notation that is used throughout the dissertation. Content-specific notations are introduced in the respective chapters.

We denote $\mathbb{Z}$ as the infinite set of all integers and $\mathbb{N}, \mathbb{N}_0$ as the infinite sets of all positive integers, where the latter one includes zero. We write $\mathbb{R}$ for the set of real numbers and $\mathbb{R}^+$ for the subset of positive real numbers. For two integers $i, j \in \mathbb{N}$ such that $i \leq j$ the set $\{i, i+1, \ldots, j\}$ is denoted by $[i, j]$ and in short $[j]$ if $i = 1$. We define an alphabet as $\Sigma_q = \{0, \ldots, q-1\} \subset \mathbb{N}_0$ of cardinality $|\Sigma_q| = q$, e.g., the binary alphabet $\Sigma_2 = \{0, 1\}$. The DNA alphabet is denoted as $\Sigma_{\mathrm{DNA}} = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, where a one-to-one mapping to Σ_4 can be assumed. The set of all vectors of length n formed with symbols from Σ_q is denoted as Σ_q^n and the set of arrays of dimension $m \times n$ over Σ_q as $\Sigma_q^{m \times n}$. We denote $\Sigma_q^\star = \cup_{i=0}^{\infty} \Sigma_q^i$. We denote generic sets with calligraphic letters $\mathcal{A}$, vectors with bold lower case $\mathbf{x} = (x_1, \ldots, x_n) \in \Sigma_q^n$, and arrays with upper case bold letters $\mathbf{X} \in \Sigma_q^{m \times n}$. With $|\mathcal{A}|$ we refer to the cardinality of the set $\mathcal{A}$. The transpose of an array $\mathbf{X}$ is denoted as $\mathbf{X}^T$. We denote a substring of a vector $\mathbf{x} \in \Sigma_q$ from index a up to index b as $\mathbf{x}_a^b$. For a bit x we denote its complement as $\overline{x}$. Similarly, for a binary vector $\mathbf{x} \in \Sigma_2^n$, we denote by $\overline{\mathbf{x}}$ the complement of $\mathbf{x}$, i.e., every bit is flipped. Further, given a vector $\mathbf{x} \in \Sigma_q^n$ a run of length r in $\mathbf{x}$ is defined as a sequence

of r consecutive equal symbols $x_i = x_{i+1} = \ldots = x_{i+r-1}$. Consequently, $r(\mathbf{x})$ denotes the number of runs in $\mathbf{x} \in \Sigma_q^n$. For a prime power q, let $\mathbb{F}_q$ denote the finite field of size q, $\mathbb{F}_q^n$ the vector space of length n over $\mathbb{F}_q$, and the $\mathbb{F}_q^{m \times n}$ the matrix space over $\mathbb{F}_q$ of dimension $m \times n$.

We denote the logarithm of a real number $a \in \mathbb{R}$ with respect to base $b \in \mathbb{R}^+$ as $\log_b(a)$. For $n, m \in \mathbb{N}$ with $m \leq n$, we write $n! = n \cdot (n-1) \cdot \ldots \cdot 1$ as the factorial and $\binom{n}{k} = \frac{n!}{(n-m)!m!}$ as the binomial coefficient.

We write random variables (RVs) with sans-serif letters X and random vectors with sans-serif bold letter $\mathbf{X}$, where their realizations are written as x and $\mathbf{x}$, respectively. We denote as $\mathbb{P}(\mathsf{X})$ the probability mass function of the discrete RV X, and abbreviate the probability of a particular event $\mathbb{P}(\mathsf{X} = x)$ simply by $\mathbb{P}(x)$. For two RV X, Y, we denote the joint distribution as $\mathbb{P}(\mathsf{X}, \mathsf{Y})$ and the conditional distribution as $\mathbb{P}(\mathsf{Y}|\mathsf{X})$. We write $\mathbb{E}_{\mathsf{X}}[f(\mathsf{X})]$ as the expectation of a real valued function with respect to the probability distribution of the RV X. Similarly, we denote $\mathbb{V}_{\mathsf{X}}[f(\mathsf{X})]$ as the variance. We denote $\mathrm{H}(\mathsf{X})$ as the entropy, $\mathrm{H}(\mathsf{X}|\mathsf{Y})$ as the conditional entropy, and $\mathrm{I}(\mathsf{X};\mathsf{Y})$ as the mutual information.

Throughout this dissertation, we write $f(n) \approx g(n)$, $f(n) \lesssim g(n)$, and $f(n) \gtrsim g(n)$ if the equality or inequality holds for $n \to \infty$. Further, we write $f(n) \propto g(n)$ if the two functions are proportional, i.e., if their ratio is constant. Additionally, we use the Bachmann-Landau notation, i.e., we write $f(n) \in o(g(n))$ if *for all* $c \in \mathbb{R}^+$ and given the existence of a value $n_0 > n$, it holds that $|f(n)| \leq c \cdot |g(n)|$ for all $n > n_0$, and we write $f(n) \in O(g(n))$ if *there exists* a $c \in \mathbb{R}^+$ and a value $n_0 > n$ such that it holds $|f(n)| \leq c \cdot |g(n)|$ for all $n > n_0$.

2.3 Principles and Challenges of Reliable Data Storage

In communication and storage scenarios, the objective is to transmit information reliably over space (communication) or over time (storage) despite natural distortion. To ensure this, it is crucial to mathematically quantify the information to be transmitted and the noise process responsible for the data changes. Then, the design of algorithms and mathematical concepts are needed, which alter the data by adding intelligently and efficiently redundancy to the data that is to be transmitted, enabling the error-free reconstruction of the data after transmission, even in the presence of errors. The scientific field that addresses this issue, among others, is called information theory and coding theory. The research was kicked off by Shannon in 1949 [Sha48] and Hamming [Ham50] in 1950. For a broad overview, we refer to the books on this topic [CT06; RL09; Rot06]. For DNA storage-oriented introduction on error-correction, we refer to [SH22; Sim22; Len22; SKS+24].

This section presents the main information- and coding-theoretic concepts needed for this dissertation. We start by presenting the essential components of a storage/communication process, as considered in this dissertation. Then, we briefly discuss the two main modeling approaches and address possible error types.

2.3.1 Transmission Models

The transmitter desires to reliably send the receiver one of K possible messages. We interpret the message as an information vector $\mathbf{u} \in \Sigma_q^k$ such that $k = \log_q(K)$.

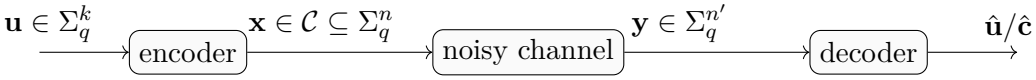


Figure 2.4: Basic transmission concept.

An *encoder* assigns according to a bijective mapping $\textsc{Enc} : \Sigma_q^k \to \mathcal{C}$ the vector $\mathbf{u}$ to a vector $\mathbf{x} \in \mathcal{C} \subseteq \Sigma_q^n$, where $\mathcal{C}$ is called a *code* (see Definition 2.1). The vector $\mathbf{x} = \textsc{Enc}(\mathbf{u})$ is then transmitted via the *noisy channel* inflicting transmission errors outputting a received vector $\mathbf{y} \in \Sigma_q^{n'}$, where n' is not necessarily equal to the input length n. At the receiver side, the *decoder*'s task is, given the channel law, the code $\mathcal{C}$, and the channel output $\mathbf{y}$, to output an estimate of the information vector $\hat{\mathbf{u}}$, or an estimate $\hat{\mathbf{c}}$ due to the bijective mapping of the encoder. Hence, the *decoder* is a mapping $\textsc{Dec} : \Sigma_q^{n'} \to \mathcal{C}$ with the objective of reliably achieving $\mathbf{x} = \hat{\mathbf{x}}$. For the sake of presentation, we fixed the encoder's, channel's, and decoder's input/output alphabet to be the same which is not necessarily true. We have depicted the transmission model in Figure 2.4.

The essence of providing reliable communication over the channel lies in the design of the code $\mathcal{C}$, defined in the following.

Definition 2.1. *A code is a non-empty subset $\mathcal{C}$ of Σ_q^n.*

We call n the *length*, $|\mathcal{C}|$ the *cardinality*, and $k = \log_q(|\mathcal{C}|)$ the *dimension* of the code. Further, we denote as $R = \frac{k}{n}$ the *rate* and as $r = n - k = n - \log_q(|\mathcal{C}|)$ the *redundancy* of the code. Elements of the code $\mathbf{x} \in \mathcal{C}$ are called *codewords*. We denote a code $\mathcal{C} \subseteq \Sigma_q^n$ as $[n, k]_q$ when the code parameters are not directly obvious. *Concatenated codes* consist of multiple component codes combined to enhance the overall error-correction capability [For65]. In particular, we consider in this thesis serial concatenated codes, where the input data is serially encoded with each component code. Further, each code may also be tailored to tackle different aspects of the noise introduced by the channel.

The dissertation considers two types of noisy channels and decoding scenarios.

1. **Adversarial model**
   The channel inflicts a fixed number of errors in an arbitrary pattern. Our objective is to minimize the amount of redundancy r such that any pattern is always correctly decodable, also in the worst-case scenario. (see Part I)

2. **Stochastic model**
   The channel follows a probabilistic channel law given by a conditional probability distribution $\mathbb{P}(\mathbf{Y}|\mathbf{X})$. In this scenario, our main objective for a fixed channel law is to maximize the rate R when achieving vanishing decoding error probability on average. (see Part II)

In general, several quantities in both scenarios can be investigated. In this dissertation, we focus on quantifying theoretical or simulation-based bounds on the abovementioned parameters, rate and redundancy, and construct codes $\mathcal{C}$ together with decoders attempting to converge to these bounds for certain types of noisy channels. Before going in-depth about the two scenarios, we discuss the possible error types treated in the dissertation.

2.3.2 Error Types

We consider four basic error types, which can be divided into two classes: *memoryless errors* considering *erasures* and *substitutions*, and *synchronization errors* considering *insertions* and *deletions*. The main difference between the two classes is that memoryless errors induce no further implication on other symbols, e.g., changing the position information of the transmitted symbols.

Assume the vector $\mathbf{x} = (x_1, \ldots, x_n) \in \Sigma_q^n$ is transmitted and the vector $\mathbf{y} = (y_1, \ldots, y_{n'})$ is received. We explain in the following the different error types:

1. **Erasure:**
   The symbol information of $x_i \in \Sigma_q$ in $\mathbf{x} \in \Sigma_q^n$ is lost. That is indicated by substituting x_i with an additional symbol ?, resulting at the receiver in the vector
   $$\mathbf{y} = (x_1, \ldots, x_{i-1}, ?, x_{i+1}, \ldots, x_n) \in \{\Sigma_q \cup \{?\}\}^n \ .$$

2. **Substitution:**
   The symbol x_i is substituted by $a \in \Sigma_q \setminus \{x_i\}$ resulting in the received vector
   $$\mathbf{y} = (x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) \in \Sigma_q^n \, .$$

3. **Deletion:**
   A deletion of the symbol x_i at the i^{th} position results in the vector
   $$\mathbf{y} = (y_1, \ldots, y_{n-1}) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \in \Sigma_q^{n-1} \, .$$

4. **Insertion:**
   An insertion of a symbol $a \in \Sigma_q$ at the i^{th} position in $\mathbf{x}$ results in the received vector
   $$\mathbf{y} = (y_1, \ldots, y_{n+1}) = (x_1, \ldots, x_{i-1}, a, x_i, \ldots, x_n) \in \Sigma_q^{n+1} \, .$$

We observe that in case of an erasure simply the value information of the symbol is lost, but all symbol position information is unaffected. In case of a substitution error, for a decoder, the error position is not directly apparent, and on top of that, the correct value is to be found. However, there is no implication for other symbols. A deletion removes the value of the affected symbol and alters the positional information of any subsequent symbol, i.e., it shifts all symbols one position towards the start of the vector. An insertion does not alter the symbol at the affected position, but it shifts all trailing symbols by one position towards the end of the vector, including the symbol at the respective position. We provide an example for each error type in Figure 2.5.

In this dissertation, we will consider combinations of the aforementioned error types. In addition to the term *synchronization error* which refers to an insertion or deletion, the literature uses other terminology for a specific combination of error types as follows.

1. **Insdel:**
   An *insdel error* is defined to be either an insertion or a deletion error.

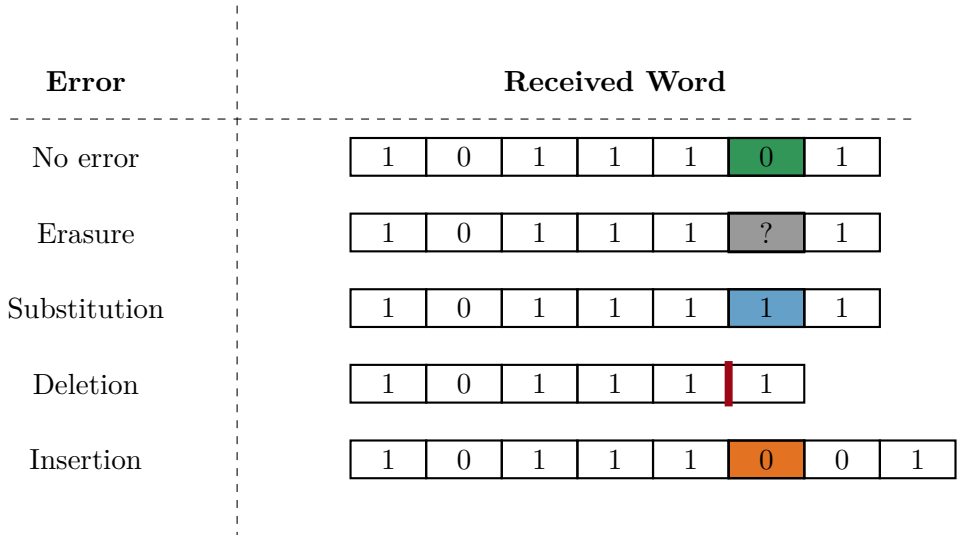


Figure 2.5: Given a transmission of the vector $(1,0,1,1,1,0,1)$, the different error types are illustrated by providing a possible received vector.

2. **Edit:**
   An *edit error* is defined to be an insertion, a deletion, or a substitution error.

Furthermore, we define three different distance metrics that relate two vectors $\mathbf{x}, \mathbf{y} \in \Sigma_q^*$.

Definition 2.2 (Hamming distance)**.** *For two equal length vectors $\mathbf{x}, \mathbf{y} \in \Sigma_q^n$, we define the* Hamming distance $\mathrm{d}_{\mathsf{H}}(\mathbf{x}, \mathbf{y})$ *as the minimum number of substitutions to transform $\mathbf{x}$ into $\mathbf{y}$.*

Definition 2.3 (Levenshtein distance)**.** *For two arbitrary length vectors $\mathbf{x}, \mathbf{y} \in \Sigma_q^*$, we define the* Levenshtein distance $\mathrm{d}_{\mathsf{L}}(\mathbf{x}, \mathbf{y})$ *as the minimum number of insertion and deletions to transform $\mathbf{x}$ into $\mathbf{y}$.*

Definition 2.4 (Edit distance)**.** *For two arbitrary length vectors $\mathbf{x}, \mathbf{y} \in \Sigma_q^*$, we define the* edit distance $\mathrm{d}_{\mathsf{E}}(\mathbf{x}, \mathbf{y})$ *as the minimum number of insertion, deletions, and substitutions to transform $\mathbf{x}$ into $\mathbf{y}$.*

We remark that for two vectors $\mathbf{x}, \mathbf{y} \in \Sigma_q^n$, a large Hamming distance does not imply as well a large Levenshtein (or Edit) distance as the following example shows, or vice versa.

Example 2.1. *Let $\mathbf{x} = (0,1,0,1,\ldots)$ and $\mathbf{y} = (1,0,1,0,\ldots)$, both of length n, we have $\mathrm{d}_{\mathsf{H}}(\mathbf{x}, \mathbf{y}) = n$ and $\mathrm{d}_{\mathsf{L}}(\mathbf{x}, \mathbf{y}) = 2$.*

The field of error correction codes for erasure and substitution errors has been extensively studied. For these types of errors, classes of *linear codes* are predominantly used due to their good analytical properties and existing efficient encoding and decoding algorithms.

Definition 2.5 (Linear Code)**.** *A linear code is a k-dimensional subspace of a vector space over $\mathbb{F}_q^n$.*

Examples for widely used linear codes are the *Cyclic-Redundancy-Check Code*, *Hamming Code*, *Reed-Solomon Code*, *Bose-Ray-Chaudhuri-Hocquenghem Code*, *Convolutional Codes*, *Low-Density-Parity-Check Code*, *Polar Code*, etc. We refer to [Rot06; RL09] for a fundamental overview. In the chapters of this work, we will introduce the theory of synchronization-error-correction in both the adversarial and stochastic channel scenarios. We follow the literature by using the terminology *synchronization errors* only in the field of stochastic models and refer to these errors simply as insertion/deletion errors in the adversarial modeling.

Part I

Codes for Adversarial Insertion/Deletion Channels

Chapter 3

Preliminaries on Insertion/Deletion Error-Correction for Adversarial Channels

This part of the dissertation discusses novel channel coding techniques, including insertion and deletion errors on specific adversarial channels, which will be discussed thoroughly in the respective chapters. Therefore, first, we present basic principles on error-correction for insertion/deletion errors in an adversarial setting.

In the insertion/deletion adversarial channel, the codeword $\mathbf{x}$, which is an element of a code $\mathcal{C} \subseteq \Sigma_q^n$ is inflicted with (at most) t insertion/deletion errors by the channel outputting the noisy version $\mathbf{y} \in \Sigma_q^{n'}$. In this dissertation, we consider unique decoding under the nearest-codeword decoding-principle [Rot06], i.e., a decoder decides on the estimate $\hat{\mathbf{x}}$ given $\mathbf{y}$ and $\mathcal{C}$ according to the rule

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathcal{C}} \mathrm{d}_{\mathsf{L}}\left(\mathbf{x}, \mathbf{y}\right) .$$

We denote as $\mathbb{D}_t(\mathbf{x}) \subseteq \Sigma_q^{n-t}$ and $\mathbb{I}_t(\mathbf{x}) \subseteq \Sigma_q^{n+t}$ the sets of all vectors which can be obtained from $\mathbf{x} \in \Sigma_q^n$ by t deletions or t insertions, respectively. Similarly, we denote $\mathbb{ID}_t(\mathbf{x}) \subseteq \bigcup_{t_{\mathrm{I}},t_{\mathrm{D}}\in\mathbb{N}_0:t_{\mathrm{I}}+t_{\mathrm{D}}=t} \Sigma_q^{n+t_{\mathrm{I}}-t_{\mathrm{D}}}$ as the set of all vectors which can be obtained from $\mathbf{x} \in \Sigma_q^n$ by t_{I} insertions and t_{D} deletions such that $t_{\mathrm{I}} + t_{\mathrm{D}} = t$. We call these sets the deletion, insertion, or insdel *ball*, respectively.[1]

We define a t-deletion correcting code according to the nearest-codeword decoding rule in the following.

Definition 3.1 ([Lev66b])**.** *A code $\mathcal{C}_{q,n,t} \subseteq \Sigma_q^n$ is a t-deletion correcting code if for all pairs $\mathbf{x}, \mathbf{y} \in \mathcal{C}_{q,n,t}$, $\mathbf{x} \neq \mathbf{y}$, it holds that*

$$\mathrm{d}_{\mathsf{L}}\left(\mathbf{x}, \mathbf{y}\right) > 2t\,,$$

or rephrased

$$\mathbb{D}_t(\mathbf{x}) \cap \mathbb{D}_t(\mathbf{y}) = \emptyset\,.$$

[1] Strictly speaking, the sets $\mathbb{D}_t(\mathbf{x})$, $\mathbb{I}_t(\mathbf{x})$, and $\mathbb{ID}_t(\mathbf{x})$ must be called *spheres*. However, we abuse terminology and refer to these sets as balls to follow the terminology used by the literature on deletion-correcting codes.

3.1 Review on Insertion/Deletion Correcting Codes

The algebraic concepts correcting insertions and deletions date back to the 1960s [Gil60; Sel62; Lev66b; Ull66; Ull67; CH69]. In his seminal work [Lev66b], Levenshtein first defined the notion of t-insertion/deletion correcting codes. He proved the insertion/deletion correcting code equivalence (see Section 3.2). Moreover, he presented theoretical bounds for the cardinality of deletion correcting codes, showing the optimal redundancy to correct t deletions is asymptotically $t\log_2(n)$ (see Section 3.3). Most importantly, he showed that the binary Varshamov-Tenengolts codes (VT codes) [VT65], which are number-theoretic codes, can correct a single deletion error with asymptotic optimal redundancy. VT codes were originally designed to correct asymmetric errors. Levenshtein presented a similar number-theoretic construction based on encoding runs of bits instead of single bits in [Lev66a]. Tenengolts presented close to optimal q-ary single-insertion/deletion correcting codes in [Ten84] by applying the VT code on a binary signature of the q-ary vectors and using an additional parity constraint. Solving effciently (in terms of redundancy) the multiple insertion/deletion correcting problem remained challenging for a long time. Helberg presented explicit multiple insertion/deletion correcting codes [HF02; AGPF$^+$11] based on VT-like constraints with $O(n)$ redundancy.

In [SZ99], a binary concatenated coding scheme constituting of classic outer codes (e.g., Reed-Solomon codes) and short inner codes is presented aimed to correct a constant fraction of worst-case deletions asymptotically well. By inserting known marker sequences, the outer codeword is separated into small blocks, where each block is protected by a short, greedily constructed multiple-deletion correcting code. This was later improved by [GW17] using the same principle idea. This construction could also be adapted to correct a constant number of t deletions, achieving redundancy $O(\sqrt{n})$.

Approaching Levenshtein's bound was successful in [BGZ17] achieving $O(t^2\log_2(n))$ redundancy. The authors build upon the aforementioned concatenated coding scheme; however, they use patterns within the vectors as markers, ensuring the existence of such patterns using efficient pattern-enriching methods. For the particular case of $t = 2$, we have three important binary constructions achieving redundancy of $8\log_2(n)$[GS18], $7\log_2(n)$ [SRB19], and $4\log_2(n)$ [GH21]. The work in [GS18] improved the techniques in [BGZ17]. A number-theoretic approach based on so-called *VT-sketches* is used in [SRB19], refined in [GH21] with additional run encoding similar as in [Lev66a]. For arbitrary t, the first order-optimal binary construction with redundancy $O(t\log_2(n))$ was presented in [CJL$^+$18] in the context of document exchange protocols. Sima et al. generalized their binary two deletion construction to arbitrary t in their seminal work in [SB20] achieving $8t\log_2(n)$ redundancy. By using only a single pattern and applying multiple VT-sketches on a pattern-indicator function, the authors improved the concatenated construction from [BGZ17]. Using the syndrome compression technique introduced in [SGB20c], one can construct systematic binary deletion codes [SGB20b] and non-systematic q-ary deletion codes [SGB20a] ($q < k$, k is the message length) with $4t\log_q(n)$ redundancy. Using precoding techniques, Song et al. extended the non-binary construction to arbitrary q [SPC$^+$21] and improved the systematic binary construction in [SPC$^+$22] to $(4t-1)\log_2(n)$ bits. The idea of *differential VT codes* has been introduced in [NCS24], providing a new way of constructing non-binary single deletion codes

by encoding the difference of the non-binary symbols.

For a more extensive overview of insertion/deletion correcting codes, we refer to the surveys [MBT10; HS21].

3.2 Insertion and Deletion Equivalence

In his seminal work, Levenshtein proved the following equivalence of error-correcting codes.

Theorem 3.1 ([Lev66b])**.** *A code $\mathcal{C}_{q,n,t} \subseteq \Sigma_q^n$ is t-deletion correcting code if and only if it is a t-insdel correcting code.*

We can reformulate this statement to the following observation using error balls.

Lemma 3.1 ([KK13; CK14])**.** *The following statements are equivalent for any pair $\mathbf{x}, \mathbf{y} \in \Sigma_q^n$.*

$$\begin{aligned} & \mathrm{d}_{\mathsf{L}}(\mathbf{x}, \mathbf{y}) \leq 2t \\ \iff & \mathbb{D}_t(\mathbf{x}) \cap \mathbb{D}_t(\mathbf{y}) \neq \emptyset \\ \iff & \mathbb{ID}_t \cap (\mathbf{x}) \cap \mathbb{ID}_t(\mathbf{y}) \neq \emptyset \\ \iff & \mathbb{I}_t(\mathbf{x}) \cap \mathbb{I}_t(\mathbf{y}) \neq \emptyset\,. \end{aligned}$$

Hence, in the adversarial channel setting, we can focus on designing deletion-correcting codes and claim the same code properties for insdel correcting codes (or insertion correcting codes), and vice versa, simplifying the analysis and construction of codes. Hence, we focus on t-deletion correcting codes in the following.

3.3 Lower Bound on the Redundancy of Codes

We can quantify the performance of a specific t-error correcting code by deriving upper bounds on the cardinality of general t-error correcting codes using sphere-packing techniques with the error balls on the receiving space [Rot06; KK13]. An upper bound on the cardinality directly implies a lower bound on the redundancy, i.e., the number of symbols we need to add to ensure the error-correction capability. In short, the methodology counts how many disjoint error balls can be packed in the space of all possible words in the receiving space. For t deletion erros the receiving space would be Σ_q^{n-t} and for t insertion errors it would be Σ_q^{n+t}. One important observation is that for any $\mathbf{x} \in \Sigma_q^n$ the size of the deletion ball $\mathbb{D}_t(\mathbf{x})$ varies depending on $\mathbf{x}$ (thus also the insdel ball). We illustrate this in the following example.

Example 3.1. *For $\mathbf{x} = (0,0,1,1,1,0)$ and $\mathbf{y} = (0,0,0,0,0)$, we have*

$$\begin{aligned} &\mathbb{D}_1(\mathbf{x}) = \{(0,1,1,1,0), (0,0,1,1,0), (0,0,1,1,1)\} \text{ and} \\ &\mathbb{D}_1(\mathbf{y}) = \{(0,0,0,0)\}. \end{aligned}$$

For $t > 1$ practical closed form expressions for $|\mathbb{D}_t(\mathbf{x})|$ or $|\mathbb{ID}_t(\mathbf{x})|$ are yet unknown [MKB08]. In the special case of $t = 1$, we have that $|\mathbb{D}_t(\mathbf{x})| = r(\mathbf{x})$ [Lev02], where $r(\mathbf{x})$ denotes the number of runs, i.e., the number of substrings of consecutive identical symbols, in $\mathbf{x}$. In contrast, the cardinality of the insertion ball is independent of $\mathbf{x}$ [KK13]. Specifically, for any $\mathbf{x} \in \Sigma_q^n$ we have $|\mathbb{I}_t(\mathbf{x})| = \binom{n+t}{t}(q-1)^t$. Due to the insertion/deletion equivalence (see Theorem 3.1), one can directly formulate an asymptotic sphere-packing argument using the insertion ball over the space Σ_q^{n+t}. Given the fact that for $n \to \infty$ and t constant respect to n we have $\binom{n+t}{t} \sim \frac{n^t}{t!}$, we can derive

$$|\mathcal{C}_{q,n,t}| \leq \frac{q^{n+t}}{\binom{n+t}{t}(q-1)^t} \sim \frac{t! q^{n+t}}{n^t (q-1)^t} \,.$$

However, this bound is q^t times larger asymptotically than using the generalized sphere-packing bound, i.e., a more sophisticated bounding technique that takes into account the irregular deletion balls and applying it on the space Σ_q^{n-t} [KK13; CK14; FVY15]. We discuss and apply the generalized sphere packing bound in Section 4.3; however, in what follows, we focus on asymptotic statements. We can show using concentration bounds that almost all words have a deletion ball of size at least $\left(\frac{q-1}{q}\right)^t \frac{n^t}{t!}$ for $n \to \infty$ [Lev66b], hence, concluding with the following statement.

Theorem 3.2 ([KK13; Lev02])**.** *For any t-deletion correcting code $\mathcal{C}_{q,n,t}$ it holds that*

$$|\mathcal{C}_{q,n,t}| \lesssim \frac{t!\, q^n}{n^t (q-1)^t} \,.$$

Consequently, we have $r \gtrsim t \log_q(n) + t \log_q(q-1) - \log_q(t!)$.

We call codes that meet this bound *optimal*. Thus, the objective is to construct codes that are close to, approach, or meet this bound. Note that there are also lower bounds on the cardinality of a t-deletion correcting code, so-called Gilbert-Varshamov bounds, we refer to [Lev66b; Lev02; SGD14].

3.4 Code Constructions

In this section, we present three code constructions that are essential for this work. The most prominent insertion/deletion correcting codes are the *VT codes*, which we discuss more thoroughly. We present the construction, two possible ways for their single insertion/deletion error-correction capability, and discuss their cardinality. Moreover, we briefly discuss the non-binary VT codes and Sima et al.'s systematic binary t-deletion correcting code.

Construction 3.1 ([VT65])**.** *For $n \geq 1$ and $a \in [0, n]$, we define the* Varshamov-Tenengolts code *(VT codes) as*

$$\mathcal{VT}_a(n) = \left\{ \mathbf{x} \in \Sigma_2^n \,:\, \sum_{i=1}^{n} i x_i \equiv a \mod n+1 \right\} .$$

We present two different proving techniques, one showing that the codewords have disjoint error balls, the other by provding a correct decoder, which inspired the proving techniques in Chapter 4 and Chapter 5, respectively. We will prove the following statement, where the proofs can be found in [Lev66b; Slo02].

Theorem 3.3. *The VT code in Construction 3.1 is a single-deletion correcting code.*

Proof of codewords having disjoint deletion balls. We need to show that for any distinct codewords $\mathbf{x}, \mathbf{y} \in \mathcal{VT}_a(n)$ we have $\mathbb{D}_1(\mathbf{x}) \cap \mathbb{D}_1(\mathbf{y}) = \emptyset$. The proof proceeds by contraposition, i.e., we assume there exists a $\mathbf{z} \in \mathbb{D}_1(\mathbf{x}) \cap \mathbb{D}_1(\mathbf{y})$. Assume that $\mathbf{z}$ was obtained from $\mathbf{x}$ by deleting the bit x_{d_x} at position d_x and from $\mathbf{y}$ by deleting the bit y_{d_y} at position d_y, and without loss of generality $d_x < d_y$. Hence, we can deduce the following relations between $\mathbf{x}$ and $\mathbf{y}$.

1. For $i \in [1, d_x - 1]$ and $i \in [d_y + 1, n]$, we have $x_i = y_i$.
2. For $i \in [d_x + 1, d_y]$, we have $x_i = y_{i-1}$.

Using these relations and the construction principle of VT codes, we can deduce the following.

$$\begin{aligned} &\sum_{i=1}^{n} i x_i - \sum_{i=1}^{n} i y_i \equiv 0 \mod n+1\,, \\ \iff &\sum_{i=1}^{d_x-1} i x_i + d_x x_{d_x} + \sum_{i=d_x+1}^{d_y} i x_i + \sum_{i=d_y+1}^{n} i x_i \\ &\qquad - \sum_{i=1}^{d_x-1} i y_i - \sum_{i=d_x}^{d_y-1} i y_i - d_y y_{d_y} - \sum_{i=d_y+1}^{n} i y_i \equiv 0 \mod n+1\,. \end{aligned}$$

Applying the relations above and sum reindexing, we arrive at the following expression.

$$\underbrace{d_x x_{d_x}}_{\in\{0,d_x\}} + \underbrace{\sum_{i=d_x+1}^{d_y} x_i}_{\in[0,d_y-d_x-1]} - \underbrace{d_y y_{d_y}}_{\in\{0,d_y\}} \equiv 0 \mod n+1\,. \tag{3.1}$$

Because we consider binary vectors, we obtain the possible values indicated in the brackets below the equation. We make the following four case distinctions:

1. *Case* $x_{d_x} = y_{d_y} = 0$: The equivalence in Equation (3.1) evaluates to

$$\sum_{i=d_x+1}^{d_y} x_i \equiv 0 \mod n+1\,.$$

For the left, it holds that $0 \leq \ldots \leq d_y - d_x - 1 < n + 1$. Moreover, the term only equals to zero if all $x_i = 0$ in the interval $[d_x + 1, d_y]$, which means that both, $\mathbf{x}$ and $\mathbf{y}$, have to be all-zero within the interval $[d_x, d_y]$, hence $\mathbf{x} = \mathbf{y}$. Otherwise, the term does not equal zero, which contradicts $\mathbf{x}, \mathbf{y} \in \mathcal{VT}_a(n)$.

2. *Case* $x_{d_x} = 1$, $y_{d_y} = 0$: The equivalence in Equation (3.1) evaluates to

$$d_x + \sum_{i=d_x+1}^{d_y} x_i \equiv 0 \mod n+1 \,.$$

For the left term it holds that $0 < d_x < \ldots < d_y - 1 < n+1$ violating the equivalence, hence, both $\mathbf{x}$ and $\mathbf{y}$ cannot be distinct codewords.

3. *Case* $x_{d_x} = 0$, $y_{d_y} = 1$: The equivalence in Equation (3.1) evaluates to

$$\sum_{i=d_x+1}^{d_y} x_i - d_y \equiv 0 \mod n+1 \,.$$

For the left term it holds that $-(n+1) < -d_y < \ldots < -d_x - 1 < 0$ violating the equivalence, hence, both $\mathbf{x}$ and $\mathbf{y}$ cannot be distinct codewords.

4. *Case* $x_{d_x} = y_{d_y} = 1$: The equivalence in Equation (3.1) evaluates to

$$d_x + \sum_{i=d_x+1}^{d_y} x_i - d_y \equiv 0 \mod n+1 \,.$$

For the left term it holds that $-(n+1) < -d_y + d_x < \ldots < -1 < 0$ violating the equivalence, hence, both $\mathbf{x}$ and $\mathbf{y}$ cannot be codewords.

Consequently, we have shown that no codewords share an element in their deletion ball. □

Proof using a decoder. We show a decoder that will always correct to the original codeword. Suppose a $\mathbf{z} \in \mathbb{D}_1(\mathbf{x})$ was received from $\mathbf{x} \in \mathcal{VT}_a(n)$ by deleting the bit of value b at position d. We denote L_0 and L_1 as the number of 0's and 1's to the left of the deleted bit, respectively. Similarly, we denote R_0 and R_1 as the number of 0's and 1's to the right. The decoder works as follows.

1. Compute the weight (number of ones) $w = L_1 + R_1$ of $\mathbf{z}$ and compute the difference

$$d = a - \sum_{i=1}^{n-1} i z_i \mod n+1$$

2. If $s = 0$:

$$d = R_1 \leq w < n+1$$

Hence, we insert a 0 after R_1 ones from the right into $\mathbf{z}$ to restore $\mathbf{x} \in \mathcal{VT}_a(n)$.

3. If $s = 1$:

$$d = p + R_1 = 1 + L_0 + L_1 + R_1 = 1 + w + L_0 > w < n + 1$$

Hence, we insert a 1 after L_0 zeroes from the left into $\mathbf{z}$ to restore $\mathbf{x} \in \mathcal{VT}_a(n)$.

The following derivations show the correctness of the decoder. We compute the difference between the transmitted and received vectors.

$$\begin{aligned}
d &= a - \sum_{i=1}^{n-1} i z_i = \sum_{i=1}^{n} i x_i - \sum_{i=1}^{n-1} i z_i \\
&= \underbrace{\sum_{i=1}^{p-1} i(z_i - x_i)}_{=0} + \underbrace{p \cdot x_i}_{=p \cdot s} + \sum_{i=p+1}^{n} i x_i - \underbrace{\sum_{i=p}^{n-1} i z_i}_{z_i = x_{i+1}} \\
&= p \cdot s + \sum_{i=p+1}^{n} i x_i - \sum_{i=p+1}^{n} (i-1) z_{i-1} = p \cdot s + \sum_{i=p+1}^{n} i x_i - \sum_{i=p+1}^{n} (i-1) x_i \\
&= p \cdot s + \sum_{i=p+1}^{n} x_i \\
&= p \cdot s + R_1
\end{aligned}$$

Together with the observation that $0 \leq d < n+1$, this concludes the proof of the decoder. □

Further, we can say the following about the cardinality of the VT code.

Lemma 3.2. *There exists an integer $a \in [0, n]$ such that $\mathcal{VT}_a(n)$ has at least the following cardinality.*

$$|\mathcal{VT}_a(n)| \geq \frac{2^n}{n+1}$$

Hence, its redundancy is at most $\log_2(n+1)$.

Proof. Observe that all cosets $\mathcal{VT}_a(n)$ form a partition of the whole space Σ_2^n, i.e.,

$$\bigcup_{a \in [0,n]} \mathcal{VT}_a(n) = \Sigma_2^n \,.$$

Moreover, they are pairwise disjoint; hence, for any $a, b \in [0, n]$ and $a \neq b$, we have

$$\mathcal{VT}_a(n) \cap \mathcal{VT}_b(n) = \emptyset \,.$$

It follows by the pigeon-hole principle that there must exist at least one coset a (of the $(n+1)$ possible) with cardinality larger than the average. This concludes the statement. □

On a side note, it can be shown [Slo02] that for $a \in [0, n] \setminus \{0, 1\}$, we have

$$|\mathcal{VT}_0(n)| \geq |\mathcal{VT}_a(n)| \geq |\mathcal{VT}_1(n)| \,.$$

The VT code cannot be directly extended to non-binary alphabets. However, Tenengolts was able to construct non-binary single-deletion codes using a binary signature vector which is protected by a binary VT code and a parity constraint.

Construction 3.2 ([Ten84])**.** *For a vector $\mathbf{x} \in \Sigma_q^n$, we associate the binary signature vector $\mathbf{s}(\mathbf{x}) \in \Sigma_2^n$, where $s(\mathbf{x})_1 = 1$ and $s(\mathbf{x})_i = 1$ if and only if $s(\mathbf{x})_i \geq s(\mathbf{x})_{i-1}$. We define the non-binary VT-Codes $\mathcal{VT}_{a,b}(n,q) \subseteq \Sigma_q^n$ as*

$$\mathcal{VT}_{a,b}(n,q) = \left\{ \mathbf{x} \in \Sigma_q^n : \sum_{i=1}^{n} (i-1) \cdot s(\mathbf{x})_i \equiv a \mod n, \sum_{i=1}^{n} x_i \equiv b \mod q \right\}.$$

Theorem 3.4. *The non-binary VT code in Construction 3.2 is a single-deletion correcting code with cardinality at least $\frac{q^n}{qn}$.*

We provide a sketch of the theorem.

Proof sketch via decoder (Full proof in [Ten84]). Observe that a single deletion in a codeword results in a single deletion at the same position in its binary signature vector. Since the binary signature is a binary VT codeword, one can find the position of the deletion up to a run in the original signature vector. A run in the signature vector corresponds to consecutive ascending (or descending) symbols in the original non-binary codeword. Due to the parity fixed by the constraint on the non-binary sequence, one can deduce the value and the exact position.

The cardinality follows again by the pigeon-hole principle. □

Note that the constructions do not imply an efficient encoder/decoder and are generally non-systematic, i.e., the information word does not appear in the codeword. Due to Levenshtein's proof, an efficient decoder for the VT codes was obtained. Linear-time encoders and systematic constructions for the binary VT codes have been discussed in [AGF98] and for the non-binary VT code in [Ten84; CKN19; AVF18; NCS23].

For the work in Chapter 5, we use the systematic t-deletion correcting code from [SGB20b]. The idea is to use higher order VT-*sketches* (similar constraints as used in Chapter 4) and Reed-Solomon codes [RS60] on specific indicator vectors to protect the information vector $\mathbf{u}$. Together with the syndrome compression technique from [SGB20c], a systematic construction with final redundancy of at most $4t \log_2(n) + o(\log_2 n)$ redundancy is achieved.

Chapter 4

Combinations of Insertion/Deletion and Substitution Errors

Abstract

Correcting insertions/deletions as well as substitution errors simultaneously plays an important role in DNA-based data storage systems as well as in classical communications. This chapter deals with the fundamental task of constructing codes that can correct a single insertion or deletion along with a single substitution. A non-asymptotic upper bound on the size of non-binary single-deletion s-substitution correcting codes is derived, showing that the non-asymptotic redundancy of such a code of length n has to be at least $(s+1)\log_q n$. An explicit construction of binary single-deletion single-substitution correcting codes with at most $6\log_2 n + 8$ redundancy bits is presented.

This chapter is based on the work [SWWZ+23]. Lorenz Welter contributed to the conception of the project, the derivation of the results, and the writing of the manuscript.

4.1 Introduction

In DNA storage and file/symbol synchronization, not only do insertions and deletions occur, but classical substitution errors also occur. Therefore, codes correcting an arbitrary combination of these types of errors were studied. Recall that an edit error is a deletion, insertion, or substitution error. With one additional bit of redundancy, the VT codes can correct one edit error [Lev66b]. A systematic construction for these codes was presented in [SKF99]. The work of [CKN+21] extended the single edit correcting codes to the DNA alphabet and [GGR+22] recently generalized it for arbitrary alphabets, both providing order optimal constructions with redundancy $\log_2(n) + O(\log_2(\log_2(n)))$ and linear-time encoders as well. Moreover, the work of [SGB20b] presented an order-optimal binary systematic t-edit error correcting code with redundancy $4t\log_2(n) + o(\log_2(n))$ (see also Section 3.4).

Noteworthy are the works of [CJL+18] and [Hae19] in the field of document exchange protocols, where the results can be directly translated to systematic error-correcting codes. In [CJL+18], the author suggests a construction to correct t insdel with redundancy $O(t\log_2(n))$. Moreover, the author of [Hae19] presents a systematic construction correcting t edit errors with $O(t\log_2^2(n/t) + t)$ redundancy.

Our contribution. However, different results can be achieved by fixing the insdels and substitutions to different constants t and s, respectively. Therefore, one can leverage this

prior knowledge when constructing insdel and substitution error-correcting codes. Hence, this chapter studies codes correcting combinations of insdels and substitutions. One of our main results is an explicit construction of a binary single-deletion single-substitution correcting code with redundancy at most $6\log_2(n)+8$ (see Sections 4.4 and 4.5). Moreover, we also derive a non-asymptotic upper bound on the cardinality of q-ary single-deletion s-substitution codes, which shows that roughly at least redundancy $(s+1)\log_q(n)$ is necessary (see Section 4.3).

4.2 Definitions and Preliminaries

For two positive integers, $t \leq n$ and $s \leq n-t$, $\mathbb{DS}_{t,s}(\mathbf{x})$ is the set of all words received from $\mathbf{x} \in \Sigma_q^n$ after exactly t deletions and at most s substitutions. Note that the order in which the errors occur does not matter, and thus we will mostly assume that first the deletions occurred. We formally define the code in the following.

Definition 4.1. *A code $\mathcal{C}_{t,s}^{\mathrm{DS}} \subseteq \Sigma_q^n$ is called a t-deletion s-substitution correcting code if it can correct any combination of at most t deletions and at most s substitutions.*

Recall that this implies that for all $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}_{t,s}^{\mathrm{DS}}$ it holds that $\mathbb{DS}_{t,s}(\mathbf{c}_1) \cap \mathbb{DS}_{t,s}(\mathbf{c}_2) = \emptyset$. Note that given two codewords $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}_{t,s}^{\mathrm{DS}}$, then there exists no word $\mathbf{z} \in \mathbb{DS}_{t',s'}(\mathbf{c}_1) \cap \mathbb{DS}_{t',s'}(\mathbf{c}_2)$ for given $0 \leq t' \leq t$, $0 \leq s' \leq s$ and $\mathbf{z} \in \Sigma_q^{n-t'}$. We define similarly a *t-insdel s-substitution correcting code* to be a code that corrects any combination of at most t insdels and at most s substitutions.

The goal is to study codes correcting insdels and substitutions. Similarly to the equivalence between insdel and deletion correcting codes (see Section 3.2), the following lemma holds.

Lemma 4.1. *A code $\mathcal{C}_{t,s}^{\mathrm{DS}}$ is a t-insdel s-substitution correcting code if and only if it is a t-deletion s-substitution correcting code.*

Proof. We only show the "if" direction while the "only if" part follows by the definition of t insdels. Observe that also in the insdel case it does not differ in which order the deletions, insertions and substitutions happen. Now assume that for two words $\mathbf{x}, \mathbf{y} \in \Sigma_q^n$ that there is a $\mathbf{z} \in \mathbb{DS}_{t,s}(\mathbf{x}) \cap \mathbb{DS}_{t,s}(\mathbf{y})$. Moreover, let $\mathbf{x}' \in \mathbb{DS}_{0,s}(\mathbf{x})$ and $\mathbf{y}' \in \mathbb{DS}_{0,s}(\mathbf{y})$ such that $\mathbf{z} \in \mathbb{DS}_{t,0}(\mathbf{x}') \cap \mathbb{DS}_{t,0}(\mathbf{y}')$. Thus, from the well-known insdel equivalence result by Levenshtein (see Theorem 3.1) there exists a word $\mathbf{w}$ which can be reached from $\mathbf{x}'$ and $\mathbf{y}'$ by t insdels other than $\mathbf{z}$. □

Therefore, the main focus of the chapter is on t-deletion s-substitution correcting codes and specifically for $t = 1$.

4.3 Bounds on the Size of Codes

The method used to compute a non-asymptotic upper bound for the cardinality of any single-deletion s-substitution code $|\mathcal{C}_{1,s}^{\mathrm{DS}}|$ is similar to the one from [KK13] and [FVY15]. For clarity of the results, the principal concepts of this method are briefly reviewed. Recall that

a sphere-packing bound can be obtained by counting the maximum number of disjoint error balls in the receiving space (see Section 3.3). The main idea is to construct a hypergraph $\mathfrak{H}_s(\mathcal{X}, \mathcal{E}_s)$ out of the channel graph with vertices $\mathcal{X} = \Sigma_q^{n-1} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ where $m = q^{n-1}$ and hyperedges $\mathcal{E}_s = \{E_1, \ldots, E_\ell\} = \{\mathbb{DS}_{1,s}(\mathbf{x}) : \mathbf{x} \in \Sigma_q^n\}$ with $\ell = q^n$. We call a matching $\mathcal{M} \subseteq \mathcal{E}_s$ of $\mathfrak{H}_s$ a collection of pairwise disjoint hyperedges. Hence, analogously to packing the maximum number of disjoint balls into the receiving space, we can find the maximum sized matching $\tau(\mathfrak{H})$ in $\mathfrak{H}_s$. Further, we define a transversal as a subset of all vertices $\mathcal{T} \subseteq \mathcal{X}$ of $\mathfrak{H}_s$ that intersects all hyperedges in $\mathfrak{H}_s$. We denote the minimum sized transversal as $\nu(\mathfrak{H}_s)$. Let $\mathbf{I}_s$ be the $m \times \ell$ incidence matrix of $\mathfrak{H}_s$ where $\mathbf{I}_s(i,j) = 1$ if $\mathbf{x}_i \in E_j$. We can represent a matching as a binary vector $\mathbf{z} \in \{0,1\}^\ell$, satisfying $\mathbf{I}_s \cdot \mathbf{w} \leq \mathbf{1}$ and similarly a transversal as a binary vector $\mathbf{w} \in \{0,1\}^m$, satisfying $\mathbf{I}_s^T \cdot \mathbf{w} \geq \mathbf{1}$. Hence, we can deduce two integer linear programming problems to find the maximum-sized matching or the smallest-sized transversal:

$$\tau(\mathfrak{H}_s) = \max\left\{\sum_{i=1}^{\ell} z_i : \mathbf{I} \cdot \mathbf{z} \leq \mathbf{1}, \mathbf{z} \in \{0,1\}^\ell\right\},$$
$$\nu(\mathfrak{H}_s) = \min\left\{\sum_{i=1}^{m} w_i : \mathbf{I}^T \cdot \mathbf{w} \geq \mathbf{1}, \mathbf{w} \in \{0,1\}^m\right\}.$$

A key observation is that these two optimization problems satisfy weak duality, hence, $\tau(\mathfrak{H}_s) \leq \nu(\mathfrak{H}_s)$. Moreover, we can relax the optimization problems such as $\mathbf{z} \in (\mathbb{R}^+)^\ell$ and $\mathbf{w} \in (\mathbb{R}^+)^m$. Hence, the solutions $\tau^*(\mathfrak{H}_s)$ and $\nu^*(\mathfrak{H}_s)$ of the relaxed matching and transversal optimization problem are an upper and lower bound on the maximization and minimization problem, respectively. Further, the relaxed versions of the optimization problems satisfy strong duality. Hence, we conclude that

$$|\mathcal{C}_{1,s}^{\mathrm{DS}}| \triangleq \tau(\mathfrak{H}_s) \leq \tau^*(\mathfrak{H}_s) = \nu^*(\mathfrak{H}_s) \leq \nu(\mathfrak{H}_s)\,.$$

Thus, the objective is to find the elements $w_{\mathbf{y}} \geq 0$ of the vector $\mathbf{w} \in (\mathbb{R}^+)^m$, called *fractional transversal*, which need to fulfill the condition

$$\sum_{\mathbf{y} \in \mathbb{DS}_{1,s}(\mathbf{x})} w_{\mathbf{y}} \geq 1 \qquad \forall\, \mathbf{x} \in \Sigma_q^n\,. \tag{4.1}$$

Consequently, the following expression is an upper bound for $|\mathcal{C}_{1,s}^{\mathrm{DS}}|$:

$$|\mathcal{C}_{1,s}^{\mathrm{DS}}| \leq \nu^*(\mathfrak{H}_s) = \sum_{\mathbf{y} \in \Sigma_q^{n-1}} w_{\mathbf{y}}\,, \tag{4.2}$$

for any valid fractional transversals $w_{\mathbf{y}}$ fulfilling Equation (4.1).

4.3.1 An Upper Bound on the Cardinality of Single-Deletion Single-Substitution Correcting Codes

Before determining a valid fractional transversal, an essential property for any $\mathbf{y} \in \mathbb{DS}_{1,s}(\mathbf{x})$ is studied in the following.

Claim 4.1. *For all* $\mathbf{x} \in \Sigma_q^n$ *and* $\mathbf{y} \in \mathbb{DS}_{1,s}(\mathbf{x})$*, it holds that*

$$r(\mathbf{y}) \leq r(\mathbf{x}) + 2s\,.$$

Proof. First, observe that for any $\mathbf{y}' \in \mathbb{DS}_{1,0}(\mathbf{x})$, deleting a symbol in $\mathbf{x}$ does not add a run in $\mathbf{y}'$. However, a substitution can increase the number of runs when substituting in $\mathbf{y}'$ a symbol anywhere in the middle of a run with a length greater than three. Thus, resulting in the property that $r(\mathbf{y}) \leq r(\mathbf{y}') + 2s \leq r(\mathbf{x}) + 2s$. □

Thus, the monotonicity argument $|\mathbb{DS}_{1,s}(\mathbf{y})| \leq |\mathbb{DS}_{1,s}(\mathbf{x})|$ as in the single deletion case of [KK13] does not necessarily hold and choosing $w_{\mathbf{y}} = \frac{1}{|\mathbb{DS}_{1,s}(\mathbf{y})|}$ does not suffice as a feasible solution.

The following lemma extends the result from [SY19] and provides the size of $\mathbb{DS}_{1,1}(\mathbf{x})$ for the q-ary alphabet.

Lemma 4.2. *For any word* $\mathbf{x} \in \Sigma_q^n$*,*

$$|\mathbb{DS}_{1,1}(\mathbf{x})| = \begin{cases} (n-1)(q-1)+1 & r(\mathbf{x}) = 1\,, \\ r(\mathbf{x})\left[(n-3)(q-1)+(q-2)\right]+(q+2) & \\ & r(\mathbf{x}) \geq 2\,. \end{cases}$$

Proof. Note that this is only an extension from the binary case, which already has been shown in [SY19, Theorem 2] to be $|\mathbb{DS}_{1,1}(\mathbf{x})| = (n-3)r(\mathbf{x}) + 4$ for $\mathbf{x} \in \Sigma_2^n$. The derived formula here leads to the same expression for $q = 2$. Moreover, the same proof strategy is applied and differs only slightly. Therefore, we will recap the proof and highlight the differences.

The proof assumes $r(\mathbf{x}) \geq 3$, but the theorem can be verified for $r(\mathbf{x}) = 1, 2$. Let $\mathbf{x}^{(i)} \in \mathbb{DS}_{1,0}(\mathbf{x})$ be such that the deleted symbol was in the i^{th} run of $\mathbf{x}$. Since $\mathbb{DS}_{1,0}(\mathbf{x}) = r(\mathbf{x})$, it follows that

$$\mathbb{DS}_{1,1}(\mathbf{x}) = \bigcup_{\mathbf{x}' \in \mathbb{DS}_{1,0}(\mathbf{x})} \mathbb{DS}_{0,1}(\mathbf{x}') = \bigcup_{1 \leq i \leq r(\mathbf{x})} \mathbb{DS}_{0,1}(\mathbf{x}^{(i)})\,.$$

Observe that for all $1 \leq i_1 < i_2 \leq r(\mathbf{x})$, it holds that $\mathrm{d_H}(\mathbf{x}^{(i_2)}, \mathbf{x}^{(i_1)}) = i_2 - i_1$. We will derive the ball size via the inclusion-exclusion principle. Hence, we have the following case distinctions on the intersections of the single-substitution balls.

1. For $1 \leq i \leq r(\mathbf{x})$:
   $|\mathbb{DS}_{0,1}(\mathbf{x}^{(i)})| = (n-1)(q-1)+1$ *(binary case: n).*
2. For $2 \leq i \leq r(\mathbf{x})$:
   $\mathrm{d_H}(\mathbf{x}^{(i-1)}, \mathbf{x}^{(i)}) = 1$, thus $|\mathbb{DS}_{0,1}(\mathbf{x}^{(i-1)}) \cap \mathbb{DS}_{0,1}(\mathbf{x}^{(i)})| = q$ *(binary case: 2).*

3. For $3 \leq i \leq r(\mathbf{x})$:
   $\mathrm{d_H}(\mathbf{x}^{(i-2)}, \mathbf{x}^{(i)}) = 2$, thus $|\mathbb{DS}_{0,1}(\mathbf{x}^{(i-2)}) \cap \mathbb{DS}_{0,1}(\mathbf{x}^{(i)})| = 2\,.$

4. For $3 \leq i \leq r(\mathbf{x})$:
   $|\mathbb{DS}_{0,1}(\mathbf{x}^{(i-2)}) \cap \mathbb{DS}_{0,1}(\mathbf{x}^{(i-1)}) \cap \mathbb{DS}_{0,1}(\mathbf{x}^{(i)})| = 1\,.$

5. For $3 \leq j$ and $j+1 \leq i \leq r(\mathbf{x})$:
   $\mathrm{d_H}(\mathbf{x}^{(i-j)}, \mathbf{x}^{(i)}) \geq 3$, thus $|\mathbb{DS}_{0,1}(\mathbf{x}^{(i-j)}) \cap \mathbb{DS}_{0,1}(\mathbf{x}^{(i)})| = 0\,.$

As stated in brackets, the expansion to q-ary alphabet changed only the quantity in Term 1 from n, and in Term 2 from 2, in the binary case to the respective numbers. Hence, the ball size can be concluded as follows.

$$\begin{aligned}|\mathbb{DS}_{1,1}(\mathbf{x})| &= \underbrace{r(\mathbf{x}) \cdot [(n-1)(q-1)+1]}_{\text{Term 1}} - \underbrace{(r(\mathbf{x})-1) \cdot q}_{\text{Term 2}} - \underbrace{(r(\mathbf{x})-2) \cdot 2}_{\text{Term 3}} + \underbrace{(r(\mathbf{x})-2) \cdot 1}_{\text{Term 4}} \\ &= r(\mathbf{x})nq - 2qr(\mathbf{x}) - r(\mathbf{x})n + r(\mathbf{x}) + q + 2 \\ &= r(\mathbf{x})\,[(n-3)(q-1) + (q-2)] + (q+2)\,.\end{aligned}$$

□

Now, from the results of Lemma 4.2 and Claim 4.1, a valid expression of a fractional transversal $w_{\mathbf{y}}$ can be derived.

Lemma 4.3. *The following choice of $w_{\mathbf{y}}$ for $\mathbf{y} \in \Sigma_q^{n-1}, n \geq 3$ is a fractional transversal for the hypergraph $\mathfrak{H}_1$:*

$$w_{\mathbf{y}}^1 = \begin{cases} \frac{1}{(n-1)(q-1)+1} & r(\mathbf{y}) \leq 3\,, \\ \frac{1}{(r(\mathbf{y})-2)[(n-3)(q-1)+(q-2)]+(q+2)} & r(\mathbf{y}) > 3\,. \end{cases}$$

Proof. For $r(\mathbf{y}) \leq 3$, it is directly verifiable that the condition in Equation (4.1) holds. In the other case, we derive the following by inserting $w_{\mathbf{y}}^1$ in Equation (4.1).

$$\begin{aligned}\sum_{\mathbf{y} \in \mathbb{DS}_{1,1}(\mathbf{x})} w_{\mathbf{y}}^1 &= \sum_{\mathbf{y} \in \mathbb{DS}_{1,1}(\mathbf{x})} \frac{1}{(r(\mathbf{y})-2)\,[(n-3)(q-1)+(q-2)] + (q+2)} \\ &\overset{\text{(a)}}{\geq} \sum_{\mathbf{y} \in \mathbb{DS}_{1,1}(\mathbf{x})} \frac{1}{(r(\mathbf{x})+2-2)\,[(n-3)(q-1)+(q-2)] + (q+2)} \\ &= \sum_{\mathbf{y} \in \mathbb{DS}_{1,1}(\mathbf{x})} \frac{1}{|\mathbb{DS}_{1,1}(\mathbf{x})|} = 1\,,\end{aligned}$$

where we have inserted the observation from Claim 4.1 in Step (a). □

The following claim will be used to compute the upper bound of the cardinality of the code. This claim extends similar statements in [GYD15, Lemma 13, 14].

Claim 4.2. *For $q \geq 2$ and $n \geq 5$:*

$$\sum_{k=1}^{n} \binom{n}{k} (q-1)^k \frac{1}{k} \leq \frac{q^{n+1}}{(q-1)(n-2)} .$$

Proof. First, we reformulate the left term of the claim, similar to [GYD15, Lemma 13].

$$\begin{aligned}
\sum_{k=1}^{n} \binom{n}{k} (q-1)^k \frac{1}{k} &\overset{\text{(b)}}{=} \sum_{k=1}^{n} \frac{(q-1)^k}{k} \sum_{j=k}^{n} \binom{j-1}{k-1} \\
&= \sum_{k=1}^{n} (q-1)^k \sum_{j=k}^{n} \frac{1}{j} \binom{j}{k} \\
&= \sum_{j=1}^{n} \frac{1}{j} \sum_{k=1}^{j} \binom{j}{k} (q-1)^k \\
&= \sum_{j=1}^{n} \frac{1}{j} \left(q^j - 1 \right) ,
\end{aligned}$$

where we used the hockey-stick identity in Step (b). Hence, we can prove the claim by showing that the following inequality holds.

$$\sum_{j=1}^{n} \frac{1}{j} (q^j - 1) \leq \frac{q^{n+1}}{(q-1)(n-2)} . \tag{4.3}$$

The expression will be proven via induction under the conditions stated in the claim.

Base case: For $q = 2$ and $n = 5$ the left-hand side of the inequality evaluates to 14.78 and the right-hand side to 21.3. Now, we fix $n = 5$ and show that it holds for any other $q \geq 2$. Define the function

$$\begin{aligned}
h(q) &\triangleq \sum_{j=1}^{5} \frac{q^j - 1}{j} - \frac{q^6}{3(q-1)} \\
&= \frac{1}{60} \left(-6q^6 + q^5 + 5q^4 + 10q^3 + 30q^2 - 197q + 137 \right) .
\end{aligned}$$

The function $h(q)$ has two roots at approximately -2.196 and 0.835 and its only maximum at $q \approx -1.49$. Since $\left(\frac{dh(q)}{dq} \right)_{q=0.835} \leq 0$, it is verified that $h(q) \leq 0$ for values $q \geq 2$.

Induction hypothesis: The expression in Equation (4.3) holds for n and q.

Induction step: We assume now that the above expression holds for all values $n \geq 5$ and

$q \geq 2$, and show that the same holds for $n+1$.

$$\begin{aligned}
\sum_{j=1}^{n+1} \frac{1}{j}(q^j - 1) &\leq \frac{q^{n+2}}{(q-1)(n-1)}\,, \\
\sum_{j=1}^{n} \frac{1}{j}(q^j - 1) + \frac{q^{n+1}-1}{n+1} &\overset{\text{(c)}}{\leq} \frac{q^{n+2}}{(q-1)(n-1)}\,, \\
\frac{q^{n+1}}{(q-1)(n-2)} + \frac{q^{n+1}-1}{n+1} &\overset{\text{(d)}}{\leq} \frac{q^{n+2}}{(q-1)(n-1)}\,.
\end{aligned}$$

In Step (c), we have split the sum to the terms from $j \in [n]$ and $j = n+1$. Furthermore, in Step (d), we inserted the induction assumption. After dividing the expression by q^{n+1} it is enough to show that

$$\begin{aligned}
\frac{1}{(q-1)(n-2)} + \frac{1}{n+1} &\leq \frac{q}{(q-1)(n-1)}\,, \\
(n+1)(n-1) + (q-1)(n-2)(n-1) &\leq q(n-2)(n+1)\,, \\
-2nq + 3n + 4q - 3 &\leq 0\,,
\end{aligned}$$

which is always true for $n \geq 5$ and $q \geq 2$. □

The below upper bound on $|\mathcal{C}_{1,1}^{\mathrm{DS}}|$ is presented by combining everything.

Theorem 4.1. *For $q \leq n, n \geq 6$ the following is an upper bound on $|\mathcal{C}_{1,1}^{\mathrm{DS}}|$:*

$$|\mathcal{C}_{1,1}^{\mathrm{DS}}| \leq \frac{3 \cdot q^{n-1}}{(n-5)(n-3)(q-1)} + 4q + 2\,.$$

Proof. The number of words in Σ_q^{n-1} with r runs is given by $q(q-1)^{r-1}\binom{n-2}{r-1}$ [KK13], thus we have $\sum_{r=1}^{n} q(q-1)^{r-1}\binom{n-2}{r-1} = q^{n-1}$. We compute the sum over all words in Σ_q^{n-1} of the corresponding elements of the fractional transversal $w_{\mathbf{y}}^1$ from Lemma 4.3. For $r = 1, 2, 3$ define the function

$$\begin{aligned}
g(n,q) &\triangleq \sum_{r=1}^{3} q(q-1)^{r-1}\binom{n-2}{r-1}\frac{1}{(n-1)(q-1)+1} \\
&\leq \frac{q}{(n-1)(q-1)}\left(1 + (q-1) + (q-1)^2\right)\,.
\end{aligned}$$

The rest is given by

$$\begin{aligned}
&\sum_{r=4}^{n-1} q(q-1)^{r-1}\binom{n-2}{r-1} \cdot \frac{1}{(r-2)\left[(n-3)(q-1)+(q-2)\right]+(q+2)} \\
&\leq \frac{q}{(n-3)(q-1)} \sum_{r=4}^{n-1} (q-1)^{r-1}\binom{n-2}{r-1}\frac{1}{r-2}\,.
\end{aligned}$$

For simplicity, we first analyze the summation in the above term without the left multiplicative term:

$$\begin{aligned} f(n,q) &\triangleq \sum_{r=4}^{n-1}(q-1)^{r-1}\binom{n-2}{r-1}\frac{1}{r-2} \\ &= \sum_{r=2}^{n-1}(q-1)^{r+1}\binom{n-2}{r+1}\frac{1}{r} \\ &\overset{\text{(e)}}{=} \sum_{r=2}^{n-3}(q-1)^{r+1}\frac{(n-2)!}{(n-r-3)!\,r!}\left(\frac{1}{r}-\frac{1}{r+1}\right) \\ &= \sum_{r=2}^{n-3}(q-1)^{r+1}\frac{(n-2)!}{(n-r-3)!\,r!}\frac{1}{r} - \sum_{r=2}^{n-3}(q-1)^{r+1}\binom{n-2}{r+1} \\ &= (n-2)(q-1)\sum_{r=2}^{n-3}(q-1)^{r}\binom{n-3}{r}\frac{1}{r} - \sum_{r=3}^{n-2}(q-1)^{r}\binom{n-2}{r}. \end{aligned}$$

We used in Step (e) the fact that $\frac{1}{r(r+1)} = \frac{1}{r} - \frac{1}{r+1}$. Subsequently, in the last expression, the right part of the difference can be calculated as follows

$$\sum_{r=3}^{n-2}(q-1)^{r}\binom{n-2}{r} = \left[q^{n-2} - \frac{(q-1)^2(n-2)(n-3)}{2} - (q-1)(n-2) - 1\right],$$

where we applied the binomial formula $\sum_{k=0}^{n}\binom{n}{k}x^{n-k}y^k = (x+y)^n$. For the left part, the following inequality can be derived by the observation of Claim 4.2.

$$\begin{aligned} &(n-2)(q-1)\sum_{r=2}^{n-3}(q-1)^{r}\binom{n-3}{r}\frac{1}{r} \\ &\qquad \le (n-2)(q-1)\left(\frac{q^{n-2}}{(q-1)(n-5)} - (n-3)(q-1)\right). \end{aligned}$$

Thus, an upper bound for $f(n,q)$ is computed by

$$f(n,q) \le \left(\frac{(n-2)}{(n-5)} - 1\right)q^{n-2} - \frac{1}{2}(n-2)(n-3)(q-1)^2 + (q-1)(n-2) + 1\,.$$

Next, the bounds on the precomputed quantities $f(n,q)$ and $g(n,q)$ are combined in the following manner

$$|\mathcal{C}_{1,1}^{\mathrm{DS}}| \le \frac{q\cdot f(n,q)}{(n-3)(q-1)} + g(n,q)\,.$$

For a better understanding, we split the computation of the expression into two parts, where

one only includes terms with q^n terms and the other one the rest. First, we derive

$$\frac{q}{(q-1)(n-3)}\left(\frac{n-2}{n-5}-1\right)q^{n-2}=\frac{3\cdot q^{n-1}}{(n-3)(n-5)(q-1)}\,.$$

In a consecutive step, the remaining terms of the expression are computed.

$$\begin{aligned}
&\frac{q}{(n-3)(q-1)}\left(-\frac{(n-2)(n-3)(q-1)^2}{2}+(q-1)(n-2)+1\right)\\
&\qquad+\frac{q}{(n-1)(q-1)}\left(1+(q-1)+(q-1)^2\right)\\
&=-\frac{q(n-2)(n-3)(q-1)^2-2}{(n-3)(q-1)2}+\frac{q(q-1)(n-2)}{(n-3)(q-1)}\\
&\qquad+\frac{q}{(n-1)(q-1)}+\frac{q(q-1)}{(n-1)(q-1)}+\frac{q(q-1)^2}{(n-1)(q-1)}\\
&\overset{\text{(f)}}{\leq}\frac{q(n-2)}{n-3}+\frac{q}{(n-1)(q-1)}+\frac{q^2}{n-1}\\
&\overset{\text{(g)}}{\leq}2q+2+2q=4q+2\,,
\end{aligned}$$

where in Step (f) we drop the negative summand, which holds for $n\geq 6$ and $q\geq 2$. Further, we use in Step (g) the inequality $n\geq q$. Finally, the bound in the theorem can be concluded from the last two results by summation. □

We state in the following corollary a looser but more readable bound of that given in Theorem 4.1.

Corollary 4.1. *For $n\geq 12$ and $n\geq q\geq 2$, it holds that*

$$|\mathcal{C}_{1,1}^{\mathrm{DS}}|\leq\frac{4\cdot q^n}{n^2}\,.$$

Consequently, we have that $r_{1,1}^{\mathrm{DS}}\geq 2\log_q(n)-2$.

Proof. Given the constraint in the corollary, it will be shown that the result of Theorem 4.1 is bounded by the stated expression, i.e.,

$$\frac{3\cdot q^{n-1}}{(n-5)(n-3)(q-1)}+4q+2\leq\frac{4\cdot q^n}{n^2}\,.$$

Since $n\geq 12$ and $q\geq 2$, the following is equivalent to show:

$$\frac{3\cdot q^{n-1}}{(n-5)(n-3)(q-1)}+4q+2\leq\frac{4q^{n-1}}{(n-5)(n-3)(q-1)}\,.$$

Since $n\geq 12$ and $q\geq 2$ multiply by the shared denominator and divide everything by q^{n-1},

the subsequent inequality is equivalent.

$$\frac{(4q+2)(n-5)(n-3)(q-1)}{q^{n-1}} \leq 1\,.$$

By bounding $4q+2 \leq 5q$ and taking the $\log_q(\cdot)$ on both sides, we reformulate as follows.

$$-n+3+\log_q(5)+2\log_q(n) \leq 0\,.$$

Using the fact that $\log_2(x) \geq \log_q(x)$ for $x \geq 1$ and $q \geq 2$, the inequality in the corollary holds for $n \geq 12$.

The lower bound on the redundancy follows by

$$r_{1,1}^{\mathrm{DS}} \geq n - \log_q\left(|\mathcal{C}_{1,1}^{\mathrm{DS}}|\right) = n - \frac{4\cdot q^n}{n^2} = n - \Big(2+n-2\log_q(n)\Big) = 2\log_q(n) - 2$$

□

4.3.2 An Upper Bound on the Cardinality of Single-Deletion s-Substitution Correcting Codes

To state a legitimate fractional transversal for the case of s substitutions, first, a lower bound on the cardinality of the ball size $|\mathbb{DS}_{1,s}(\mathbf{x})|$ is derived.

Claim 4.3. *For all $\mathbf{x} \in \Sigma_q^n$, it holds that*

$$|\mathbb{DS}_{1,s}(\mathbf{x})| \geq r(\mathbf{x})\tbinom{n-2s-1}{s}(q-1)^s.$$

Proof. Note that there is an explicit expression for $|\mathbb{DS}_{1,s}(\mathbf{x})|$ in [ASY21, Theorem 6] in the binary case, derived via the inclusion-exclusion principle similarly as in Lemma 4.2. We focus only on the lower bound on the ball size stated in the claim in the q-ary case. From [Rot06], for the substiution ball we have that

$$|\mathbb{DS}_{0,s}(\mathbf{x})| = \sum_{s'=0}^{s} \tbinom{n}{s'}(q-1)^{s'}$$

for any $\mathbf{x} \in \Sigma_q^n$. Additionally, we remember for the single deletion ball it holds that $|\mathbb{DS}_{1,0}(\mathbf{x})| = r(\mathbf{x})$. We recall that $\mathbf{x}^{(i)} \in \mathbb{DS}_{1,0}(\mathbf{x})$ for $\mathbf{x} \in \Sigma_q^n$ denotes the word obtained by a deletion in the i^{th} run with $1 \leq i \leq r(\mathbf{x})$. A lower bound on $|\mathbb{DS}_{1,s}(\mathbf{x})|$ will be calculated using the same concept of Lemma 4.2, i.e., we have that

$$\mathbb{DS}_{1,s}(\mathbf{x}) = \bigcup_{1\leq i\leq r(\mathbf{x})} \mathbb{DS}_{0,s}(\mathbf{x}^{(i)})\,.$$

We remind that $d_{\mathrm{H}}(\mathbf{x}^{(i_1)}, \mathbf{x}^{(i_2)}) = i_2 - i_1$, where $i_1 \leq i_2$. Therefore, it follows that $\mathbb{DS}_{0,s}(\mathbf{x}^{i_1}) \cap \mathbb{DS}_{0,s}(\mathbf{x}^{i_2}) \neq \emptyset$ only if $i_2 - i_1 < 2s+1$, which directly translates to $d_{\mathrm{H}}(\mathbf{x}^{(i_1)}, \mathbf{x}^{(i_2)}) \geq 2s+1$. Thus, if we discard $2s$ positions in $\mathbf{x}^{(i)}$ and neglect all sum terms $0 \leq s' < s$ in $|\mathbb{DS}_{0,s}(\mathbf{x}^{(i)})|$, we can derive the lower bound $|\mathbb{DS}_{1,s}(\mathbf{x})| \geq r(\mathbf{x})\tbinom{n-2s-1}{s}(q-1)^s$. □

Consequently, a fractional transversal for the single-deletion s-substitution case of the hypergraph $\mathfrak{H}_s$ can be formulated.

Lemma 4.4. *The following choice of $w_{\mathbf{y}}$ with $\mathbf{y} \in \Sigma_q^{n-1}$ and $n \geq 2s+1 \geq 3$ is a fractional transversal for the hypergraph $\mathfrak{H}^s$*

$$w_{\mathbf{y}}^s = \begin{cases} \frac{1}{\binom{n-2s-1}{s}(q-1)^s} & r(\mathbf{y}) \leq 2s+1\,, \\ \frac{1}{(r(\mathbf{y})-2s)\binom{n-2s-1}{s}(q-1)^s} & r(\mathbf{y}) > 2s+1\,. \end{cases}$$

Proof outline. The same proof strategy as in Lemma 4.3 can be applied to prove that Equation (4.1) for $w_{\mathbf{y}}^s$ is fulfilled. For the proof, it is necessary to recall that we use for the construction of $w_{\mathbf{y}}^s$ a lower bound on the ball size stated in Claim 4.3 and the run change observation of Claim 4.1. □

As a result of Lemma 4.4, an upper bound for the cardinality of a single-deletion s-substitution correcting code can be stated.

Theorem 4.2. *For $n \geq 4s+1$, the following is an upper bound on $|\mathcal{C}_{1,s}^{\mathrm{DS}}|$.*

$$|\mathcal{C}_{1,s}^{\mathrm{DS}}| \leq \frac{s!(2s+1)}{(n-3s)^s(q-1)^{s+1}(n-1)} \cdot \left[q^n + \frac{q^{2s+2}(n-1)^{2s-1}}{(2s)!} \right].$$

Proof. We calculate the term $\sum_{r=1}^{n-1} q(q-1)^{r-1}\binom{n-2}{r-1} w_{\mathbf{y}}^s$ for all words $\mathbf{y} \in \Sigma_q^{n-1}$ with $w_{\mathbf{y}}^s$ stated as in Lemma 4.4. First, consider only the words with $r(\mathbf{y}) \leq 2s+1$.

$$\begin{aligned} \sum_{r=1}^{2s+1} q(q-1)^{r-1}\binom{n-2}{r-1}\frac{1}{\binom{n-2s-1}{s}(q-1)^s} &= \frac{q}{\binom{n-2s-1}{s}(q-1)^s}\sum_{r=0}^{2s}(q-1)^r\binom{n-2}{r} \\ &\overset{\text{(h)}}{\leq} \frac{q}{\binom{n-2s-1}{s}(q-1)^s}(2s+1)\binom{n-2}{2s}(q-1)^{2s} \\ &\overset{\text{(i)}}{\leq} (2s+1)q^{s+1}\frac{s!}{(n-3s)^s}\frac{(n-2)^{2s}}{(2s)!}\,. \end{aligned}$$

In the computations we used in Step (h) the trivial bound on the sum of the binomial coefficient $\sum_{i=0}^{k}\binom{n}{i}x^i \leq (k+1)\binom{n}{k}x^k$ for $0 \leq k \leq \frac{n}{2}$. Subsequently, we applied the two inequalities $\binom{n}{k} \geq \frac{(n-k+1)^k}{k!}$ and $\binom{n}{k} \leq \frac{(n)^k}{k!}$ in Step (i).

We now derive the sum over all words with $r(\mathbf{y}) \geq 2s+2$.

$$\begin{aligned}
&\sum_{r=2s+2}^{n-1} q(q-1)^{r-1}\binom{n-2}{r-1}\frac{1}{\binom{n-2s-1}{s}}\frac{1}{(q-1)^s}\frac{1}{r-2s} \\
&\overset{\text{(j)}}{\leq} \frac{q}{(q-1)^s}\frac{1}{\binom{n-2s-1}{s}}\sum_{r=2s+2}^{n-1}(q-1)^{r-1}\binom{n-2}{r-1}\frac{2s+1}{r} \\
&\leq \frac{q}{(q-1)^{s+1}}\frac{2s+1}{\binom{n-2s-1}{s}}\frac{1}{n-1}\sum_{r=2s+2}^{n-1}(q-1)^{r}\binom{n-1}{r} \\
&\leq \frac{q}{(q-1)^{s+1}}\frac{2s+1}{\binom{n-2s-1}{s}}\frac{1}{n-1}q^{n-1} \\
&\overset{\text{(k)}}{\leq} \frac{(2s+1)s!}{(q-1)^{s+1}}\frac{q^n}{(n-3s)^s(n-1)}.
\end{aligned}$$

In Step (j) we applied the inequality $\frac{1}{r-2s} \leq \frac{2s+1}{r}$ and in Step (k) we used again the fact that $\binom{n}{k} \geq \frac{(n-k+1)^k}{k!}$.

We conclude the statement of the theorem by adding the sum of all words with $r \leq 2s+1$ to the last derived equation. □

Corollary 4.2. *For $n \geq 28$, $2s+2 \leq \log_2(n)$ and $q \geq 2$, it holds that*

$$|\mathcal{C}_{1,s}^{\mathrm{DS}}| \leq \frac{2 \cdot s! \cdot (2s+1) \cdot q^n}{(n-3s)^s(q-1)^{s+1}(n-1)}\,.$$

Consequently, we have that $r_{1,s}^{\mathrm{DS}} \geq (s+1)\log_q(n) + (s+1)\log_q(q-1) - \log_q(c_s)$, where c_s is a constant only depending on s.

Proof. By comparing the statement and the result of Theorem 4.2, it is enough to show that

$$q^n + \frac{q^{2s+2}(n-1)^{2s-1}}{(2s)!} \leq 2q^n.$$

Dropping the denominator $(2s!)$, getting the left summand to the right of the inequality, and taking the $\log_q(\cdot)$, the following equivalent form is derived.

$$(2s+2) + (2s-1)\log_q(n-1) \leq n.$$

Using the fact that $\log_2(x) \geq \log_q(x)$ for $x \geq 1$ and $q \geq 2$ and the constraint that $(2s+2) \leq \log_2(n)$, the following is derived.

$$\log_2(n) + \log_2^2(n) \leq n,$$

which holds for $n \geq 28$. For the redundancy, we have

$$\begin{aligned} r_{1,s}^{\mathrm{DS}} &= n - \log_q\left(|\mathcal{C}_{1,s}^{\mathrm{DS}}|\right) \\ &\geq n - \Big(\log_q\left(2s!(2s+1)\right) + n - s\log_q(n-3s) - (s+1)\log_q(q-1) - \log_q(n-1)\Big) \\ &\geq s\log_q(n-3s) + \log_q(n-1) + (s+1)\log_q(q-1) - \log_q\left(2s!(2s+1)\right) \\ &\overset{\text{(l)}}{\geq} (s+1)\log_q(\frac{n}{s}) + (s+1)\log_q(q-1) - \log_q\left(2s!(2s+1)\right) \\ &\overset{\text{(m)}}{\geq} (s+1)\log_q(n) + (s+1)\log_q(q-1) - \log_q(c_s)\,. \end{aligned}$$

In Step (l), we used the fact that $\frac{n}{s} \leq (n-3s) \leq (n-1)$ for the parameter regime of the corollary and in Step (m) we simply define $c_s = s^{s+1}2s!(2s+1)$. □

Note that unlike in the proof of Theorem 4.1, in the proof of Theorem 4.2, Claim 4.2 is not applied. Instead, Claim 4.3 and a different upper bound is used. For this reason, the bound for the value of $s = 1$ in Corollary 4.2 is not the same as the bound stated Corollary 4.1.

4.4 Properties of Number-Theoretic Codes

In this section, we study a family of number-theoretic codes and their properties in the presence of a single-deletion single-substitution error.

Definition 4.2. *For non-negative integers a and N, a binary code $\mathcal{C} \subseteq \Sigma_2^n$ is called a $(\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n); a, N)$-congruent code if defined as*

$$\mathcal{C}(\boldsymbol{\gamma}; a, N) = \left\{ \mathbf{c} \in \Sigma_2^n \mid \sum_{i=1}^{n} \gamma_i \cdot c_i \equiv a \mod N \right\}.$$

We provide lemmas that derive several basic properties in case the intersection of the single-deletion single-substitution balls of two codewords in $\mathcal{C}(\boldsymbol{\gamma}; a, N)$ is not trivial. For the rest of this section it is assumed that $\mathbf{x}, \mathbf{y} \in \mathcal{C}(\boldsymbol{\gamma}; a, N)$, where $\mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y}) \neq \emptyset$ so that there exists $\mathbf{z} \in \mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y})$. For simplicity of notation, the expression $\mathbf{x}(d, e)$ is defined as the error word achieved from $\mathbf{x}$ by deleting the bit at index d and substituting the bit at index e. The variables $d_x, d_y, e_x, e_y \in [n]$ are indices such that $\mathbf{z} = \mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$. It can be assumed without loss of generalization that $d_x < d_y$. We define $+$ as the addition, $-$ as the summation, and $\cdot$ as the multiplication operator over the real number. To shift the values of the substituted bits from binary to ± 1, the following notation is used: $\delta_i \triangleq 2 \cdot x_i - 1$. To better understand the error model and the definition of the variables, we provide an illustration in Figure 4.1.

More precisely, the following lemmas investigate the relations of the codewords $\mathbf{x}, \mathbf{y} \in \mathcal{C}(\boldsymbol{\gamma}; a, N)$ given that $\mathbf{z} = \mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$ and the resulting relations due to the codeword constraints for different orderings of the variables $d_x, d_y, e_x, e_y \in [n]$. We need to distinguish the three cases:

1. $e_x, e_y \notin [d_x, d_y]$, such that $e_x \neq e_y$ (see Lemma 4.5) .

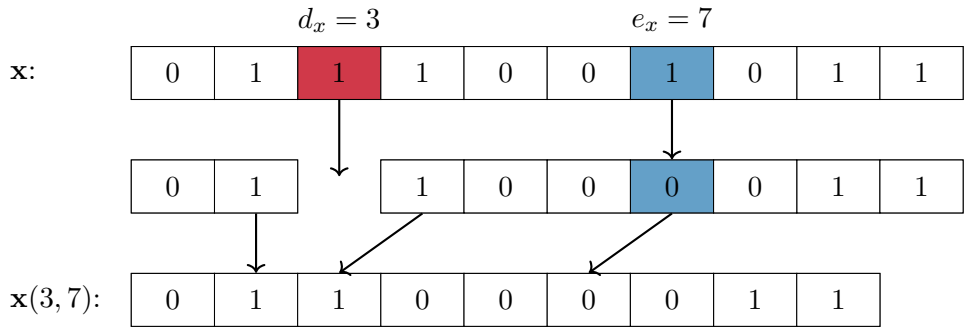


Figure 4.1: An example of $\mathbf{x}$ and the error word $\mathbf{x}(d_x, e_x) = \mathbf{x}(3, 7)$. Further, we denote $\delta_{e_x} = \delta_7 = 2 \cdot x_7 - 1 = 1$.

2. $e_x \in [d_x, d_y]$ and $e_y \notin [d_x, d_y]$ (see Lemma 4.6) .

3. $e_x, e_y \in [d_x, d_y]$, such that $e_x \neq e_y + 1$ (see Lemma 4.7) .

Further, we will provide in Remark 4.1 an explanation of why we do not consider the choices $e_x = e_y$ in Case 1 and $e_x = e_y + 1$ in Case 3. Moreover, we prove in Claim 4.4 that the case of $e_x \notin [d_x, d_y]$ and $e_y \in [d_x, d_y]$ can be omitted due to symmetry properties of a $(\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n); a, N)$-congruent code. We start with the analysis of the first case.

Lemma 4.5. *For the ordering $e_x, e_y \notin [d_x, d_y]$ with $e_x \neq e_y$ the following properties hold:*

(i) *For $i \in [d_x + 1, d_y]$, $x_i = y_{i-1}$.*
(ii) *For $i \in \{e_x, e_y\}$, $x_i = \overline{y_i}$.*
(iii) *For $i \in [n] \setminus ([d_x, d_y] \cup \{e_x, e_y\})$, $x_i = y_i$.*
(iv) $\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) = x_{d_x} \cdot \gamma_{d_x} + \sum_{i=d_x+1}^{d_y} (\gamma_i - \gamma_{i-1}) \cdot x_i - y_{d_y} \cdot \gamma_{d_y}$.
(v) $\delta_{e_x} \cdot \gamma_{e_x} + \delta_{e_y} \cdot \gamma_{e_y} + x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) \equiv 0 \mod N$.

Proof. We can conclude Properties 4.5–(i) and 4.5–(ii) directly from the definition of $\mathbf{z} = \mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$ by distinguishing the following three intervals:

1. For any $i \in [1, d_x - 1] \setminus \{e_x, e_y\}, z_i = x_i = y_i$.
2. For any $i \in [d_x, d_y - 1], z_i = x_{i+1} = y_i$ and $z_{d_y} = x_{d_y+1}$.
3. For any $i \in [d_y + 1, n] \setminus \{e_x, e_y\}, z_i = x_{i+1} = y_{i+1}$.

For the substituted bit at index e_x, it holds that either $e_x < d_x$ or $e_x > d_y$ due to the ordering fixed in the statement of the lemma. In the first case, we have that $x_{e_x} = \overline{z_{e_x}}$ and $y_{e_x} = z_{e_x}$, while in the latter we have $x_{e_x} = \overline{z_{e_x-1}}$ and $y_{e_x} = z_{e_x-1}$. Both cases result to $x_{e_x} = \overline{y_{e_x}}$. The case of the substituted bit at index e_y can be proved similarly. This concludes the proof of Property 4.5–(iii). We illustrate in Figure 4.2 the relations of $\mathbf{x}$ and $\mathbf{y}$ for the specific ordering in the lemma.

In order to prove Property 4.5–(iv), the following summation is calculated as

$$\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) = x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} \gamma_i \cdot x_i - \sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i \, .$$

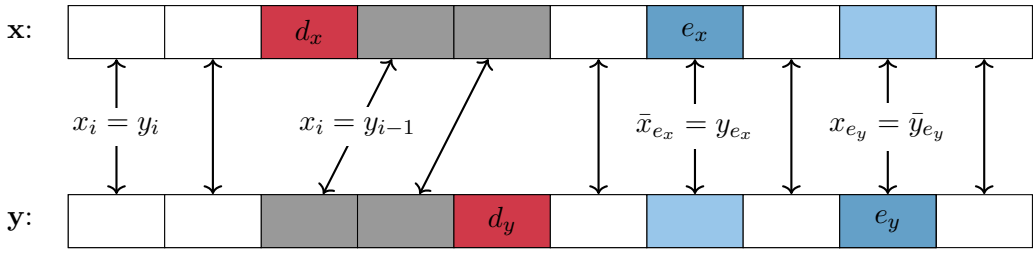


Figure 4.2: Illustration for relations between words $\mathbf{x}$ and $\mathbf{y}$, assuming there exists $\mathbf{z} \in \mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y})$ such that $e_x, e_y \notin [d_x, d_y]$ (see Lemma 4.5).

We reformulate the last summation on the right-hand side using Property 4.5–(i) as follows.

$$\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i = \sum_{i=d_x+1}^{d_y} \gamma_{i-1} \cdot y_{i-1} = \sum_{i=d_x+1}^{d_y} \gamma_{i-1} \cdot x_i \,.$$

Thus, the equality can be rewritten as

$$x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} \gamma_i \cdot x_i - \sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i = x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} (\gamma_i - \gamma_{i-1}) \cdot x_i \,,$$

which concludes the proof of Property 4.5–(iv).

Since $\mathbf{x}, \mathbf{y} \in \mathcal{C}(\boldsymbol{\gamma}; a, N)$, it holds that

$$\sum_{i=1}^{n} \gamma_i x_i \equiv \sum_{i=1}^{n} \gamma_i y_i \equiv a \mod N \,,$$

which can be reformulated to

$$\sum_{i=1}^{n} \gamma_i x_i - \sum_{i=1}^{n} \gamma_i y_i \equiv 0 \mod N \,.$$

Furthermore, Property 4.5–(ii) implies that

$$\sum_{i=1}^{d_x-1} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_y+1}^{n} \gamma_i \cdot (x_i - y_i) = \gamma_{e_x} \cdot (x_{e_x} - y_{e_x}) + \gamma_{e_y} \cdot (x_{e_y} - y_{e_y}).$$

Additionally, Property 4.5–(iii) leads to the fact that for any $e \in \{e_x, e_y\}$ we have that $x_e - y_e = x_e - \overline{x_e} = 2 \cdot x_e - 1 = \delta_{x_e}$. Combined with Property 4.5–(iv), the following

equivalence holds:

$$\begin{aligned}
&\delta_{e_x} \cdot \gamma_{e_x} + \delta_{e_y} \cdot \gamma_{e_y} + x_{d_x} \cdot \gamma_{d_x} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) - y_{d_y} \cdot \gamma_{d_y} \\
&= \sum_{i=1}^{d_x-1} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_y+1}^{n} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) \\
&= \sum_{i=1}^{n} \gamma_i x_i - \sum_{i=1}^{n} \gamma_i y_i \equiv 0 \mod N\,.
\end{aligned}$$

This proves Property 4.5–(v). □

The following lemmas extend Lemma 4.5 for the remaining cases between the indices e_x, e_y and the interval $[d_x, d_y]$.

Lemma 4.6. *For the ordering $e_x \in [d_x, d_y]$ and $e_y \notin [d_x, d_y]$ the following properties hold:*

(i) *For $i \in [d_x + 1, d_y] \setminus \{e_x\}$, it holds $x_i = y_{i-1}$.*
(ii) *For $i = e_y$, $x_i = \overline{y_i}$ and for $i = e_x$, $x_i = \overline{y_{i-1}}$.*
(iii) *For $i \in [n] \setminus ([d_x, d_y] \cup \{e_y\}), x_i = y_i$.*
(iv) $\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) = x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \delta_{e_x} \cdot \gamma_{e_x-1} + \sum_{i=d_x+1}^{d_y} (\gamma_i - \gamma_{i-1}) \cdot x_i$.
(v) $\delta_{e_x} \cdot \gamma_{e_x-1} + \delta_{e_y} \cdot \gamma_{e_y} + x_{d_x} \cdot \gamma_{d_x} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) - y_{d_y} \cdot \gamma_{d_y} \equiv 0 \mod N$.

Proof. Similar to the proof of Lemma 4.5, and by the definition of $\mathbf{z}$, it is possible to conclude the following three relations between the words $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ on the intervals indicated in the lemma:

1. For any $i \in [1, d_x - 1]/\{e_y\}, z_i = x_i = y_i$.
2. For any $i \in [d_x, d_y - 1]/\{e_x\}, z_i = x_{i+1} = y_i$ and $z_{d_y} = x_{d_y+1}$.
3. For any $i \in [d_y + 1, n]/\{e_y\}, z_i = x_{i+1} = y_{i+1}$.

According to the definitions of the ordering, we have the following relations at the substituted bits with indices e_x and e_y. First, for the substituted bit at index $d_x \leq e_x < d_y$, it holds that $x_{e_x} = \overline{z_{e_x-1}}$ and $y_{e_x-1} = z_{e_x-1}$, which is equivalent to $x_{e_x} = \overline{y_{e_x-1}}$. For the other substituted bit at index e_y it holds that either $e_y < d_x$ or $e_y > d_y$. In the first case, we have $x_{e_y} = \overline{z_{e_y}}$ and $y_{e_y} = z_{e_y}$, whereas in the second case we have $x_{e_y} = \overline{z_{e_y-1}}, y_i = z_{e_y-1}$. In both cases, it is equivalent to write $x_{e_y} = \overline{y_{e_y}}$. Consequently, we can conclude the proof of Properties 4.6–(i) to 4.6–(iii). We provide an illustration of the relations between $\mathbf{x}$ and $\mathbf{y}$ for the specific ordering in the lemma in Figure 4.3.

Before proving Property 4.6–(iv), we first reformulate the summation $\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i$ to

$$\begin{aligned}
\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i &= \sum_{i=d_x,\ i\neq e_x-1}^{d_y-1} \gamma_i \cdot y_i + \gamma_{e_x-1} \cdot y_{e_x-1} \\
&\overset{\text{(n)}}{=} \sum_{i=d_x,\ i\neq e_x-1}^{d_y-1} \gamma_i \cdot x_{i+1} + \gamma_{e_x-1} \cdot \overline{x_{e_x}}\,.
\end{aligned}$$

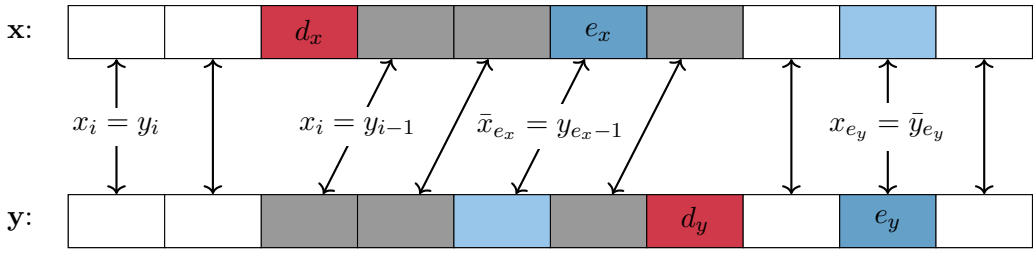


Figure 4.3: Illustration for relations between words $\mathbf{x}$ and $\mathbf{y}$, assuming there exists $\mathbf{z} \in \mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y})$ such that $e_x \in [d_x, d_y]$ and $e_y \notin [d_x, d_y]$ (see Lemma 4.6).

Since we first split the single summation into the two terms which fit the intervals of Properties 4.6–(i) and 4.6–(ii), we applied in Step (n) their already concluded relations. By simple addition and subtraction of $\gamma_{e_x-1} \cdot x_{e_x}$ on the last expression, we can simplify the summation interval, which results in

$$\begin{aligned}\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i &= \sum_{i=d_x}^{d_y-1} \gamma_i \cdot x_{i+1} - \gamma_{e_x-1} \cdot x_{e_x} + \gamma_{e_x-1} \cdot \overline{x_{e_x}} \\ &= \sum_{i=d_x}^{d_y-1} \gamma_i \cdot x_{i+1} + \gamma_{e_x-1} \cdot (\overline{x_{e_x}} - x_{e_x}) \\ &= \sum_{i=d_x}^{d_y-1} \gamma_i \cdot x_{i+1} - \delta_{e_x} \cdot \gamma_{e_x-1}\,,\end{aligned}$$

where we inserted in the last step the definition of $(x_e - \overline{x_e}) = 2 \cdot x_e - 1 = \delta_{x_e}$. To prove Property 4.6–(iv), we first separate the summation on the left-hand side as follows.

$$\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) = x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} \gamma_i \cdot x_i - \sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i.$$

The expression is transformed into the following by substituting the last element with the new form derived above.

$$\begin{aligned}\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) &= x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} \gamma_i \cdot x_i - \left(\sum_{i=d_x}^{d_y-1} \gamma_i \cdot x_{i+1} - \delta_{e_x} \cdot \gamma_{e_x-1} \right) \\ &= x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} (\gamma_i - \gamma_{i-1}) \cdot x_i + \delta_{e_x} \cdot \gamma_{e_x-1}\,.\end{aligned}$$

In the last step, we shifted the indices by one up in the second summation's elements, enabling us to rewrite the two summations using a single sum. This concludes the proof of Property 4.6–(iv).

It remains to show Property 4.6–(v). Analogously to the proof of Property 4.5–(v) in

Lemma 4.5, due to the fact that $\mathbf{x}, \mathbf{y} \in \mathcal{C}(\boldsymbol{\gamma}; a, N)$ we have

$$\sum_{i=1}^{n} \gamma_i x_i - \sum_{i=1}^{n} \gamma_i y_i = \sum_{i=1}^{n} \gamma_i (x_i - y_i) \equiv 0 \mod N .$$

We already analyzed the difference in the interval of summation indices $i \in [d_x, d_y]$ in the proof of Property 4.6–(iv). Hence, we first focus on the other indices $i \in [n] \setminus [d_x, d_y]$ to compute

$$\begin{aligned} \sum_{i=1}^{d_x-1} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_y+1}^{n} \gamma_i \cdot (x_i - y_i) &\overset{\text{(o)}}{=} \gamma_{e_y} \cdot (x_{e_y} - y_{e_y}) \\ &\overset{\text{(p)}}{=} \gamma_{e_y} \cdot (x_{e_y} - \overline{x_{e_y}}) \\ &= \gamma_{e_y} \cdot \delta_{x_{e_y}} . \end{aligned}$$

The equality in Step (o) follows from Property 4.6–(iii) and the one in Step (p) from Property 4.6–(ii). Further, the last equality simply follows by definition of $\delta_{x_{e_y}} = 2 \cdot x_{e_y} - 1 = x_{e_y} - \overline{x_{e_y}}$.

We conclude the proof of Property 4.6–(v) by combining the above derivations with Property 4.6–(iv) which leads to

$$\begin{aligned} \sum_{i=1}^{n} \gamma_i x_i - \sum_{i=1}^{n} \gamma_i y_i &= \sum_{i=1}^{d_x-1} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_y+1}^{n} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) \\ &= \delta_{e_y} \cdot \gamma_{e_y} + \delta_{e_x} \cdot \gamma_{e_x} + x_{d_x} \cdot \gamma_{d_x} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) - y_{d_y} \cdot \gamma_{d_y} \\ &\equiv 0 \mod N . \end{aligned}$$

□

It remains to show the case where both indices of the substituted bits are in the interval spanned by the deleted bits.

Lemma 4.7. *For the ordering $e_x, e_y \in [d_x, d_y]$ with $e_x \neq e_y + 1$ the following properties hold:*

(i) *For $d_x < i \leq d_y$, $x_i = y_{i-1}$.*
(ii) *For $i \in \{e_x, e_y + 1\}$, $x_i = \overline{y_{i-1}}$.*
(iii) *For $i \in [n]/[d_x, d_y], x_i = y_i$.*
(iv) $\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) = \delta_{e_x} \cdot \gamma_{e_x - 1} + \delta_{e_y+1} \cdot \gamma_{e_y} + x_{d_x} \cdot \gamma_{d_x} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) - y_{d_y} \cdot \gamma_{d_y}$.
(v) $\delta_{e_x} \cdot \gamma_{e_x - 1} + \delta_{e_y+1} \cdot \gamma_{e_y} + x_{d_x} \cdot \gamma_{d_x} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) - y_{d_y} \cdot \gamma_{d_y} \equiv 0 \mod N$.

Proof. Similar to the previous proofs, via the definition of $\mathbf{z} = \mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$, we can conclude the following relations which proves Properties 4.7–(i) and 4.7–(iii):

1. For any $i \in [1, d_x - 1]/$, $z_i = x_i = y_i$.
2. For any $i \in [d_x, d_y - 1]/\{e_x, e_y\}$, $z_i = x_{i+1} = y_i$ and $z_{d_y} = x_{d_y+1}$.
3. For any $i \in [d_y + 1, n]$, $z_i = x_{i+1} = y_{i+1}$.

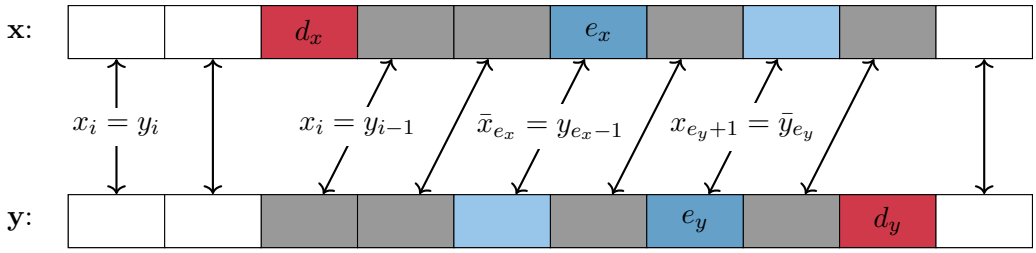


Figure 4.4: Illustration for relations between words $\mathbf{x}$ and $\mathbf{y}$, assuming there exists $\mathbf{z} \in \mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y})$ such that $e_x, e_y \in [d_x, d_y]$ (see Lemma 4.7).

For the indices of the substituted bits e_x and e_y, it is given that $d_x < e_x, e_y < d_y$. Hence, we have at index e_x the relations $z_{e_x-1} = \overline{x_{e_x}}$ and $z_{e_x-1} = y_{e_x-1}$ which equivalently is $x_{e_x} = \overline{y_{e_x-1}}$. The relations at index e_y are $z_{e_y} = \overline{y_{e_y}}$ and $x_{e_y+1} = z_{e_y}$ which is identical to $x_{e_y+1} = \overline{y_{e_y}}$. This concludes Property 4.7–(ii). We illustrate the relations of $\mathbf{x}$ and $\mathbf{y}$ for the specific ordering in the lemma in Figure 4.4.

In order to prove Property 4.7–(iv), we start again by splitting $\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i$ into terms such that we can subsequently apply Properties 4.7–(i) and 4.7–(ii) as follows.

$$\begin{aligned}\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i &= \sum_{i \in [d_x, d_y-1]/\{e_x-1, e_y\}} \gamma_i \cdot y_i + \gamma_{e_x-1} \cdot y_{e_x-1} + \gamma_{e_y} \cdot y_{e_y} \\ &= \sum_{i \in [d_x, d_y-1]/\{e_x-1, e_y\}} \gamma_i \cdot x_{i+1} + \gamma_{e_x-1} \cdot \overline{x_{e_x}} + \gamma_{e_y} \cdot \overline{x_{e_x+1}}\end{aligned}$$

In the next step, we add and subtract the missing elements for the indices $i \in \{e_x - 1, e_y\}$ to the equation and afterward shift the summation indices by one up. Thus, we further compute

$$\begin{aligned}\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i &= \sum_{i=d_x}^{d_y-1} \gamma_i \cdot x_{i+1} - \gamma_{e_x-1} \cdot x_{e_x} - \gamma_{e_y} \cdot x_{e_y+1} + \gamma_{e_x-1} \cdot \overline{x_{e_x}} + \gamma_{e_y} \cdot \overline{x_{e_y+1}} \\ &= \sum_{i=d_x+1}^{d_y} \gamma_{i-1} \cdot x_i + \gamma_{e_x-1} \cdot (\overline{x_{e_x}} - x_{e_x}) + \gamma_{e_y} \cdot (\overline{x_{e_y+1}} - x_{e_y+1}) \\ &= \sum_{i=d_x+1}^{d_y} \gamma_{i-1} \cdot x_i - \delta_{e_x} \cdot \gamma_{e_x-1} - \delta_{e_y+1} \cdot \gamma_{e_y} \,,\end{aligned}$$

where in the last step we rewrote $\delta_e = (x_e - \overline{x_e}) = 2 \cdot x_e - 1$ for $e \in \{e_x, e_y + 1\}$. Since we reformulated $\sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i$ into terms depending on elements of $\mathbf{x}$ we can finalize the proof

of Property 4.7–(iv) as follows.

$$\begin{aligned}
&\sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) \\
&\overset{\text{(q)}}{=} x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} \gamma_i \cdot x_i - \sum_{i=d_x}^{d_y-1} \gamma_i \cdot y_i \\
&\overset{\text{(r)}}{=} x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} \gamma_i \cdot x_i - \left(\sum_{i=d_x+1}^{d_y} \gamma_{i-1} \cdot x_i - \delta_{e_x} \cdot \gamma_{e_x-1} - \delta_{e_y+1} \cdot \gamma_{e_y} \right) \\
&\overset{\text{(s)}}{=} x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) + \delta_{e_x} \cdot \gamma_{e_x-1} + \delta_{e_y+1} \cdot \gamma_{e_y} \,.
\end{aligned}$$

We first separated the sum terms in Step (q) to fit our needs, then substituted the term above in Step (r) and combined the sum terms again in Step (s). Hence, we conclude the proof of Property 4.7–(iv).

In the last step, we prove Property 4.7–(v). First, we observe that Property 4.7–(iii) implies that

$$\sum_{i=1}^{d_x-1} \gamma_i \cdot (x_i - y_i) + \sum_{i=d_y+1}^{n} \gamma_i \cdot (x_i - y_i) = 0 \,.$$

Since $\mathbf{x}, \mathbf{y} \in \mathcal{C}(\boldsymbol{\gamma}; a, N)$ we can derive with the help of the above observation that

$$\begin{aligned}
\sum_{i=1}^{n} \gamma_i x_i - \sum_{i=1}^{n} \gamma_i y_i &= 0 + \sum_{i=d_x}^{d_y} \gamma_i \cdot (x_i - y_i) \\
&\overset{\text{(t)}}{=} x_{d_x} \cdot \gamma_{d_x} - y_{d_y} \cdot \gamma_{d_y} + \sum_{i=d_x+1}^{d_y} x_i \cdot (\gamma_i - \gamma_{i-1}) + \delta_{e_x} \cdot \gamma_{e_x-1} + \delta_{e_y+1} \cdot \gamma_{e_y} \\
&\equiv 0 \mod N \,,
\end{aligned}$$

where in Step (t) we have inserted Property 4.7–(iv). □

In the next step, we show that the substituted bits cancel each other out in particular arrangements of the indices e_x and e_y. In these cases, the analysis of a single-deletion and single-substitution error will be simplified to the analysis of a single-deletion error.

Remark 4.1. *The cases $e_x = e_y$ for the conditions defined in Lemma 4.5 and the case $e_x = e_y + 1$ for the conditions defined in Lemma 4.7 are left out and will remain left out for the remainder of this work. This results from the fact that, in these cases, the result of the substitution is the same. Thus, we can derive the equivalences*

1. $\overline{x_{e_x}} = \overline{y_{e_y}} \iff x_{e_x} = y_{e_y}$ *(for the conditions defined in Lemma 4.5).*
2. $\overline{x_{e_x}} = \overline{y_{e_y+1}} \iff x_{e_x} = y_{e_y+1}$ *(for the conditions defined in Lemma 4.7).*

In both cases, it follows that

$$\mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y}) \neq \emptyset \iff \mathbb{DS}_{1,0}(\mathbf{x}) \cap \mathbb{DS}_{1,0}(\mathbf{y}) \neq \emptyset \,.$$

The properties of codewords that share a single-deletion error word are well-researched in the literature (e.g., see Theorem 3.3).

As mentioned at the beginning of this section, the scenario in which $e_y \in [d_x, d_y]$ and $e_x \notin [d_x, d_y]$ can be disregarded due to some symmetry properties of the $(\boldsymbol{\gamma}; a, N)$-congruent codes. This will be shown in detail by the following claims. We denote the variable ϵ_x as 1 if $e_x \in [d_x, d_y]$ and 0 otherwise. The variable ϵ_y is defined similarly.

Claim 4.4. *For any $\mathbf{x}, \mathbf{y} \in \Sigma_2^n$ and for any $d_x, d_y, e_x, e_y \in [n]$ the substituted bits in two single-deletion single-substitution error words are interchangeable, i.e. $\mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$ if and only if $\mathbf{x}(d_x, e_y + \epsilon_y) = \mathbf{y}(d_y, e_x - \epsilon_x)$.*

Proof. Recall that we have proven Lemmas 4.5 to 4.7 given the assumption that $\mathbf{z} = \mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$. First, we analyze the case that

$$e_x, e_y \notin [d_x, d_y] \iff_{\text{def}} \quad \epsilon_x = \epsilon_y = 0 \,.$$

According to Lemma 4.5, the following equalities follow.

$$\begin{aligned} &x_{e_x} = \overline{y_{e_x}}, x_{e_y} = \overline{y_{e_y}} \,, \\ &x_i = y_{i-1}, \forall i \in [d_x + 1, d_y] \,, \\ &x_i = y_i, \forall i \in [n]/[d_x, d_y] \cup \{e_x, e_y\} \,. \end{aligned}$$

These statements are equivalent in the case of the assumption of

$$\mathbf{x}(d_x, e_y) = \mathbf{y}(d_y, e_x) \,,$$

which completes the first case.

Next, we consider the case that

$$e_x \in [d_x, d_y], e_y \notin [d_x, d_y] \iff_{\text{def}} \quad \epsilon_x = 0, \epsilon_y = 1 \,.$$

From Lemma 4.6, the following equalities hold

$$\begin{aligned} &x_{e_x} = \overline{y_{e_x - 1}}, x_{e_y} = \overline{y_{e_y}} \,, \\ &x_i = y_{i-1}, \forall i \in [d_x + 1, d_y]/\{e_x\} \,, \\ &x_i = y_i, \forall i \in [n]/[d_x, d_y] \cup \{e_y\} \,. \end{aligned}$$

Notice that by the same lemma, these equalities are equivalent precisely to the fact that

$$\mathbf{x}(d_x, e_y) = \mathbf{y}(d_y, e_x - 1) \,.$$

The rest of the cases $\epsilon_x = 0, \epsilon_y = 1$ and $\epsilon_x = 1, \epsilon_y = 1$ can be proven similarly using the properties of Lemmas 4.6 and 4.7, respectively. □

At the end of this section, we show some properties of the complement words $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$, which simplify the proof of the presented code construction in Section 4.5.

Claim 4.5. *For any $\mathbf{x}, \mathbf{y} \in \Sigma_2^n$ having a non-empty intersection between the errors balls is invariant to the complement operation, or formally:*

$$\mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y}) \neq \emptyset \iff \mathbb{DS}_{1,1}(\overline{\mathbf{x}}) \cap \mathbb{DS}_{1,1}(\overline{\mathbf{y}}) \neq \emptyset \,.$$

Proof. We have that

$$\mathbb{DS}_{1,1}(\mathbf{x}) \cap \mathbb{DS}_{1,1}(\mathbf{y}) \neq \emptyset \iff \exists\, d_x, d_y, e_x, e_y \in [n] : \mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$$

For these fixed indices, it directly follows that $\overline{\mathbf{x}}(d_x, e_x) = \overline{\mathbf{y}}(d_y, e_y)$ which in a next step implies $\mathbb{DS}_{1,1}(\overline{\mathbf{x}}) \bigcap \mathbb{DS}_{1,1}(\overline{\mathbf{y}}) \neq \emptyset$. The opposite direction is proved similarly. □

Claim 4.6. *For any $\mathbf{x}, \mathbf{y} \in \Sigma_2^n$ being in the same $(\boldsymbol{\gamma}; a, N)$-congruent code is invariant to the complement operation. Formally, for a weight vector $\boldsymbol{\gamma} \in \mathbb{Z}^n$ and non-negative integers a and N, it holds that*

$$\mathbf{x}, \mathbf{y} \in \mathcal{C}(\boldsymbol{\gamma}; a, N) \Rightarrow \exists\, b : \overline{\mathbf{x}}, \overline{\mathbf{y}} \in \mathcal{C}(\boldsymbol{\gamma}; b, N) \,.$$

Proof. The proof for $\mathbf{x}$ is identical for $\mathbf{y}$; hence, we focus only on $\mathbf{x}$. From the definition of a $(\boldsymbol{\gamma}; a, N)$-congruent code it follows that

$$\mathbf{x} \in \mathcal{C}(\boldsymbol{\gamma}; a, N) \iff \sum_{i=1}^{n} \gamma_i \cdot x_i \equiv a \mod N \,.$$

We denote $b' = \sum_{i=1}^{n} \gamma_i \mod N$. Since $\mathbf{x} \in \Sigma_2^n$, the syndrome of $\overline{\mathbf{x}}$ satisfies

$$\sum_{i=1}^{n} \gamma_i \cdot \overline{x_i} = \sum_{i=1}^{n} \gamma_i \cdot (1 - x_i) = \sum_{i=1}^{n} \gamma_i - \left(\sum_{i=1}^{n} \gamma_i \cdot x_i \right) \equiv b' - a \mod N \,,$$

which, in turn, is equivalent to the following fact.

$$\overline{\mathbf{x}} \in \mathcal{C}(\boldsymbol{\gamma}; b' - a, N) \,.$$

Since no additional assumptions were made about $\mathbf{x}$, the exact same proof with the same values can be repeated for $\mathbf{y}$. Hence, we can conclude that

$$\overline{\mathbf{x}}, \overline{\mathbf{y}} \in \mathcal{C}(\boldsymbol{\gamma}; b' - a, N)$$

and so there exists such $b \equiv b' - a \bmod N$ such that $\overline{\mathbf{x}}, \overline{\mathbf{y}} \in \mathcal{C}(\boldsymbol{\gamma}; b, N)$. □

4.5 Construction of Binary Single-Deletion Single-Substitution Correcting Codes

In this section, we present our main result. An explicit construction for a binary single-deletion single-substitution correcting code with the redundancy of at most $6 \cdot \log_2(n) + 8$ bits and its correctness is presented. It is based on the intersection of multiple $(\boldsymbol{\gamma}; a, N)$-congruent codes $\mathcal{C}(\boldsymbol{\gamma}; a, N)$ (see Definition 4.2) where each component makes use of different *VT-sketches* (higher-order VT-constraints) introduced by [SRB19]. The proof of the construction was inspired by techniques introduced in [SB20].

Construction 4.1. *Define four weight vectors* $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\eta}, \mathbf{1} \in \mathbb{Z}^n$ *as*

$$\begin{aligned}
\boldsymbol{\alpha} &= (1, 2, 3, \ldots, n)\ , \\
\boldsymbol{\beta} &= \left(\sum_{i=1}^{1} i, \sum_{i=1}^{2} i, \sum_{i=1}^{3} i, \ldots, \sum_{i=1}^{n} i \right)\ , \\
\boldsymbol{\eta} &= \left(\sum_{i=1}^{1} i^2, \sum_{i=1}^{2} i^2, \sum_{i=1}^{3} i^2, \ldots, \sum_{i=1}^{n} i^2 \right)\ , \\
\mathbf{1} &= (1, \ldots, 1)\,.
\end{aligned}$$

For fixed integers $a \in [0, 3n], b \in [0, 3n^2], c \in [0, 3n^3], d \in [0, 4]$*, the code is defined as*

$$\mathcal{C}^{\mathrm{DS}}_{a,b,c,d} = \mathcal{C}(\boldsymbol{\alpha}; a, 3n+1) \cap \mathcal{C}(\boldsymbol{\beta}; b, 3n^2+1) \cap \mathcal{C}(\boldsymbol{\eta}; c, 3n^3+1) \cap \mathcal{C}(\mathbf{1}; d, 5).$$

Remember that d_x and e_x denote the deleted and the substituted bit indices in the word $\mathbf{x}$. Correspondingly, the indices d_y and e_y are defined for the word $\mathbf{y}$. The definition of ϵ_x is 1 if $e_x \in [d_x, d_y]$ and 0 otherwise, and the definition of ϵ_y is similar. The following shows that Construction 4.1 is a single-deletion single-substitution correcting code.

Theorem 4.3. *For any indices* $e_x, e_y, d_x, d_y \in [n]$ *and for any two codewords* $\mathbf{x}, \mathbf{y} \in \mathcal{C}^{\mathrm{DS}}_{a,b,c,d}$*, it holds that*

$$\mathbf{x}(d_x, e_x) \neq \mathbf{y}(d_y, e_y)\,.$$

Proof. We will show the proof by contradiction; hence we assume that $\mathbf{x}(d_x, e_x) = \mathbf{y}(d_y, e_y)$. Similar as in Lemmas 4.5 to 4.7, we distinguish different cases of orderings of the variables $d_x, d_y, e_x, e_y \in [n]$. However, we start by providing general statements first. For simplicity, we denote $\widehat{e}_x = e_x - \epsilon_x$ and $\widehat{e}_y = e_y + \epsilon_y$ throughout the proof.

From Properties 4.5–(v) to 4.7–(v), and the fact that $\mathcal{C}^{\mathrm{DS}}_{a,b,c,d} \subseteq \mathcal{C}(\mathbf{1}; d, 5)$ we deduce that

$$\delta_{e_x} + \delta_{\widehat{e}_y} + x_{d_x} - y_{d_y} + 0 \equiv 0 \mod 5\,.$$

This is equivalent to the following system of equations:

$$x_{d_x} = y_{d_y},\ \delta_{e_x} = -\delta_{\widehat{e}_y}\,.$$

Given the first equation and by Claims 4.5 and 4.6, it is possible to assume that $x_{d_x} = y_{d_y} = 0$. Moreover, we can assume without loss of generalization that $e_x < e_y$ according to Claim 4.4. Further, we define the set $\mathcal{S}$ as

$$\mathcal{S} \triangleq \{d_x + 1 \leq i \leq d_y | x_i = 1\} \subseteq [d_x, d_y]\,.$$

Moreover, we observe that the elements of the weight vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\eta}$ at index j can be written as $\sum_{i=1}^{j} i^k$ for $k \in \{0, 1, 2\}$, respectively. We substitute the observations and values mentioned above into the equivalences derived in Properties 4.5–(v) to 4.7–(v) for the fact that $\mathbf{x}, \mathbf{y} \in \mathcal{C}^{\mathrm{DS}}_{a,b,c,d}$. Hence, for any $k \in \{0, 1, 2\}$, it holds that

$$\delta_{e_x} \cdot \sum_{j=1}^{\widehat{e}_x} j^k + \delta_{\widehat{e}_y} \cdot \sum_{j=1}^{e_y} j^k + \sum_{j \in \mathcal{S}} j^k \equiv 0 \mod 3n^{k+1} + 1\,.$$

We point out that $|\delta_{e_x} \cdot \sum_{j=1}^{\widehat{e}_x} j^k| < \sum_{j=1}^{n} j^k < n \cdot n^k = n^{k+1}$, which is also true for $|\delta_{\widehat{e}_y} \cdot \sum_{j=1}^{e_y} j^k|$ and $|\sum_{j \in \mathcal{S}} j^k|$. As a result, the left part of the equivalences is at least $-3 \cdot n^{k+1}$ and at most $3 \cdot n^{k+1}$. Therefore, the congruences are strict equalities. We reformulate the obtained equalities as

$$\sum_{j \in \mathcal{S}} j^k = -\delta_{\widehat{e}_y} \cdot (-\sum_{j=1}^{\widehat{e}_x} j^k + \sum_{j=1}^{e_y} j^k) = -\delta_{\widehat{e}_y} \cdot (\sum_{\widehat{e}_x+1}^{e_y} j^k)\,.$$

Notice that the left-hand side is always non-negative. Hence, the right-hand side's sign is non-negative, so we conclude $\delta_{\widehat{e}_y} = -1$. Thus, the equation can be transformed into

$$\sum_{\widehat{e}_x+1}^{e_y} j^k = \sum_{j \in \mathcal{S}} j^k. \tag{4.4}$$

As deduced before, we can assume that $d_x < d_y$ and $\widehat{e}_x < e_y$. Hence, there are six possible orderings of the four indices. We will show a complete proof of the case $\widehat{e}_x, e_y < d_x$ and the case $\widehat{e}_x < d_x < e_y < d_y$. Guidance to prove the rest of the cases can be found afterward.

Case $\widehat{e}_x, e_y < d_x$: Under this assumption, two sets are defined as

$$\mathcal{S}_1 \triangleq [\widehat{e}_x + 1, e_y],\ \mathcal{S}_2 \triangleq \mathcal{S}\,.$$

We note that for any two indices $i \in \mathcal{S}_1$ and $j \in \mathcal{S}_2$ it holds that

$$i < j\,. \tag{4.5}$$

Hence, Equation (4.4) can be altered to the following form. For any $k \in \{0, 1, 2\}$, it holds

$$\sum_{j \in \mathcal{S}_1} j^k = \sum_{j \in \mathcal{S}_2} j^k\,.$$

For $k = 0$, this equality is $|\mathcal{S}_1| = |\mathcal{S}_2|$, which means that the cardinalities of the sets are

equal. For $k = 1$, this equality is $\sum_{j\in\mathcal{S}_1} j = \sum_{j\in\mathcal{S}_2} j$, which in return means that the sum of elements of $\mathcal{S}_1$ and $\mathcal{S}_2$ are equal as well. However, if the cardinality of the sets is the same, according to Equation (4.5), the elements sum in $\mathcal{S}_2$ should be strictly bigger than the elements sum in $\mathcal{S}_1$. This is a contradiction to the code constraints; hence, we can conclude this case.

Case $\widehat{e}_x < d_x < e_y < d_y$: Under this assumption, three sets are defined as

$$\mathcal{S}_1 \triangleq [\widehat{e}_x + 1, d_x],\ \mathcal{S}_2 \triangleq [d_x + 1, e_y] \setminus \mathcal{S},\ \mathcal{S}_3 \triangleq [e_y + 1, d_y] \cap \mathcal{S}\,.$$

We point out that any three indices $i \in \mathcal{S}_1,\ j \in \mathcal{S}_2$, and $\ell \in \mathcal{S}_3$ satisfy the relation

$$i < j < \ell\,. \tag{4.6}$$

In this context, we can rewrite Equation (4.4) as follows. For any $k \in \{0, 1, 2\}$, it holds that

$$\sum_{j\in\mathcal{S}_1} j^k + \sum_{j\in\mathcal{S}_2} j^k - \sum_{j\in\mathcal{S}_3} j^k = 0.$$

This can be written using the subsequent matrix form. Moreover, there exist integers v_1, v_2, v_3 such that at least one is non-zero, and the following equality holds.

$$\mathbf{A}\cdot\mathbf{v} \triangleq \begin{pmatrix} \sum_{j\in\mathcal{S}_1} 1 & \sum_{j\in\mathcal{S}_2} 1 & \sum_{j\in\mathcal{S}_3} 1 \\ \sum_{j\in\mathcal{S}_1} j & \sum_{j\in\mathcal{S}_2} j & \sum_{j\in\mathcal{S}_3} j \\ \sum_{j\in\mathcal{S}_1} j^2 & \sum_{j\in\mathcal{S}_2} j^2 & \sum_{j\in\mathcal{S}_3} j^2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\,.$$

In this case, $v_1 = v_2 = 1, v_3 = -1$ is such a solution. The equality means that $\mathbf{A}$ has a non-trivial solution to the homogeneous system of equalities, which also means $\det(\mathbf{A}) = 0$.

However, we could also directly calculate $\det(\mathbf{A})$. To proceed, we use the multi-linearity property of a determinant, which states that

$$\begin{vmatrix} r\cdot a_1 + b_1 & c & d \\ r\cdot a_2 + b_2 & e & f \\ r\cdot a_3 + b_3 & g & h \end{vmatrix} = r\cdot \begin{vmatrix} a_1 & c & d \\ a_2 & e & f \\ a_3 & g & h \end{vmatrix} + \begin{vmatrix} b_1 & c & d \\ b_2 & e & f \\ b_3 & g & h \end{vmatrix}\,.$$

Thus, the determinant of the matrix $\mathbf{A}$ can be computed by

$$\begin{vmatrix} \sum_{j\in\mathcal{S}_1} 1 & \sum_{j\in\mathcal{S}_2} 1 & \sum_{j\in\mathcal{S}_3} 1 \\ \sum_{j\in\mathcal{S}_1} j & \sum_{j\in\mathcal{S}_2} j & \sum_{j\in\mathcal{S}_3} j \\ \sum_{j\in\mathcal{S}_1} j^2 & \sum_{j\in\mathcal{S}_2} j^2 & \sum_{j\in\mathcal{S}_3} j^2 \end{vmatrix} = \sum_{i\in\mathcal{S}_1}\sum_{j\in\mathcal{S}_2}\sum_{\ell\in\mathcal{S}_3} \begin{vmatrix} 1 & 1 & 1 \\ i & j & \ell \\ i^2 & j^2 & \ell^2 \end{vmatrix}\,.$$

Notice that each element in the sum is a determinant of a Vandermonde matrix. For this type of matrices, it holds that

$$\begin{vmatrix} 1 & 1 & 1 \\ i & j & \ell \\ i^2 & j^2 & \ell^2 \end{vmatrix} = (j - i)\cdot(\ell - j)\cdot(\ell - i)\,.$$

Table 4.1: The definitions for the sets $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ for all necessary orderings of the indices $\widehat{e}_x, e_y, d_x, d_y \in [n]$ to show the contradiction in Equation (4.7). Recall that $\widehat{e}_x = e_x - \epsilon_x$ and $\mathcal{S} \triangleq \{d_x + 1 \leq i \leq d_y | x_i = 1\} \subseteq [d_x, d_y]$.

Case	Ordering	$\mathcal{S}_1 \triangleq$	$\mathcal{S}_2 \triangleq$	$\mathcal{S}_3 \triangleq$
1	$\widehat{e}_x < e_y < d_x < d_y$	$[\widehat{e}_x + 1, e_y]$	$\mathcal{S}$	$\emptyset$
2	$\widehat{e}_x < d_x < e_y < d_y$	$[\widehat{e}_x + 1, d_x]$	$[d_x + 1, e_y] \setminus \mathcal{S}$	$[e_y + 1, d_y] \cap \mathcal{S}$
3	$\widehat{e}_x < e_y < d_x < d_y$	$[\widehat{e}_x + 1, e_y]$	$\mathcal{S}$	$\emptyset$
4	$\widehat{e}_x < d_x < e_y < d_y$	$[\widehat{e}_x + 1, d_x]$	$[d_x + 1, e_y] \setminus \mathcal{S}$	$[e_y + 1, d_y] \cap \mathcal{S}$
5	$\widehat{e}_x < d_x < d_y < e_y$	$[\widehat{e}_x + 1, d_x]$	$\mathcal{S}$	$[d_y + 1, e_y]$
6	$d_x < \widehat{e}_x < e_y < d_y$	$\mathcal{S} \cap [d_x, \widehat{e}_x]$	$[\widehat{e}_x + 1, e_y] \setminus \mathcal{S}$	$[e_y + 1, d_y] \cap \mathcal{S}$
7	$d_x < \widehat{e}_x < d_y < e_y$	$\mathcal{S} \cap [d_x, \widehat{e}_x]$	$[\widehat{e}_x + 1, e_y] \setminus \mathcal{S}$	$[d_y + 1, e_y]$
8	$d_x < d_y < \widehat{e}_x < e_y$	$\mathcal{S}$	$[\widehat{e}_x + 1, e_y]$	$\emptyset$

Consequently, we obtain the following contradiction due to Equation (4.6):

$$0 = \sum_{i \in \mathcal{S}_1} \sum_{j \in \mathcal{S}_2} \sum_{\ell \in \mathcal{S}_3} \begin{vmatrix} 1 & 1 & 1 \\ i & j & \ell \\ i^2 & j^2 & \ell^2 \end{vmatrix} = \sum_{i \in \mathcal{S}_1} \sum_{j \in \mathcal{S}_2} \sum_{\ell \in \mathcal{S}_3} (j - i) \cdot (\ell - j) \cdot (\ell - i) > 0 \,. \tag{4.7}$$

This concludes this case.

The same proof which is used to show the latter case can be concluded for all other orderings. We provide the required definitions of the three sets $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ in Table 4.1. Note we have shown explicitly Case 1 and 2 of the table and provide the guidance for the proofs via the definitions of the three sets for the other cases. This concludes the proof. □

We have shown that $\mathcal{C}^{\mathrm{DS}}_{a,b,c,d}$ guarantees to correct single-deletion and single-substitution errors. Via the pigeonhole principle, the following conclusion about the redundancy is obtained.

Corollary 4.3. *There exist $a \in [0, 3n], b \in [0, 3n^2], c \in [0, 3n^3], d \in [0, 4]$ such that the code $\mathcal{C}^{\mathrm{DS}}_{a,b,c,d}$ is a binary single-deletion single-substitution correcting code with at most $6 \cdot \log_2(n) + 8$ redundancy bits.*

Proof. The sets $\mathcal{C}^{\mathrm{DS}}_{a,b,c,d}$ form a partition of the whole space Σ_2^n, where $|\Sigma_2^n| = 2^n$. Notice, there are $3n + 1$ options for a, $3n^2 + 1$ options for b, $3n^3 + 1$ options for c, and 5 options for d. Because of that, there is a total number of $(3n + 1) \cdot (3n^2 + 1) \cdot (3n^3 + 1) \cdot 5$ disjoint sets $\mathcal{C}^{\mathrm{DS}}_{a,b,c,d}$. By the pigeonhole principle, there exists a choice of a, b, c, d such that

$$|\mathcal{C}^{\mathrm{DS}}_{a,b,c,d}| \geq \frac{2^n}{(3n + 1) \cdot (3n^2 + 1) \cdot (3n^3 + 1) \cdot 5} \,.$$

By the definition of redundancy $r^{\mathrm{DS}} \triangleq n - \log_2 \left(|\mathcal{C}^{\mathrm{DS}}_{a,b,c,d}| \right)$ and using logarithm rules, we can

compute an upper bound on the redundancy as follows.

$$\begin{aligned}
r^{\mathrm{DS}} &\leq n - \log_2\left(\frac{2^n}{(3n+1)\cdot(3n^2+1)\cdot(3n^3+1)\cdot 5}\right) \\
&= \log_2(3n+1) + \log_2(3n^2+1) + \log_2(3n^3+1) + \log_2(5) \\
&\leq \log_2(3.5n) + \log_2(3.5n^2) + \log_2(3.5n^3) + \log_2(5) \\
&= \log_2(3.5) + \log_2(n) + \log_2(3.5) + \log_2(n^2) + \log_2(3.5) + \log_2(n^3) + \log_2(5) \\
&= 3\cdot\log_2(3.5) + \log_2(5) + \log_2(n) + 2\cdot\log_2(n) + 3\cdot\log_2(n) \\
&\leq 8 + 6\log_2(n)\,.
\end{aligned}$$

This concludes the proof. □

In the proof of Theorem 4.3, the $\mathbf{1} = (1, \ldots, 1)$ weight vector is only used for simplicity purposes, as it directly leads to the fact that the following equalities hold:

$$x_{d_x} = y_{d_y},\ \delta_{e_x} = -\delta_{\widehat{e_y}}\,.$$

One might notice that this weight vector is, in fact, redundant for single-deletion single-substitution error-correcting purposes. This was shown explicitly in [SPC+20], where the authors presented a slight improvement to Construction 4.1 that can correct a single-deletion single-substitution error with the redundancy of $6\log_2 n + 3$ bits using only the $\boldsymbol{\alpha}, \boldsymbol{\beta}$, and $\boldsymbol{\eta}$ weight vectors.

4.6 Conclusion

In this chapter, we have considered the problem of correcting combinations of deletions and substitutions. First, we have presented non-asymptotic upper bounds on the cardinality of a non-binary single-deletion s-substitution correcting code. Subsequently, a non-asymptotic lower bound on the redundancy of such codes is approximately $r \geq (s+1)\log_q(n)$. Further, we have shown general properties of number-theoretic codes when codewords are corrupted by a single-deletion single-substitution error. We have leveraged these properties to construct a binary single-deletion single-substitution correcting code with redundancy at most $6\log_2(n) + 8$. The construction of the presented code is not explicit in the sense of providing an encoder and decoder, but only in terms of presenting explicit constraints for a single-deletion single-substitution correcting code. The problem of finding an encoder and decoder for this construction is an interesting problem for future work and research.

On a final note, the work in [SW24b] has extended the results of deriving bounds on the cardinality of q-ary codes correcting t insdels and s substitutions of length n to be asymptotically at least $(t+s)\log_q(n)$. Moreover, the work in [SPC+20] provided a systematic construction for correcting a single deletion and s substitutions with redundancy $(3s + 4)\log_2 n + o(\log_2 n)$ for $s \geq 1$. Recently, the authors generalized their construction to the case for multiple insdels with redundancy $(4t+3s)\log_2 n + o(\log_2 n)$ [SPC+22] and also provide a method to construct q-ary codes. In the same line of thought, [GGR+22] has provided a linear-time encodable and list-decodable binary code with list-size 2 correcting a single-

deletion and single-substitution error with the redundancy of $4\log_2 n + O(\log_2 \log_2 n)$. This result has been improved by [SCN22] to $3\log_2 n + 4$. However, the problem of constructing optimal codes remains an open challenge.

Chapter 5

Criss-Cross Insertion and Deletion Errors

Abstract

We investigate the problem of correcting multiple criss-cross insertions and deletions in arrays. More precisely, we study the unique recovery of $n \times n$ arrays affected by *t-criss-cross deletions* defined as any combination of t_r row and t_c column deletions such that $t_\mathrm{r} + t_\mathrm{c} = t$ for a given t. We show an equivalence between correcting t-criss-cross deletions and t-criss-cross insertions and show that a code correcting t-criss-cross insertions/deletions has redundancy asymptotically at least $tn + t \log n - \log(t!)$. A a code construction with an existential encoding and an explicit decoding algorithm is presented for a simplified pattern, i.e., a fixed $(1, 1)$-criss-cross insertion/deletion error. Then, we present an existential construction of a t-criss-cross insertion/deletion correcting code with redundancy bounded from above by $tn + \mathcal{O}(t^2 \log^2 n)$. The main code construction ingredients are systematic binary t-deletion correcting codes and Gabidulin codes. The first ingredient helps to locate the indices of the inserted/deleted rows and columns, thus transforming the insertion/deletion-correction problem into a row/column erasure-correction problem, which is then solved using the second ingredient.

This chapter is based on the work [BWS+21; WBWZ+22; SWB+22]. Lorenz Welter contributed to the conception of the projects, the derivation of the results, and the writing of the manuscripts.

5.1 Introduction

This chapter considers the coding problem for insertion/deletion errors in the two-dimensional space. The motivation stems from the fact that coding in the two-dimensional space has proved profitable for data storage and wireless communications [Sid76; Rot91; BB97; Rot97; LGH00; GP08; WZ16]. In particular within DNA storage applications, systems using Illumina sequencing could profit due to the array-like reading mechanism. Moreover, two-dimensional deletion coding has proven beneficial for racetrack memories [CHK21]. Therefore, it is vital to understand the generalization of the well-studied one-dimensional insertion/deletion correcting codes to the two-dimensional space. The main advantage of coding in the two-dimensional space is to leverage the structure of the code arrays rather than applying one-dimensional insertion/deletion correcting codes on each dimension of the array. The deletion (and clearly also the insdel) correction problem is more involved due to the loss of synchronization in the locations of the inserted and deleted rows and columns. Along this line of thought, the trace-reconstruction problem is investigated for the two-dimensional space in [KMM+21; ZZ24]. Moreover, coding for deletions over the two-dimensional space is also

considered in [CSF+14; SWZG+17; SSAG+17; BE24]. In [CSF+14; SWZG+17; SSAG+17] codes that correct bursts of deletions in the one-dimensional space are constructed. The main idea is to view the codeword as a binary array and use the structure of that array to detect and correct bursts of deletions that happen in the one-dimensional codeword. In [BE24], the authors consider the problem of database matching under synchronization errors.

Given a certain number of deletions t and an array $\mathbf{X}$, we assume that the array can be affected by any combination of t_r row and t_c column deletions such that $t_\mathrm{r} + t_\mathrm{c} = t$. This type of deletions is called *t-criss-cross deletions*. We define *t-criss-cross insertions* similarly. Our goal is to construct codes that can uniquely recover the array $\mathbf{X}$ from any t-criss-cross deletion or any t-criss-cross insertion, and we refer to these codes as *t-criss-cross insertion/deletion correcting codes*. We borrow this terminology from previous works that studied the problem of correcting criss-cross erasures and substitution errors in the two-dimensional space. In [Hag20], Hagiwara constructed codes correcting criss-cross deletions with at most t_r row deletions and at most t_c column deletions for given values of t_r and t_c. The constructed codes have redundancy in the order of $n(t_\mathrm{r}^2 + t_\mathrm{c}^2 + (t_\mathrm{r} + t_\mathrm{c}) \log_2 n)$. The construction splits the array into locators and information parts. The locators are carefully structured arrays that can exactly recover the index of any deleted rows and columns in the array. Then, a tensor-product erasure-correcting code is used to recover the lost symbols in the information part.

Our contribution. First, we extend the well-known insertion/deletion equivalence from vectors to arrays (see Section 5.3). Subsequently, we present an asymptotic upper bound (in the code length) on the cardinality of t-criss-cross insertion/deletion correcting codes (see Section 5.4). Our bound implies that the redundancy of any t-criss-cross insertion/deletion correcting code is bounded from below by approximately $tn+t \log_2 n$. For our code construction, we also follow the strategy of first locating the errors and subsequently correcting them. To better grasp the code construction concept, we first present a simplified code construction for the problem of correcting a fixed combination of one row and one column deletion or insertion error (see Section 5.5). Then, we construct existential t-criss-cross insertion/deletion correcting codes based on locator arrays, binary systematic t-deletion correcting codes, and Gabidulin codes (see Section 5.6). We also show that this code can correct t-criss-cross insertions by providing an explicit decoder (see Section 5.7). The main improvements of our construction over the one in [Hag20] is to use a collection of binary deletion correcting codes to locate the indices of the deleted columns and a Gabidulin code to correct the erasures. This significantly reduces the redundancy of the code. However, small locator arrays are still needed to complement the deletion correcting codes. Then, the deletion-correction problem is transformed into a row/column erasure-correction problem, which can be solved by using Gabidulin codes that have optimal redundancy for row/column erasure-correction [GP08]. The redundancy of the presented construction is $tn + O(t^2 \log_2^2 n)$ (see Section 5.8). For the considered problem setting, we substantially improve upon the construction of [Hag20] that needs a redundancy of approximately $2n \cdot (t^2 + t \log_2 n)$ in this setting.[1]

[1] [Hag20]'s construction is applicable for arbitrary $n \times m$ arrays, while our work focuses on squared arrays.

5.2 Definitions and Preliminaries

This section formally defines the codes and notations used throughout this chapter.

Recall that for an array $\mathbf{X} \in \Sigma_q^{n\times n}$ and $i, j \in [n]$, we refer to the entry of $\mathbf{X}$ positioned at the i^{th} row and the j^{th} column by capitalized $\mathbf{X}_{i,j}$ to emphasize that it is an element of an array. We denote the i^{th} row and the j^{th} column of $\mathbf{X}$ by $\mathbf{X}_{i,[n]}$ and $\mathbf{X}_{[n],j}$, respectively. Similarly, we denote by $\mathbf{X}_{[i_1:i_2],[j_1:j_2]}$ the subarray of $\mathbf{X}$ formed by rows i_1 to i_2 and their corresponding entries from columns j_1 to j_2. Moreover, for two arrays $\mathbf{X} \in \Sigma_q^{n\times m_1}$ and $\mathbf{Y} \in \Sigma_q^{n\times m_2}$ we denote by $\mathbf{Z} = (\mathbf{X} \parallel \mathbf{Y})$ the concatenation of these two arrays with $\mathbf{Z} \in \Sigma_q^{n\times(m_1+m_2)}$. In an array $\mathbf{X} \in \Sigma^{n\times m}$, a column-run of length r is defined as a sequence of r consecutive equal columns $\mathbf{X}_{[n],j} = \mathbf{X}_{[n],j+1} = \cdots = \mathbf{X}_{[n],j+r-1}$. Row-runs in an array $\mathbf{X}$ are defined similarly. Recall, given a vector $\mathbf{x} \in \Sigma_q^n$ a run of length r in $\mathbf{x}$ is defined as a sequence of r consecutive equal bits $x_i = x_{i+1} = \ldots = x_{i+r-1}$.

Formally, for positive integers $t_{\mathrm{r}}, t_{\mathrm{c}} \leq n$, we define a $(t_{\mathrm{r}}, t_{\mathrm{c}})$-*criss-cross deletion* in an array $\mathbf{X}$ to be the deletion of any t_{r} rows and t_{c} columns of $\mathbf{X} \in \Sigma_q^{n\times n}$. For a positive integer t, we define a *t-criss-cross deletion* in an array $\mathbf{X}$ to be the collection of all $(t_{\mathrm{r}}, t_{\mathrm{c}})$-criss-cross deletions in $\mathbf{X}$ such that $t_{\mathrm{r}} + t_{\mathrm{c}} = t$. Further, $(t_{\mathrm{r}}, t_{\mathrm{c}})$-*criss-cross insertion* and a *t-criss-cross insertion* are defined similarly. We denote by $\mathbb{D}_{t_{\mathrm{r}},t_{\mathrm{c}}}(\mathbf{X})$ the set of all arrays that result from $\mathbf{X}$ after a $(t_{\mathrm{r}}, t_{\mathrm{c}})$-criss-cross deletion (i.e., the two-dimensional deletion ball). Similarly, we define the set $\mathbb{I}_{t_{\mathrm{r}},t_{\mathrm{c}}}(\mathbf{X})$ for the insertion case. We refer to $\widetilde{\mathbf{X}}$ as the array resulting from a t-criss-cross deletion or insertion in $\mathbf{X}$, where the number and type of errors (deletions or insertions) that happened in $\mathbf{X}$ is evident from the context. To better understand the error model, we illustrate the possible deletion patterns in a criss-cross deletion error in Figure 5.1.

Definition 5.1 ($(t_{\mathrm{r}}, t_{\mathrm{c}})$-criss-cross deletion correcting code)**.** *A $(t_{\mathrm{r}}, t_{\mathrm{c}})$-criss-cross deletion correcting code $\mathcal{C}^{\mathrm{cc}}_{(t_{\mathrm{r}},t_{\mathrm{c}})}$ is a code that can correct any $(t_{\mathrm{r}}, t_{\mathrm{c}})$-criss-cross deletion. A $(t_{\mathrm{r}}, t_{\mathrm{c}})$-criss-cross insertion correcting code is defined similarly.*

Definition 5.2 (t-criss-cross deletion correcting code)**.** *A t-criss-cross deletion correcting code $\mathcal{C}^{\mathrm{cc}}_{t}$ is a code that can correct any t-criss-cross deletion. A t-criss-cross insertion correcting code is defined similarly.*

Throughout this chapter, we assume that $t_{\mathrm{r}} + t_{\mathrm{c}} = t$ is a constant with respect to n. Moreover, we emphasize that we do not consider combinations of insertions and deletions.

5.3 Insertion/Deletion Equivalence for Criss-Cross Errors

We study the insertion and deletion equivalence for arrays. This property is well understood in the vector case as proven already by Levenshtein in 1966 [Lev66b] (see Theorem 3.1). To show the array equivalence, we first derive a preliminary result for criss-cross insertion and deletions with equal row and column deletions/insertions (see Section 5.3.1). Then, we will show that a t-criss-cross deletion code is equivalently also a t-criss-cross insertion correcting code (see Section 5.3.2).

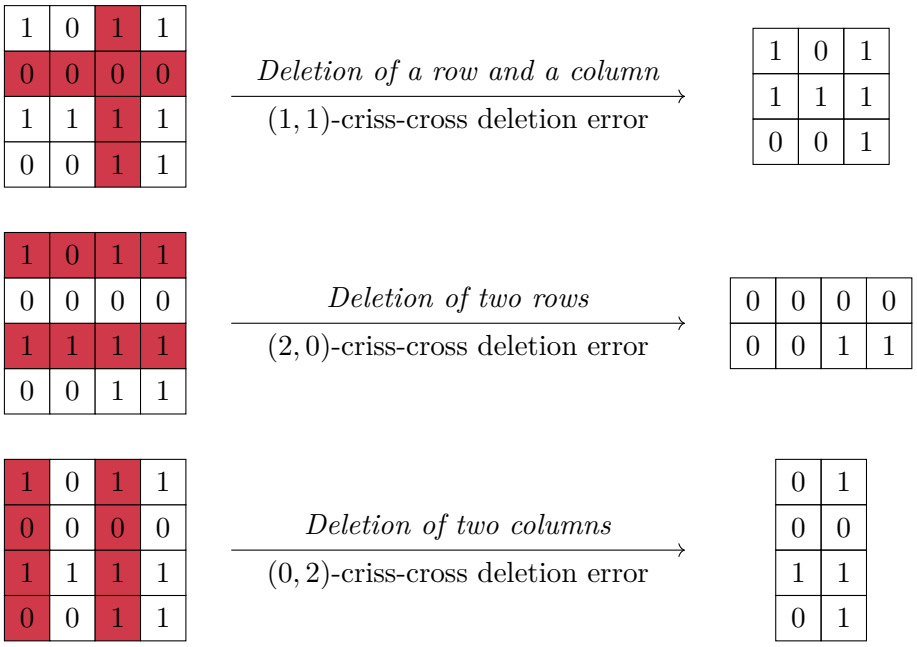


Figure 5.1: Illustration of the variations of the 2-criss-cross error.

5.3.1 Equal Row and Column Deletion and Insertion Equivalence

In this subsection, we first show the equivalence between a $(1,1)$-criss-cross deletion correcting code and a $(1,1)$-criss-cross insertion correcting code (Theorem 5.1). Then we use the result of Theorem 5.1 to prove the more general equivalence between (t,t)-criss-cross deletion correcting codes and (t,t)-criss-cross insertion correcting codes for all $t \in [n-1]$ (Corollary 5.1).

Theorem 5.1. *A code $\mathcal{C}^{\mathrm{cc}}_{(1,1)} \subseteq \Sigma_q^{n\times n}$ is a $(1,1)$-criss-cross deletion correcting code if and only if it is a $(1,1)$-criss-cross insertion correcting code.*

Corollary 5.1. *For any integer $t \in [n-1]$, a code $\mathcal{C}^{\mathrm{cc}}_{(t,t)} \subseteq \Sigma_q^{n\times n}$ is a (t,t)-criss-cross deletion correcting code if and only if it is a (t,t)-criss-cross insertion correcting code.*

Note that in the one-dimensional case, the equivalent formulation of Theorem 5.1 holds since the intersection of the deletion balls of two vectors is not trivial if and only if the intersection of their insertion balls is not trivial [Lev66b]. Since this property holds over any alphabet, the following lemma can be derived by considering the arrays as one-dimensional vectors where each element is a row/column.

Lemma 5.1 (see Theorem 3.1 or [Lev66b])**.** *For a positive integer m and two arrays $\mathbf{X} \in \Sigma_q^{m\times m}, \mathbf{Y} \in \Sigma_q^{m\times m}$,*

$$\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{1,0}(\mathbf{Y}) \neq \emptyset \iff \mathbb{I}_{1,0}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset\,,$$
$$\mathbb{D}_{0,1}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) \neq \emptyset \iff \mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{0,1}(\mathbf{Y}) \neq \emptyset.$$

While the last lemma is derived from properties of vectors, the next one, albeit similar, requires a complete proof.

Lemma 5.2. *For a positive integer m and two arrays $\mathbf{X} \in \Sigma_q^{(m+1)\times m}$, $\mathbf{Y} \in \Sigma_q^{m\times(m+1)}$,*

$$\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) \neq \emptyset \iff \mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset.$$

Proof. We show the "if" direction while the "only if" part is proved similarly. That is, we prove that if $\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) \neq \emptyset$ then $\mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset$. Assume that there exists $\mathbf{D} \in \Sigma_q^{m\times m}$ such that $\mathbf{D} \in \mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y})$ and by contradiction assume that $\mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) = \emptyset$.

Let $i_{\mathrm{r}}, j_{\mathrm{c}}$ be the row and column indices deleted in $\mathbf{X}$ and $\mathbf{Y}$, respectively, to obtain $\mathbf{D}$. Let $\mathbf{r}$ denote $i_{\mathrm{r}}^{\mathrm{th}}$ row of $\mathbf{X}$, i.e., $\mathbf{X}_{i_{\mathrm{r}},[m]}$, after an insertion of 0 in position j_{c}. Similarly, let $\mathbf{c}$ be the column $\mathbf{Y}_{[m],j_{\mathrm{c}}}$ after an insertion of 0 in position i_{r}. Notice that it is also possible to insert any symbol $a \in \Sigma_q$ in both words, as long as the symbol inserted is the same. The following relations hold from the definition of $\mathbf{D}$:

$$\begin{aligned}
&\mathbf{X}_{i,j} = \mathbf{D}_{i,j} = \mathbf{Y}_{i,j} \text{ for } 1 \leq i < i_{\mathrm{r}}, 1 \leq j < j_{\mathrm{c}},\\
&\mathbf{X}_{i+1,j} = \mathbf{D}_{i,j} = \mathbf{Y}_{i,j} \text{ for } i_{\mathrm{r}} \leq i \leq m, 1 \leq j < j_{\mathrm{c}},\\
&\mathbf{X}_{i,j} = \mathbf{D}_{i,j} = \mathbf{Y}_{i,j+1} \text{ for } 1 \leq i < i_{\mathrm{r}}, j_{\mathrm{c}} \leq j \leq m,\\
&\mathbf{X}_{i+1,j} = \mathbf{D}_{i,j} = \mathbf{Y}_{i,j+1} \text{ for } i_{\mathrm{r}} \leq i \leq m, j_{\mathrm{c}} \leq j \leq m.
\end{aligned} \tag{5.1}$$

Let $\mathbf{I}^{\mathbf{X}}$ be the result of inserting column $\mathbf{c}$ at index j_{c} into $\mathbf{X}$. The array $\mathbf{I}^{\mathbf{Y}}$ is defined similarly by inserting row $\mathbf{r}$ at index i_{r} in $\mathbf{Y}$. Notice that $\mathbf{I}^{\mathbf{X}}$ is a result of inserting a column to $\mathbf{X}$ and thus $\mathbf{I}^{\mathbf{X}} \in \mathbb{I}_{0,1}(\mathbf{X})$. For the same reasons, it holds that $\mathbf{I}^{\mathbf{Y}} \in \mathbb{I}_{1,0}(\mathbf{Y})$. We conclude the proof by showing that $\mathbf{I}^{\mathbf{X}} = \mathbf{I}^{\mathbf{Y}}$. This will be done by considering the following cases.

1. For $i < i_{\mathrm{r}}$ and for $j < j_{\mathrm{c}}$:
   Both $\mathbf{I}^{\mathbf{X}}_{i,j}, \mathbf{I}^{\mathbf{Y}}_{i,j}$ are unaffected by the insertions or deletions. Hence, it follows that
   $$\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j} = \mathbf{Y}_{i,j} = \mathbf{I}^{\mathbf{Y}}_{i,j}\,.$$

2. For $i = i_{\mathrm{r}}$ and for $j < j_{\mathrm{c}}$:
   The symbols $\mathbf{I}^{\mathbf{X}}_{i,j} = r_j$ remain unaffected by the insertion. On the other hand, $\mathbf{I}^{\mathbf{Y}}_{i,j}$ is exactly an inserted symbol into $\mathbf{Y}$ that is defined to be r_j which results in
   $$\mathbf{I}^{\mathbf{Y}}_{i,j} = r_j = \mathbf{I}^{\mathbf{X}}_{i,j}\,.$$

3. For $i < i_{\mathrm{r}}$ and for $j = j_{\mathrm{c}}$:
   The symbols $\mathbf{I}^{\mathbf{Y}}_{i,j} = c_i$ remain unaffected by the insertion. On the other hand, $\mathbf{I}^{\mathbf{X}}_{i,j}$ is exactly an inserted symbol into $\mathbf{X}$ that is defined to be c_i which results in
   $$\mathbf{I}^{\mathbf{X}}_{i,j} = c_i = \mathbf{I}^{\mathbf{Y}}_{i,j}\,.$$

4. For $i = i_{\mathrm{r}}$ and for $j = j_{\mathrm{c}}$:
   It holds that $\mathbf{I}^{\mathbf{X}}_{i,j} = c_i$ and $\mathbf{I}^{\mathbf{Y}}_{i,j} = r_j$. By definition, both of these symbols are 0, which

results in

$$\mathbf{I}^{\mathbf{X}}_{i,j} = c_i = r_j = \mathbf{I}^{\mathbf{Y}}_{i,j} .$$

5. For $i > i_{\mathrm{r}}$ and for $j < j_{\mathrm{c}}$:
   By Equation (5.1) it holds that $\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j} = \mathbf{D}_{i-1,j} = \mathbf{Y}_{i-1,j}$. On the other hand, after a row insertion in index i_{r}, it holds that $\mathbf{I}^{\mathbf{Y}}_{i,j} = \mathbf{Y}_{i-1,j}$ which results in

$$\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j} = \mathbf{D}_{i-1,j} = \mathbf{Y}_{i-1,j} = \mathbf{I}^{\mathbf{Y}}_{i,j} .$$

6. For $i > i_{\mathrm{r}}$ and for $j = j_{\mathrm{c}}$:
   By definition $\mathbf{I}^{\mathbf{X}}_{i,j} = c_i = \mathbf{Y}_{i-1,j}$. On the other hand, $\mathbf{I}^{\mathbf{Y}}$ had a row insertion in index i_{r}, which means that $\mathbf{I}^{\mathbf{Y}}_{i,j} = \mathbf{Y}_{i-1,j}$ and results in

$$\mathbf{I}^{\mathbf{X}}_{i,j} = c_i = \mathbf{Y}_{i-1,j} = \mathbf{I}^{\mathbf{Y}}_{i,j} .$$

7. For $i < i_{\mathrm{r}}$ and for $j > j_{\mathrm{c}}$:
   By Equation (5.1) it holds that $\mathbf{I}^{\mathbf{Y}}_{i,j} = \mathbf{Y}_{i,j} = \mathbf{D}_{i,j-1} = \mathbf{X}_{i,j-1}$. On the other hand, after a column insertion in index j_{c}, it holds that $\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j-1}$ which results in

$$\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j-1} = \mathbf{D}_{i,j-1} = \mathbf{Y}_{i,j} = \mathbf{I}^{\mathbf{Y}}_{i,j} .$$

8. For $i = i_{\mathrm{r}}$ and for $j > j_{\mathrm{c}}$:
   By definition $\mathbf{I}^{\mathbf{Y}}_{i,j} = r_j = \mathbf{X}_{i,j-1}$. On the other hand, $\mathbf{I}^{\mathbf{X}}$ had a column insertion in index j_{c}, which means that $\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j-1}$ and results in

$$\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j-1} = r_j = \mathbf{I}^{\mathbf{Y}}_{i,j} .$$

9. For $i > i_{\mathrm{r}}$ and for $j > j_{\mathrm{c}}$:
   The array $\mathbf{I}^{\mathbf{X}}$ had a column insertion in index j_{c}, which means that $\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j-1}$. On the other hand, $\mathbf{I}^{\mathbf{Y}}$ had a row insertion in index i_{r}, which means that $\mathbf{I}^{\mathbf{Y}}_{i,j} = \mathbf{Y}_{i-1,j}$. From Equation (5.1) it holds that $\mathbf{X}_{i,j-1} = \mathbf{D}_{i-1,j-1}$ and $\mathbf{Y}_{i-1,j} = \mathbf{D}_{i-1,j-1}$. This results in

$$\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{X}_{i,j-1} = \mathbf{D}_{i-1,j-1} = \mathbf{Y}_{i-1,j} = \mathbf{I}^{\mathbf{Y}}_{i,j}.$$

This concludes that for all $i, j \in [m+1]$, $\mathbf{I}^{\mathbf{X}}_{i,j} = \mathbf{I}^{\mathbf{Y}}_{i,j}$, which assures that $\mathbf{I}^{\mathbf{X}} = \mathbf{I}^{\mathbf{Y}}$, and hence $\mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y}) \neq \emptyset$, that contradicts our assumption. □

We now use the results of Lemmas 5.1 and 5.2 to prove Theorem 5.1.

Proof of Theorem 5.1. The proof is shown via contraposition, i.e., showing that for any $\mathbf{X}, \mathbf{Y} \in \Sigma_q^{n \times n}$, $\mathbb{D}_{1,1}(\mathbf{X}) \cap \mathbb{D}_{1,1}(\mathbf{Y}) \neq \emptyset$ if and only if $\mathbb{I}_{1,1}(\mathbf{X}) \cap \mathbb{I}_{1,1}(\mathbf{Y}) \neq \emptyset$. For the reader's convenience, a flowchart of the proof is presented in Figure 5.2. We only show the "only if" part as the "if" part follows similarly.

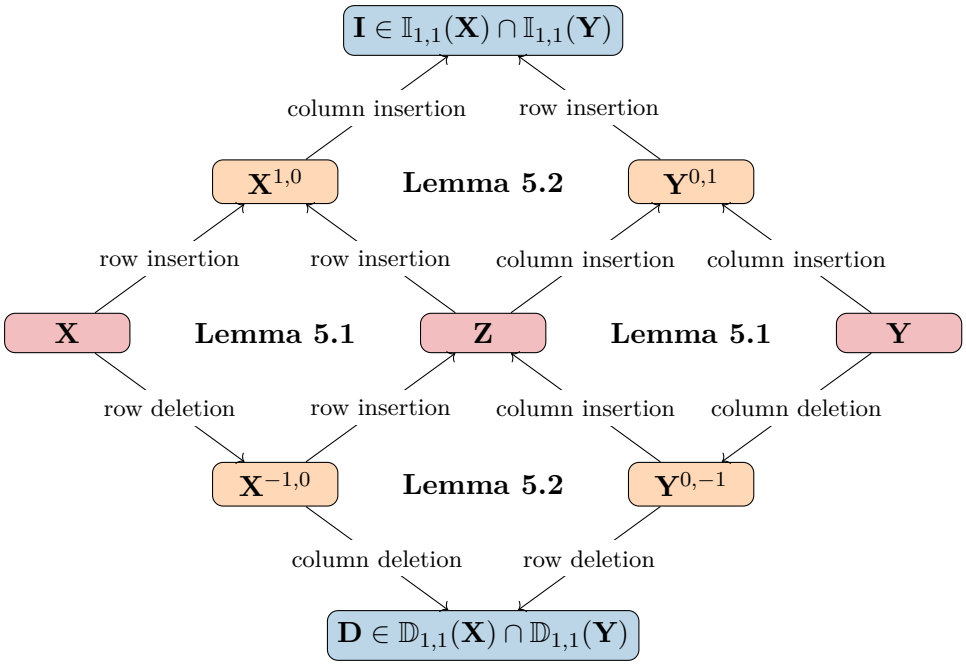


Figure 5.2: A flowchart of the proof of Theorem 5.1.

We assume that there exists an array $\mathbf{D} \in \Sigma_q^{(n-1)\times(n-1)}$ such that $\mathbf{D} \in \mathbb{D}_{1,1}(\mathbf{X}) \cap \mathbb{D}_{1,1}(\mathbf{Y})$. Hence, $\mathbf{D}$ can be obtained by deleting a row and then a column from $\mathbf{X}$, and by deleting a column and then a row from $\mathbf{Y}$.[2] We denote the intermediate arrays by $\mathbf{X}^{-1,0}$ and $\mathbf{Y}^{0,-1}$, so the following relations hold.

$$\mathbf{X} \xrightarrow{\text{row deletion}} \mathbf{X}^{-1,0} \xrightarrow{\text{column deletion}} \mathbf{D}\,,$$
$$\mathbf{Y} \xrightarrow{\text{column deletion}} \mathbf{Y}^{0,-1} \xrightarrow{\text{row deletion}} \mathbf{D}\,.$$

Consequently, it holds that

$$\mathbf{D} \in \mathbb{D}_{1,0}(\mathbf{Y}^{0,-1}) \cap \mathbb{D}_{0,1}(\mathbf{X}^{-1,0})\,,$$

and thus, according to Lemma 5.2 there exists an array $\mathbf{Z} \in \Sigma_q^{n\times n}$, such that

$$\mathbf{Z} \in \mathbb{I}_{0,1}(\mathbf{Y}^{0,-1}) \cap \mathbb{I}_{1,0}(\mathbf{X}^{-1,0})\,.$$

By definition, $\mathbf{Z} \in \mathbb{I}_{1,0}(\mathbf{X}^{-1,0})$ is equivalent to $\mathbf{X}^{-1,0} \in \mathbb{D}_{1,0}(\mathbf{Z})$. But, it is also known that $\mathbf{X}^{-1,0} \in \mathbb{D}_{1,0}(\mathbf{X})$, which means that

$$\mathbf{X}^{-1,0} \in \mathbb{D}_{1,0}(\mathbf{Z}) \cap \mathbb{D}_{1,0}(\mathbf{X}).$$

[2] The order of the row and column deletions can be chosen arbitrarily since a similar proof can be derived for a different order.

From Lemma 5.1, it follows that there exists some

$$\mathbf{X}^{1,0} \in \mathbb{I}_{1,0}(\mathbf{Z}) \cap \mathbb{I}_{1,0}(\mathbf{X}) \,.$$

The same chain of arguments can be done for $\mathbf{Y}$; we define its result by $\mathbf{Y}^{0,1}$. Due to the aforementioned derivation of $\mathbf{X}^{1,0}$ and $\mathbf{Y}^{0,1}$, we can conclude that

$$\mathbf{Z} \in \mathbb{D}_{1,0}(\mathbf{X}^{1,0}) \cap \mathbb{D}_{0,1}(\mathbf{Y}^{0,1}) \,.$$

By Lemma 5.2, we deduce that there exists an array

$$\mathbf{I} \in \mathbb{I}_{0,1}(\mathbf{X}^{1,0}) \cap \mathbb{I}_{1,0}(\mathbf{Y}^{0,1}) \,.$$

Observe that $\mathbf{I} \in \mathbb{I}_{0,1}(\mathbf{X}^{1,0})$ and $\mathbf{X}^{1,0} \in \mathbb{I}_{1,0}(\mathbf{X})$, which means $\mathbf{I}$ is obtained by inserting a row and a column in $\mathbf{X}$, i.e., $\mathbf{I} \in \mathbb{I}_{1,1}(\mathbf{X})$. A symmetrical argument holds for $\mathbf{Y}$, which assures that $\mathbf{I} \in \mathbb{I}_{1,1}(\mathbf{X}) \cap \mathbb{I}_{1,1}(\mathbf{Y})$. □

Before we prove Corollary 5.1, we will show two claims which facilitate its proof. The proofs will use the result given in Theorem 5.1.

Claim 5.1. *For any two arrays $\mathbf{X}_1, \mathbf{X}_{t+1} \in \Sigma_q^{n\times n}$, $\mathbb{D}_{t,t}(\mathbf{X}_1) \cap \mathbb{D}_{t,t}(\mathbf{X}_{t+1}) \neq \emptyset$ if and only if there exist $t-1$ arrays $\mathbf{X}_2, \ldots, \mathbf{X}_t$ such that $\mathbb{D}_{1,1}(\mathbf{X}_i) \cap \mathbb{D}_{1,1}(\mathbf{X}_{i+1}) \neq \emptyset$ for all $1 \leq i \leq t$.*

Proof. We prove the "if" part by induction. The proof of the "only if" part follows similarly and is omitted.

Base case: We need to show that if $\mathbb{D}_{1,1}(\mathbf{X}_1) \cap \mathbb{D}_{1,1}(\mathbf{X}_2) \neq \emptyset$ then $\mathbb{D}_{1,1}(\mathbf{X}_i) \cap \mathbb{D}_{1,1}(\mathbf{X}_{i+1}) \neq \emptyset$ for all $i = 1$ which follows from the assumption.

Induction hypothesis: We use the statement of the claim as the induction hypothesis.

Induction step: Assume the property holds for $t \in [n-2]$ and we show that the property holds for $t+1$. Let $\mathbf{X}_1, \mathbf{X}_{t+2}$ be such that $\mathbb{D}_{t+1,t+1}(\mathbf{X}_1) \cap \mathbb{D}_{t+1,t+1}(\mathbf{X}_{t+2}) \neq \emptyset$. Then, there exists $\mathbf{X}_1^{(1)}, \mathbf{X}_{t+1}^{(1)}$ resulting from a $(1,1)$-criss-cross deletion of $\mathbf{X}_1$ and $\mathbf{X}_{t+2}$, respectively, such that $\mathbb{D}_{t,t}(\mathbf{X}_1^{(1)}) \cap \mathbb{D}_{t,t}(\mathbf{X}_{t+1}^{(1)}) \neq \emptyset$. Thus, according to the induction hypothesis, there exist $t-2$ arrays $\mathbf{X}_1^{(1)}, \ldots, \mathbf{X}_t^{(1)}$ that satisfy $\mathbb{D}_{1,1}(\mathbf{X}_i^{(1)}) \cap \mathbb{D}_{1,1}(\mathbf{X}_{i+1}^{(1)}) \neq \emptyset$ for all $1 \leq i \leq t$.

According to Theorem 5.1, there exist t arrays $\mathbf{X}_2, \ldots, \mathbf{X}_{t+1}$ such that for all $2 \leq i \leq t+1$, $\mathbf{X}_i \in \mathbb{I}(\mathbf{X}_{i-1}^{(1)}) \cap \mathbb{I}(\mathbf{X}_i^{(1)})$. Therefore, it holds that for $1 \leq 1 \leq t+1$,

$$\mathbf{X}_i^{(1)} \in \mathbb{D}_{1,1}(\mathbf{X}_i) \cap \mathbb{D}_{1,1}(\mathbf{X}_{i+1}).$$

This completes the "if" part of the proof. □

Next, we show a similar claim for the insertion case.

Claim 5.2. *For any two arrays $\mathbf{X}_1, \mathbf{X}_{t+1} \in \Sigma_q^{n\times n}$, $\mathbb{I}_{t,t}(\mathbf{X}_1) \cap \mathbb{I}_{t,t}(\mathbf{X}_{t+1}) \neq \emptyset$ if and only if there exist $t-1$ arrays $\mathbf{X}_2, \ldots, \mathbf{X}_t$ such that $\mathbb{I}_{1,1}(\mathbf{X}_i) \cap \mathbb{I}_{1,1}(\mathbf{X}_{i+1}) \neq \emptyset$ for all $1 \leq i \leq t$.*

Proof sketch. The proof follows a similar chain of arguments as in the proof of Claim 5.1 for the deletion case by interchanging arguments for deletions to insertions, and vice versa. Thus, we omit the proof. □

Now we can prove Corollary 5.1 using Claims 5.1 and 5.2.

Proof of Corollary 5.1. The proof follows by showing that for any $\mathbf{X}_1, \mathbf{X}_{t+1} \in \Sigma_q^{n\times n}$, it holds that

$$\mathbb{D}_{t,t}(\mathbf{X}_1) \cap \mathbb{D}_{t,t}(\mathbf{X}_{t+1}) \neq \emptyset \iff \mathbb{I}_{t,t}(\mathbf{X}_1) \cap \mathbb{I}_{t,t}(\mathbf{X}_{t+1}) \neq \emptyset\,.$$

For any $\mathbf{X}_1, \mathbf{X}_{t+1} \in \Sigma_q^{n\times n}$, if $\mathbb{D}_{t,t}(\mathbf{X}_1) \cap \mathbb{D}_{t,t}(\mathbf{X}_{t+1}) \neq \emptyset$, then from Claim 5.1 we know that there exist $t-1$ arrays $\mathbf{X}_2, \ldots, \mathbf{X}_t$ such that $\mathbb{D}_{1,1}(\mathbf{X}_i) \cap \mathbb{D}_{1,1}(\mathbf{X}_{i+1}) \neq \emptyset$ for all $1 \leq i \leq t$. Then, according to Theorem 5.1, there exist t arrays $\mathbf{X}_1^{(1)}, \ldots, \mathbf{X}_t^{(1)}$ such that for all $1 \leq i \leq t$,

$$\mathbf{X}_i^{(1)} \in \mathbb{I}_{1,1}(\mathbf{X}_i) \cap \mathbb{I}_{1,1}(\mathbf{X}_{i+1}).$$

Finally, we can apply Claim 5.2 to conclude that $\mathbb{I}_{t,t}(\mathbf{X}_1) \cap \mathbb{I}_{t,t}(\mathbf{X}_{t+1}) \neq \emptyset$. The "only if" part follows similarly. □

5.3.2 Any Combinations of Row and Column Deletion and Insertion Equivalence

We show the central equivalence between t-criss-cross deletion correcting codes and t-criss-cross insertion correcting codes (see Theorem 5.2). The proof of Theorem 5.2 follows by first showing that the equivalence holds for all $(t_\mathrm{r}, t_\mathrm{r}+c)$-criss-cross insertion/deletion correcting codes, where c is a positive integer. Then, by symmetry, the equivalence holds for $(t_\mathrm{c}+c, t_\mathrm{c})$-criss-cross insertion/deletion correcting codes, which completes the proof. The proof utilizes the results of the previous subsection (Corollary 5.1 and Lemmas 5.1 and 5.2).

Theorem 5.2. *A code $\mathcal{C}_t^{\mathrm{cc}} \subseteq \Sigma_q^{n\times n}$ is a t-criss-cross deletion correcting code if and only if $\mathcal{C}_t^{\mathrm{cc}}$ is a t-criss-cross insertion correcting code.*

Proof. The goal is to show that for any $\mathbf{X}, \mathbf{Y} \in \Sigma_q^{n\times n}$ and any two non-negative integers t_r and t_c such that $t_\mathrm{r} + t_\mathrm{c} = t$, it holds that

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{Y}) = \emptyset \iff \mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{Y}) = \emptyset\,.$$

We only show the equivalence for $(t_\mathrm{r}, t_\mathrm{r}+c)$-criss-cross insertion/deletion correcting codes, i.e., for any $\mathbf{X}, \mathbf{Y} \in \Sigma_q^{n\times n}$ and any non-negative integers t_r and c such that $2t_\mathrm{r} + c = t$ we have

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) = \emptyset \iff \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) = \emptyset\,.$$

Assuming that the equivalence holds for $(t_\mathrm{r}, t_\mathrm{r}+c)$-criss-cross insertion/deletion correcting codes, then the following statement implies that the equivalence also holds for $(t_\mathrm{r}+c, t_\mathrm{r})$-

criss-cross insertion/deletion correcting codes which completes the proof.

$$\begin{aligned}\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) = \emptyset &\overset{\text{(a)}}{\iff} \mathbb{D}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{X}^T) \cap \mathbb{D}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{Y}^T) = \emptyset\,,\\ &\overset{\text{(b)}}{\iff} \mathbb{I}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{X}^T) \cap \mathbb{I}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{Y}^T) = \emptyset\,,\\ &\overset{\text{(c)}}{\iff} \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) = \emptyset\,.\end{aligned}$$

Steps (a) and (c) follow trivially by examining the transpose of the arrays $\mathbf{X}$ and $\mathbf{Y}$. Step (b) follows from the assumed equivalence.

The proof of equivalence for $(t_\mathrm{r}, t_\mathrm{r}+c)$-criss-cross insertion/deletion correcting codes proceeds as well by contraposition, i.e., we show that

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) \neq \emptyset \iff \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) \neq \emptyset\,.$$

In what follows, we only show the "only if" parts since the "if" parts follow similarly. We prove the equivalence for $(t_\mathrm{r}, t_\mathrm{r}+c)$-criss-cross insertion/deletion correcting codes by induction over $c = t_\mathrm{c} - t_\mathrm{r}$.

Base case ($c = 1$)*:* For the reader's convenience, a flowchart of this proof step is presented in Figure 5.3.

Assume that there exists an array $\mathbf{E} \in \Sigma_q^{(n-t_\mathrm{r})\times(n-t_\mathrm{r}-1)}$ such that $\mathbf{E} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{Y})$. Let $\ell = 2t_\mathrm{r}$, we define the series of arrays $\{\mathbf{C}_s\}_{s=1}^{\ell}$ to be the intermediate arrays obtained by deleting a row or a column from $\mathbf{Y}$ to reach $\mathbf{E}$. For notational convenience we let $\mathbf{C}_0 \triangleq \mathbf{Y}$ and $\mathbf{C}_{\ell+1} \triangleq \mathbf{E}$. Moreover, we follow an alternating order of deletion between columns and rows,[3] i.e.,

$$\mathbf{C}_s \in \begin{cases}\mathbb{D}_{0,1}(\mathbf{C}_{s-1}) & \text{if } s \text{ is odd},\\ \mathbb{D}_{1,0}(\mathbf{C}_{s-1}) & \text{otherwise.}\end{cases}$$

Furthermore, we have $\mathbf{C}_{\ell+1} \in \mathbb{D}_{0,1}(\mathbf{C}_\ell)$. We denote by $\mathbf{B}_\ell \in \Sigma_q^{(n-t_\mathrm{r})\times(n-t_\mathrm{r})}$ the array resulting from deleting t_r columns and t_r rows from $\mathbf{X}$ such that $\mathbf{E} \in \mathbb{D}_{0,1}(\mathbf{B}_\ell)$.

We now want to show that there exists a series of arrays $\{\mathbf{B}_s\}_{s=0}^{\ell-1}$ such that

$$\mathbf{B}_s \in \begin{cases}\mathbb{I}_{0,1}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}) & \text{if } s \text{ is odd},\\ \mathbb{I}_{1,0}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}) & \text{otherwise.}\end{cases}$$

By definition, ℓ is even and $\mathbf{C}_{\ell+1} \in \mathbb{D}_{0,1}(\mathbf{B}_\ell) \cap \mathbb{D}_{0,1}(\mathbf{C}_\ell)$. By Lemma 5.1 there exists an array $\mathbf{B}_{\ell-1} \in \Sigma_q^{(n-t_\mathrm{r})\times(n-t_\mathrm{r}+1)}$ such that $\mathbf{B}_{\ell-1} \in \mathbb{I}_{0,1}(\mathbf{B}_\ell) \cap \mathbb{I}_{0,1}(\mathbf{C}_\ell)$. Moreover, we know that there exists $\mathbf{C}_{\ell-1} \in \Sigma_q^{(n-t_\mathrm{r}+1)\times(n-t_\mathrm{r})}$ such that $\mathbf{C}_\ell \in \mathbb{D}_{1,0}(\mathbf{C}_{\ell-1})$. Consequently, we have that $\mathbf{C}_\ell \in \mathbb{D}_{0,1}(\mathbf{B}_{\ell-1}) \cap \mathbb{D}_{1,0}(\mathbf{C}_{\ell-1})$. By Lemma 5.2 there exists a $\mathbf{B}_{\ell-2} \in \Sigma_q^{(n-t_\mathrm{r}+1)\times(n-t_\mathrm{r}+1)}$ such that $\mathbf{B}_{\ell-2} \in \mathbb{I}_{1,0}(\mathbf{B}_{\ell-1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{\ell-1})$. Following the same arguments as above, one can show that for all even values of $s \in \{1, \ldots, \ell\}$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{0,1}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 5.1 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{0,1}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$.

[3] The proof works for any order of row/column deletions by arranging the chain of arguments in the proof according to the order.

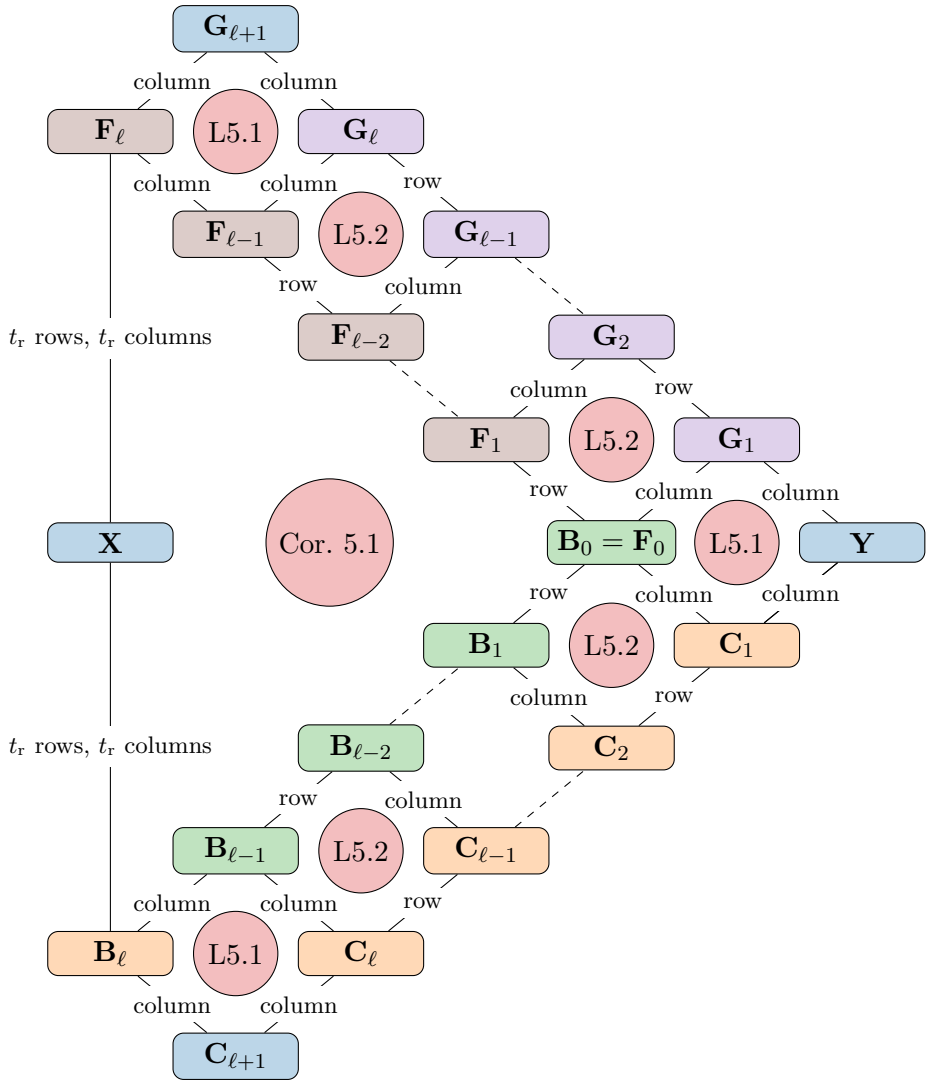


Figure 5.3: A flowchart of the proof of Theorem 5.2. Given an array $\mathbf{C}_{\ell+1} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{Y})$, we show that there exists an array $\mathbf{G}_{\ell+1} \in \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{Y})$. The series of orange marked arrays are given by the first assumption. We can prove the existence of the series of green marked arrays by Lemmas 5.1 and 5.2. The brown marked arrays are then given by applying Corollary 5.1. Lastly, the existence of the purple arrays can be shown again by Lemmas 5.1 and 5.2.

Similarly, for all odd values of $s \in \{1, \ldots, \ell\}$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{1,0}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Hence, by Lemma 5.2 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{1,0}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$. Subsequently, we will end up with an array $\mathbf{B}_0 \in \Sigma_q^{n \times n}$ such that $\mathbf{B}_\ell \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}}(\mathbf{B}_0)$. Therefore, by Corollary 5.1 there exists an array $\mathbf{F}_\ell \in \Sigma_q^{(n+t_\mathrm{r}) \times (n+t_\mathrm{r})}$ such that $\mathbf{F}_\ell \in \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}}(\mathbf{B}_0)$.

Let $\mathbf{F}_0 \triangleq \mathbf{B}_0$ and $\mathbf{F}_{-1} \triangleq \mathbf{C}_1$, we denote again the series of arrays $\{\mathbf{F}_s\}_{s=1}^{\ell}$ that are the

intermediate arrays obtained by a row or a column insertion starting from $\mathbf{F}_0$ until reaching $\mathbf{F}_\ell$. For convenience, we again consider alternating insertions of rows and columns, i.e.,

$$\mathbf{F}_s \in \begin{cases} \mathbb{I}_{1,0}(\mathbf{F}_{s-1}) & \text{if } s \text{ is odd,} \\ \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{otherwise.} \end{cases}$$

Define the array $\mathbf{G}_0 \triangleq \mathbf{Y}$, we will now show the existence of the series of arrays $\{\mathbf{G}_s\}_{s=1}^{\ell+1}$ such that

$$\mathbf{G}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{if } s \text{ is odd,} \\ \mathbb{I}_{1,0}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{otherwise.} \end{cases}$$

Following the same arguments used to construct the series $\{\mathbf{B}_s\}_{s=0}^{\ell-1}$, one can show that for all even values of $s \in \{0, \ldots, \ell\}$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{0,1}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 5.1 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{0,1}(\mathbf{F}_s) \cap \mathbb{I}_{0,1}(\mathbf{G}_s)$. Similarly, for all odd values of $s \in \{0, \ldots, \ell\}$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{1,0}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Hence, by Lemma 5.2 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{1,0}(\mathbf{G}_s) \cap \mathbb{I}_{0,1}(\mathbf{F}_s)$. As a consequence, we have shown that if there exists an array $\mathbf{C}_{\ell+1} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{Y})$, then there exists an array $\mathbf{G}_{\ell+1} \in \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{Y})$. This concludes the base case.

Induction hypothesis: For any integer $c \geq 1$ and for any arrays $\mathbf{X}, \mathbf{Y} \in \Sigma_q^{n \times n}$, it holds that

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) \neq \emptyset \iff \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{Y}) \neq \emptyset \,.$$

Induction step: Assume that the induction hypothesis holds for all values $0 \leq t_\mathrm{c} - t_\mathrm{r} \leq c$. We prove that the hypothesis holds for $t_\mathrm{c} - t_\mathrm{r} = c + 1$.

Assume that there exists an array $\mathbf{E} \in \Sigma_q^{(n-t_\mathrm{r}) \times (n-t_\mathrm{r}-c-1)}$ such that $\mathbf{E} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c+1}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c+1}(\mathbf{Y})$. Let $\ell = 2t_\mathrm{r}$, define $\mathbf{C}_0 \triangleq \mathbf{Y}$ and $\mathbf{C}_{\ell+c+1} \triangleq \mathbf{E}$. We denote by $\{\mathbf{C}_s\}_{s=0}^{\ell+c+1}$ the series of arrays resulting from a deletion of a row or a column starting from $\mathbf{Y}$ until obtaining $\mathbf{E}$ as follows

$$\mathbf{C}_s \in \begin{cases} \mathbb{D}_{0,1}(\mathbf{C}_{s-1}) & \text{if } s \text{ is odd or } s > \ell, \\ \mathbb{D}_{1,0}(\mathbf{C}_{s-1}) & \text{otherwise.} \end{cases}$$

We denote by $\mathbf{B}_{\ell+c} \in \Sigma_q^{(n-t_\mathrm{r}) \times (n-t_\mathrm{r}-c)}$ the array resulting from deleting t_r columns and $t_\mathrm{r}+c$ rows from $\mathbf{X}$ such that $\mathbf{E} \in \mathbb{D}_{0,1}(\mathbf{B}_{\ell+c})$. We now want to show that there exists a series of arrays $\{\mathbf{B}_s\}_{s=0}^{\ell+c-1}$ such that

$$\mathbf{B}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}) & \text{if } s \text{ is odd or } s \geq \ell, \\ \mathbb{I}_{1,0}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}) & \text{otherwise.} \end{cases}$$

Following the same arguments as in the base case, one can show that for all even values of $s \in \{0, \ldots, \ell\}$ and for all values of $\ell \leq s \leq \ell + c$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{0,1}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 5.1 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{0,1}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$. Similarly, for all odd values of $s \in \{0, \ldots, \ell\}$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{1,0}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 5.2 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{1,0}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$. Subsequently, we will end up with an array $\mathbf{B}_0 \in \Sigma_q^{n \times n}$ such that $\mathbf{B}_{\ell+c} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{B}_0)$. Therefore, by using the induction hypothesis, we can show the existence of the

array $\mathbf{F}_{\ell+c} \in \Sigma_q^{(n+t_\mathrm{r})\times(n+t_\mathrm{r}+c)}$ such that $\mathbf{F}_{\ell+c} \in \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{B}_0)$.

Let $\mathbf{F}_0 \triangleq \mathbf{B}_0$ and $\mathbf{F}_{-1} \triangleq \mathbf{C}_1$, we denote again the series of arrays $\{\mathbf{F}_s\}_{s=1}^{\ell+c}$ as the intermediate arrays obtained by a row or a column insertion starting from $\mathbf{F}_0$ until reaching $\mathbf{F}_\ell$ as follows

$$\mathbf{F}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{if } s \text{ is even or } s > \ell\,, \\ \mathbb{I}_{1,0}(\mathbf{F}_{s-1}) & \text{otherwise}\,. \end{cases}$$

Define the array $\mathbf{G}_0 \triangleq \mathbf{Y}$, we will now show the existence of the series of arrays $\{\mathbf{G}_s\}_{s=1}^{\ell+c+1}$ such that

$$\mathbf{G}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{if } s \text{ is odd or } s > \ell\,, \\ \mathbb{I}_{1,0}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{otherwise}\,. \end{cases}$$

Following the same arguments as in the base case, one can show that for all even values of $s \in \{0,\ldots,\ell\}$ and $\ell \leq s \leq \ell + c$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{0,1}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 5.1 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{0,1}(\mathbf{F}_s) \cap \mathbb{I}_{0,1}(\mathbf{G}_s)$. Similarly, for all odd values of $s \in \{0,\ldots,\ell\}$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{1,0}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 5.2 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{1,0}(\mathbf{G}_s) \cap \mathbb{I}_{0,1}(\mathbf{F}_s)$. As a consequence, we have shown that if there exists an array $\mathbf{C}_{\ell+c+1} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c+1}(\mathbf{X}) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c+1}(\mathbf{Y})$, then there exists an array $\mathbf{G}_{\ell+c+1} \in \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c+1}(\mathbf{X}) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c+1}(\mathbf{Y})$. This concludes the induction. □

On a final note, we want to emphasize that the number of row and column operations has to stay the same for the equivalences to hold. We illustrate this by giving the following counterexample.

Counter Example 5.1. *The equivalence of a $(1,0)$-criss-cross deletion correcting code and a $(0,1)$-criss-cross deletion correcting code does not hold. To show this, we consider two arrays $\mathbf{X},\mathbf{Y} \in \Sigma_2^{3\times 3}$ and assume there exists an array $\mathbf{D} \in \Sigma_2^{2\times 3}$ such that $\mathbf{D} \in \mathbb{D}_{1,0}^2(\mathbf{X}) \cap \mathbb{D}_{1,0}^2(\mathbf{Y})$ as follows.*

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}\,,\quad \mathbf{Y} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\,,\quad \mathbf{D} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}\,,$$

where $\mathbf{D}$ is obtained by deleting the second column from $\mathbf{X}$ and $\mathbf{Y}$. Since more than one row of $\mathbf{X}$ and $\mathbf{Y}$ are different, we see that $\mathbb{D}_{0,1}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y}) = \emptyset$ and therefore the equivalence does not hold.

5.4 Upper Bound on the Cardinality of Codes

This section presents an asymptotic upper bound on the cardinality of any t-criss-cross insertion/deletion correcting code. It implies an asymptotic lower bound on the redundancy of any binary t-criss-cross insertion/deletion correcting code, denoted by $r^{\mathrm{cc}}(n,t)$.

Lemma 5.3. *Any upper bound on the cardinality of a q-ary t-deletion-correcting code $\mathcal{C}_{q,n,t}^{\mathrm{vec}}$ with $q = 2^n$ is also an upper bound on the cardinality of a binary t-criss-cross insertion/deletion correcting code.*

Proof. Note that a 2^n-ary t-deletion correcting code $\mathcal{C}^{\text{vec}}_{2^n,n,t}$ can be seen also as a binary t column deletion correcting code by interpreting the symbols as binary columns. Since a t-criss-cross insertion/deletion correcting code $\mathcal{C}^{\text{cc}}_t$ can correct any combination of t_{r} row and t_{c} column deletions such that $t_{\text{r}} + t_{\text{c}} = t$, in particular it can also correct any t column deletions. Therefore, any upper bound on the size of $\mathcal{C}^{\text{vec}}_{2^n,n,t}$ is also a valid upper bound on the size of $\mathcal{C}^{\text{cc}}_t$. □

Corollary 5.2. *For any binary t-criss-cross insertion/deletion correcting code $\mathcal{C}^{\text{cc}}_t$ it holds that*

$$|\mathcal{C}^{\text{cc}}_t| \lesssim \frac{t!2^{n^2}}{(2^n-1)^t n^t}\,.$$

Consequently, we have $r^{\text{cc}}(n,t) \gtrsim tn + t\log_2(n) - \log_2(t!)$.

Proof. From Theorem 3.2 ([KK13]), we have for a q-ary t-deletion correcting code that $|\mathcal{C}^{\text{vec}}_{q,n,t}| \lesssim \frac{t!q^n}{(q-1)^t n^t}$ when q is fixed. Following the same arguments as in the proof of [KK13, Theorem 4.3], we can show that this bound also holds when $q = 2^n$. Therefore, for any binary t-criss-cross insertion/deletion correcting code $\mathcal{C}^{\text{cc}}_t$ it holds by Lemma 5.3 that

$$|\mathcal{C}^{\text{cc}}_t| \leq |\mathcal{C}^{\text{vec}}_{2^n,n,t}| \lesssim \frac{t!2^{n^2}}{(2^n-1)^t n^t}\,.$$

Therefore, we have

$$r^{\text{cc}}(n,t) \gtrsim n^2 - \log_2(|\mathcal{C}^{\text{cc}}_t|) \approx tn + t\log_2(n) - \log_2(t!)\,.$$

□

5.5 Simplified Error Model Construction

Before providing a code construction for the general t-criss-cross error model, we present in this section a code construction for a simplified error model of the 2-criss-cross error. In this error, we fix the error as a $(1,1)$-criss-cross insertion/deletion error, i.e., exactly one row and one column deletion or insertion. This contrasts the general error model assumed within this chapter but assists in understanding the techniques. First, we explain our construction and subsequently prove it by providing an explicit decoder for a $(1,1)$-criss-cross deletion or insertion.

The code $\mathcal{C}^{\text{cc}}_{(1,1)}$, which will be formally defined in Construction 5.1, is an existential code whose codewords are $n \times n$ binary arrays structured as shown in Figure 5.4 and as explained next. The code consists of two main components. The columns and the rows are indexed using the binary expansion $\mathbf{U}$ and $\mathbf{V}$ of two n-ary VT coded vectors (see Construction 3.2) to recover the positions of the inserted/deleted row and column.[4] The parity bits $\mathbf{p}_c$ and $\mathbf{p}_r$ are used to recover the deleted information in case of a deletion and to help detect the location of an inserted column or row in case of an insertion.

[4] We assume that n is a power of 2 so that $\log_2(n)$ is an integer, while the extension for other values of n will be apparent from the context.

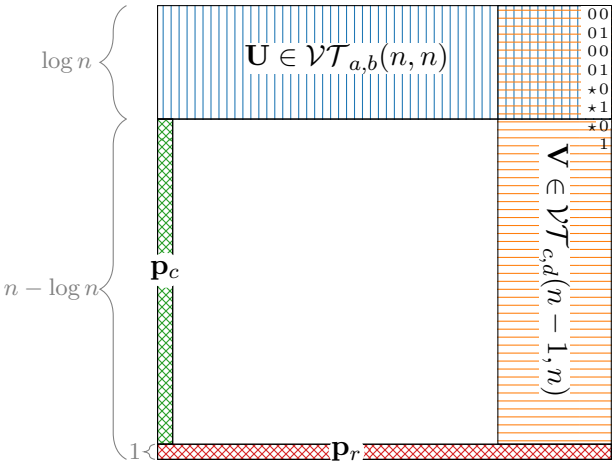


Figure 5.4: The structure of the codewords of the $(1,1)$-criss-cross insertion/deletion-correcting code $\mathcal{C}^{\text{cc}}_{(1,1)}$ presented in Construction 5.1. $\mathbf{U}$ is the binary representation of a q-ary vector $\mathbf{u} \in \mathcal{VT}_{a,b}(n,q)$ with $q = n$. Each column is viewed as a symbol of the VT-coded vector $\mathbf{u}$. The last column of $\mathbf{U}$ is an alternating sequence, and the second to last column must start with four consecutive zeros. $\mathbf{V}$ is defined similarly to $\mathbf{U}$ where each row is a symbol of a VT coded vector $\mathbf{v} \in \mathcal{VT}_{c,d}(n-1,n)$. The alternating sequence of $\mathbf{U}$ is extended by one bit in $\mathbf{V}$. For the encoding of $\mathbf{V}$, we replace the alternating sequence with the all-zero sequence. The column $\mathbf{p}_c$ is a parity column consisting of the sum of all column entries in the respective row. The row $\mathbf{p}_r$ is a parity row defined similarly to $\mathbf{p}_c$. We denote by $\mathbf{X} \in \mathcal{VT}_{a,b}(n,q)$ the binary representation of a q-ary vector $\mathbf{x} \in \Sigma_q^n$, such that $\mathbf{x} \in \mathcal{VT}_{a,b}(n,q)$.

Indexing the columns: The first $\log_2(n)$ rows of a codeword $\mathbf{C} \in \mathcal{C}^{\text{cc}}_{(1,1)}$ are the binary representation of a q-ary vector $\mathbf{u}$ encoded using a VT code $\mathcal{VT}_{a,b}(n,q)$ that can correct one insdel error, where $q = n$. The $\log_2(n) \times n$ binary array $\mathbf{U}$ satisfies the following requirements:

1. Every column of $\mathbf{U}$ is the binary representation of a symbol of the VT coded vector $\mathbf{u} \in \mathcal{VT}_{a,b}(n,n)$;
2. Any two consecutive columns are different;
3. The last column is the alternating sequence that starts with 0;
4. The first 4 bits of the second to last column are zeros.

As we shall see in the decoding proof, this array serves as an index of the columns. That is, it allows the decoder to recover the exact position of the inserted/deleted column.

Indexing the rows: The $(n-1) \times \log_2(n)$ array formed of the last $\log_2(n)$ bits of rows 1 to $n-1$ (situated at the right of the array $\mathbf{C} \in \mathcal{C}^{\text{cc}}_{(1,1)}$) is the binary representation of a q-ary vector $\mathbf{v}$ encoded using a VT code $\mathcal{VT}_{c,d}(n-1,q)$ that can correct one insdel, with $q = n$. The $(n-1) \times \log_2(n)$ binary array $\mathbf{V}$ satisfies the following requirements:

1. Each row of $\mathbf{V}$ is the binary representation of a symbol of the VT coded vector $\mathbf{v} \in \mathcal{VT}_{c,d}(n-1,n)$;
2. Any two consecutive rows are different;

3. The first $\log_2(n)$ rows also satisfy the requirements imposed on $\mathbf{U}$, except for replacing the alternating sequence by the all-zero sequence;[5]
4. The first bit below the alternating sequence is the opposite of the last bit of the alternating sequence. In other words, the alternating sequence is of length $\log_2(n)+1$. Again, here we assume that the stored bit belongs to the alternating sequence, but for the encoding of $\mathbf{v}$, we assume that the first $\log_2(n)+1$ bits of the last column are all zeros.

This array serves as an index of the rows that allows the decoder to recover the position of the inserted/deleted row.

Parities: The part of the first column of $\mathbf{C} \in \mathcal{C}^{\text{cc}}_{(1,1)}$ that is not included in $\mathbf{U}$ is a parity of the same part of all corresponding columns, i.e., each entry of that column is the sum of all bits corresponding to its same row. This column is denoted by $\mathbf{p}_c$ and is shown on the left in Figure 5.4. Moreover, the last row of $\mathbf{C}$ is a parity of all the rows and is denoted by $\mathbf{p}_r$. In case of deletion, the parities allow the decoder to recover the information in the deleted column and row. In case of an insertion, the parities help the decoder to exactly recover the index of the inserted row and column in case the arrays $\mathbf{U}$ and $\mathbf{V}$ fail to do so, as explained in more details in the next section. We start with an example to illustrate the idea before going into the formal definition of the construction. The example also demonstrates the deletion decoder.

Example 5.1. *We construct a* 9×9 *codeword of our code to illustrate the construction and the decoding algorithm. Note that we choose* n *to be* 9 *(not a multiple of* 2*) for convenience and to simplify the example. We explain in detail how the following codeword* $\mathbf{X}$ *is constructed.*

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.2}$$

The columns and the rows are indexed by codewords of q*-ary VT codes with* $q = 2^{(\lceil \log_2(n) \rceil)} = 16$ *with* $a = 2$*,* $b = 0$*,* $c = 7$ *and* $d = 0$*. The first* 4 *rows of the codeword should then be the binary representation of a* q*-ary vector* $\mathbf{u} \in \mathcal{VT}_{2,0}(9, 16)$*. For clarity of presentation, we represent a symbol* $\mathbf{x} = (x_1, x_2, x_3, x_4)^T \in \Sigma_{2^4}$ *as the decimal representation* $x = \sum_{i=1}^{4} x_i 2^{i-1}$*. Our construction requires the last symbol of* $\mathbf{u}$ *to be* 10*, i.e., its binary representation is the alternating sequence. In addition, the second to last symbol of* $\mathbf{u}$ *must be* 0*. Moreover, every two consecutive symbols of* $\mathbf{u}$ *must be different. An example is* $\mathbf{u} = (0, 1, 2, 3, 4, 5, 11, 0, 10) \in$

[5] The array $\mathbf{U}$ can still store the alternating sequence. When checking the constraints on the rows of $\mathbf{V}$, we assume the last column of $\mathbf{U}$ is the all-zero sequence. Similarly, when encoding $\mathbf{V}$, we also assume that the last column is all-zero. Since the decoder knows this information, the all-zero sequence does not need to be stored in the array.

$\Sigma_{2^4}^9$. *The binary representation* $\mathbf{U}$ *of* $\mathbf{u}$ *is the first four rows of* $\mathbf{X}$ *given in Equation* (5.2) *(marked in blue). Given* $\mathbf{u}$*, we index the columns with a vector* $\mathbf{v} \in \mathcal{VT}_{7,0}(8, 16)$ *such that the first four symbols of* $\mathbf{v}$ *are predetermined to be* $3, 10, 1$ *and* 10*, respectively. Our construction requires the last bit of the binary representation of the fifth symbol of* $\mathbf{v}$ *to be set to* 0 *as an extension of the alternating sequence of the last symbol of* $\mathbf{u}$ *(see Equation* (5.2)*). Similarly to* $\mathbf{u}$*, any two consecutive symbols of* $\mathbf{v}$ *must differ. An example is* $\mathbf{v} = (3, 10, 1, 10, 6, 8, 9, 7) \in \Sigma_{2^4}^8$*. The binary representation* $\mathbf{V}$ *of* $\mathbf{v}$ *is the transpose of the last four columns and first eight rows of the array* $\mathbf{X}$ *given in Equation* (5.2) *(marked in orange). The remaining entries of the array* $\mathbf{X}$ *not belonging to the first column nor the last row (marked in black) are arbitrary. The entries of the last row (marked in red) are the column-wise parity bits. The remaining entries of the first column (marked in green) are the row-wise parity bits.*

To illustrate the decoding strategy, assume that row 2 *and column* 7 *of* $\mathbf{X}$ *are deleted. Thus, the following array* $\mathbf{X}^{2,7}$ *is obtained.*

$$\mathbf{X}^{2,7} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{5.3}$$

The decoder looks at the last non-deleted column and knows it should be the alternating sequence because it is not the all-zero column in the first four rows. From the last column, the decoder knows that the second row is deleted. Using the row-wise parity bits of the last row, the decoder can recover the second row. Now, the decoder has all the rows of $\mathbf{U}$ *with one deleted column and can thus use the VT decoder to retrieve the value and position of the lost column. Note that since every two consecutive columns in* $\mathbf{U}$ *are different, the decoder recovers the exact location of the deleted column. Using the deleted column's index, the decoder recovers the values of the bits outside of* $\mathbf{U}$ *(rows* 5 *to* 8*) using the column-wise parity bits. The last bit of the deleted column is the sum of all other bits.*

Formally, the presented code $\mathcal{C}_{(1,1)}^{\mathrm{cc}}$ can be seen as an intersection of four codes over $\Sigma_2^{n \times n}$ that define the constraints imposed on the codewords of $\mathcal{C}_{(1,1)}^{\mathrm{cc}}$. Let $\ell \triangleq \log_2(n)$, we define $\mathbf{W}$ to be the all-zero array except for the first $\ell + 1$ bits of the last column to be the alternating sequence, i.e., $\mathbf{W}_{[\ell+1],n} = (0, 1, 0, 1, \ldots)^T$ and $\mathbf{W}_{i,j} = 0$ otherwise. We denote by $\mathbf{X} \in \mathcal{VT}_{a,b}(n, q)$ the binary representation of a q-ary vector $\mathbf{x} \in \Sigma_q^n$, such that $\mathbf{x} \in \mathcal{VT}_{a,b}(n, q)$.

In detail, we define the following sets:

$$\mathcal{U}(a,b) \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad j \in [n-1] \\ \mathbf{X}_{[4],n-1} = (0,0,0,0)^T, \\ \mathbf{X}_{[\ell+1],n} = (0,1,0,1,\ldots)^T, \\ \mathbf{X}_{[\ell],[n]} \in \mathcal{VT}_{a,b}(n,2^\ell) \end{array} \right\},$$

$$\mathcal{V}(c,d) \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{X}_{i,[n-\ell+1,n]} \neq \mathbf{X}_{i+1,[n-\ell+1,n]}, \quad i \in [n-1] \\ \mathbf{X}_{[\ell+1],n} = (0,0,0,\ldots)^T, \\ \mathbf{X}^T_{[n-\ell+1,n],n-1} \in \mathcal{VT}_{c,d}(n-1,2^\ell) \end{array} \right\},$$

$$\mathcal{V}'(c,d) \triangleq \left\{ \mathbf{Y} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{Y}_{[\ell+1],n} = (0,1,0,1,\ldots)^T, \\ \mathbf{Y} \oplus \mathbf{W} \in \mathcal{V}(c,d) \end{array} \right\},$$

$$\mathcal{P}_c \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \mathbf{X}_{i,1} = \sum_{j=2}^{n} \mathbf{X}_{i,j} \bmod 2, \quad i = \ell+1,\ldots,n-1 \right\},$$

$$\mathcal{P}_r \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \mathbf{X}_{n,j} = \sum_{i=1}^{n-1} \mathbf{X}_{i,j} \bmod 2, \quad j \in [n] \right\}.$$

Construction 5.1. *For non-negative integers a,b,c,d such that $0 \leq a,b,d \leq n-1$ and $0 \leq c \leq n-2$, the code $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ is the set of arrays $\mathbf{C} \in \Sigma_2^{n\times n}$ that belong to*

$$\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d) \triangleq \mathcal{U}(a,b) \cap \mathcal{V}'(c,d) \cap \mathcal{P}_c \cap \mathcal{P}_r.$$

The main statement of this subsection is summarized in the following theorem and corollary.

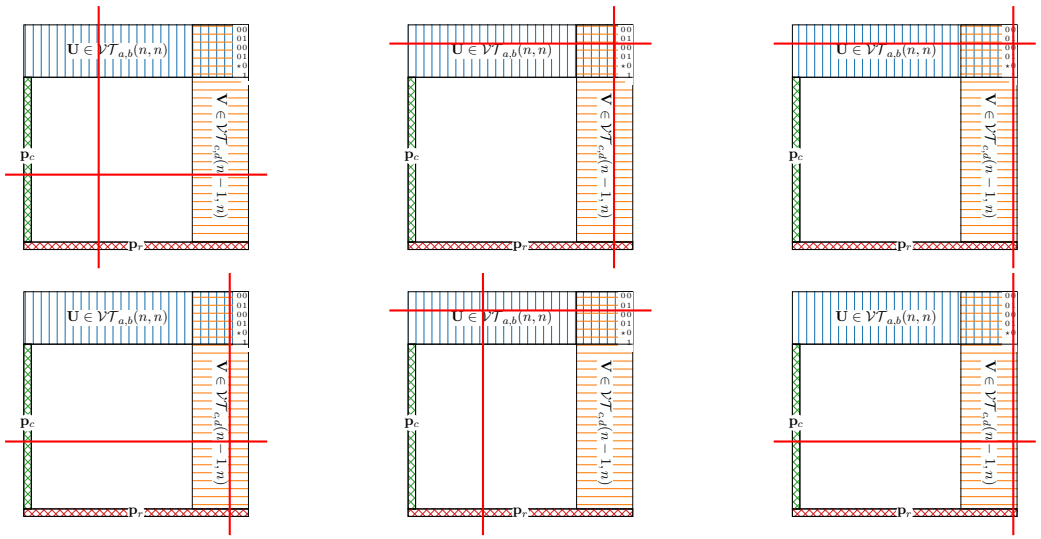
Theorem 5.3. *The code $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ (defined in Construction 5.1) is a $(1,1)$-criss-cross deletion and insertion correcting code with an explicit decoder.*

Corollary 5.3. *There exist non-negative integers a,b,c,d for which the redundancy of the code $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ (defined in Construction 5.1) is at most*

$$2n + 4\log_2(n) + 7 + 2\log_2(e)\,.$$

In the following, we show how the code construction leverages the structure of a codeword $\mathbf{C} \in \mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ to correct a $(1,1)$-criss-cross deletion or insertion. Formally, we prove Theorem 5.3. We assume that the decoder knows the dimension of the received array. In other words, the decoder knows whether a $(1,1)$-criss-cross deletion or a $(1,1)$-criss-cross insertion has happened and only needs to correct it.

Intuition: The goal of the decoder is to use the insertion/deletion correction capability of $\mathcal{VT}_{a,b}(n,n)$ and $\mathcal{VT}_{c,d}(n-1,n)$ to recover the positions of the inserted/deleted column and row. The decoder first uses the alternating sequence to check if a row of $\mathbf{U}$ is inserted/deleted and, therefore, corrects it before proceeding to the VT code decoder. In case of a deletion, the second to last column allows the decoder to detect whether the alternating sequence



(a) Case 1–(i) (b) Case 1–(ii) (c) Case 2

Figure 5.5: The distinct $(1,1)$-criss-cross deletion patterns for the decoder to distinguish.

was deleted. If the alternating sequence is deleted, the decoder cannot use $\mathbf{U}$ and has to start using $\mathbf{V}$ to detect the position of the deleted row, recover it using the parities, and then obtain the position of the deleted column from $\mathbf{U}$. The parities are used to retrieve the deleted information once the decoder has the position of the deleted row and/or column. In case of an insertion, the inserted vector may equal another consecutive vector in either $\mathbf{U}$ or $\mathbf{V}$. In this case, the decoder uses the parity bits over the remaining part of the vector to exactly recover the position of the inserted row or column. We are now ready to present the proof of Theorem 5.3.

Proof of Theorem 5.3. We split the proof into two parts according to the two error behaviors: *a)* an explicit decoder for a $(1,1)$-criss-cross deletion; and *b)* an explicit decoder for an $(1,1)$-criss-cross insertion.

a) Deletion correcting decoder: The decoder for $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ receives as input a $(n-1)\times(n-1)$ array $\widetilde{\mathbf{C}}$ resulting from a $(1,1)$-criss-cross deletion in an array $\mathbf{C}$ of $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ and works as follows. The decoder looks at the first $\ell \times (n-1)$ subarray of $\widetilde{\mathbf{C}}$ and examines the last column. For the reader's convenience, we depict the distinct deletion patterns for the cases mentioned below in Figure 5.5.

Case 1: Assume the last column of $\mathbf{C}$ is not deleted. Using the alternating sequence, the decoder can detect whether or not there was a row deletion in $\mathbf{U}$ and locate its index. This is done by locating a run of length two in the alternating sequence. The last bit of the alternating sequence falling in $\mathbf{V}$ and not in $\mathbf{U}$ ensures that the decoder can detect whether the last row of $\mathbf{U}$ is deleted or not.

Case 1–(i): If there was a row deletion in $\mathbf{U}$, the decoder uses the non-deleted part of $\mathbf{p}_r$ to recover the deleted row except for the bit in the deleted column. The decoder can now use the properties of $\mathcal{VT}_{a,b}(n,n)$ to decode the column deletion in $\mathbf{U}$. Since any two consecutive

columns in $\mathbf{U}$ are different, the decoder can locate the exact position of the deleted column and recover its value. The position of the deleted column in $\mathbf{U}$ is the same as the deleted column in the whole array. Using $\mathbf{p}_c$, the decoder can recover the remaining part of the deleted column.

Case 1–(ii): If the deleted row was not in $\mathbf{U}$, the decoder uses $\mathcal{VT}_{a,b}(n,n)$ to recover the index of the deleted column and its value within $\mathbf{U}$. It uses $\mathbf{p}_c$ to recover the value of the deleted column outside of $\mathbf{U}$ (except for the bit in the intersection of the deleted row and column). Then, the decoder uses $\mathcal{VT}_{c,d}(n-1,n)$ to recover the index of the deleted row. Again, since any two consecutive rows in $\mathbf{V}$ are different, the decoder can recover the exact position of the deleted row. Using $\mathbf{p}_r$, the decoder recovers the value of the bits of the deleted row.

Case 2: Now assume that the last column of $\mathbf{C}$ is deleted. By looking at the last column of $\widetilde{\mathbf{C}}$, the decoder knows that the alternating sequence is missing thanks to the run of 0s inserted at the beginning of the second to last column of $\mathbf{C}$. Note that irrespective of the location of the row deletion, the last column will have a run of at least three 0s, which cannot happen in the alternating sequence. Therefore, the decoder knows that the last column is deleted and starts by looking at $\mathbf{V}$. Using the parity $\mathbf{p}_c$, the decoder recovers the missing part of the deleted column that is in $\mathbf{V}$ but not in $\mathbf{U}$. By construction, the first ℓ bits of the last column of $\mathbf{V}$ are set to 0 when encoding $\mathbf{V}$ using a VT code. Thus, the decoder recovers the whole missing column. By using the property of $\mathcal{VT}_{c,d}(n-1,n)$, the decoder recovers the index of the missing row and uses $\mathbf{p}_r$ to retrieve the value of the bits of this row. After recovering the deleted row, the decoder adds the alternating sequence to $\mathbf{U}$ and recovers the whole array $\mathbf{C}$.

b) Insertion correcting decoder: The decoder for $\mathcal{C}^{\text{cc}}_{(1,1),n}(a,b,c,d)$ receives as input a $(n+1)\times(n+1)$ array $\widetilde{\mathbf{C}}$ resulting from an $(1,1)$-criss-cross insertion of an array $\mathbf{C}$ of $\mathcal{C}^{\text{cc}}_{(1,1),n}(a,b,c,d)$ and works as follows. The decoder starts by looking at the first $(\ell+1)\times(n+1)$ subarray of $\widetilde{\mathbf{C}}$ (recall that $\ell=\log_2(n)$) and examines the last two columns.

Case 1: Assume the second to last column is not an alternating sequence (special insertions that we consider in Cases 2 and 3) and the last column is the alternating sequence. Using the alternating sequence, the decoder can detect whether or not there was a row insertion in $\mathbf{U}$. This is done by locating a run of length two in the alternating sequence.

Case 1–(i): If there was a row insertion in $\mathbf{U}$, the decoder has two candidates for the inserted row: the ones that cause the run of length two in the alternating sequence. Recall that the bit-wise sum of the n rows of $\mathbf{C}$ is known to the decoder (parity check constraint). The decoder verifies which of the two candidate rows does not satisfy the parity constraints, i.e., the decoder sums the $n-1$ remaining rows together with each of the candidate rows and checks the Hamming weight of the resulting vector. The row that results in a vector with Hamming weight more than 1 is the inserted row.[6] If both resulting vectors differ and result in Hamming weight 1, the decoder is confused between two candidates for the inserted row and two candidates for the inserted column. In this case, the decoder deletes both candidate rows and both candidate columns where a "1" appears in the resulting vectors and uses the deletion correction capability of the code to recover the original message. This works since

[6]The original row can only result in at most one 1 located in the position of the inserted column.

the inserted row and column were removed, i.e., the array is now affected by *one* row deletion and *one* column deletion.

Otherwise, the decoder removes the inserted row and uses the properties of $\mathcal{VT}_{a,b}(n,n)$ to decode the column insertion in $\mathbf{U}$. Since any two consecutive columns in $\mathbf{U}$ are different, the inserted column is either different from both adjacent columns or equal to only one. In the former case, the decoder recovers the exact position of the inserted column and removes it. The position of the inserted column in $\mathbf{U}$ is the same as the inserted column in the whole array. In the latter case, the decoder has two candidates of inserted columns. The decoder uses the column parity check to verify which column is inserted and removes it. Since the inserted row is removed, the decoder will have at most one column that does not satisfy the column parity check constraints. If both columns verify the parity constraints, then they are identical.

Case 1–(ii): If the inserted row was not in $\mathbf{U}$, i.e., the alternating sequence is intact, the decoder uses $\mathcal{VT}_{a,b}(n,n)$ to recover the index and value of the inserted column in $\mathbf{U}$. If this column in $\mathbf{U}$ differs from its adjacent columns in $\mathbf{U}$, the decoder removes the whole column and corrects the inserted row. However, if the inserted column in $\mathbf{U}$ equals one of its adjacent columns (since any consecutive columns are different), then the decoder has two candidates of inserted columns. Similarly to Case 1–(i), the decoder uses the column parity check constraints to verify which column is the inserted one. After removing the inserted column, the decoder uses $\mathcal{VT}_{c,d}(n-1,n)$ to recover the index of the inserted row. Again, the decoder removes the whole row if the inserted row in $\mathbf{V}$ differs from both adjacent rows in $\mathbf{V}$. Otherwise, the decoder has two candidates for the inserted rows; therefore, the decoder uses the row parity check to recover the exact position of the inserted row.

Case 2: Now assume that the two last columns of $\mathbf{U}$ are identical. Due to the four zeros in the second to last column of $\mathbf{C}$ (now the third to last column in $\widetilde{\mathbf{C}}$), the decoder detects that a column insertion happened in one of the last two columns of $\mathbf{U}$. The decoder uses the column parity check to verify which column is the inserted one. If both columns satisfy the parity check constraints, they are identical. If both columns violate the parity check constraints in one position, similarly to Case 1–(i), the decoder deletes both columns, and both rows where the columns do not satisfy the parity check constraint and uses the deletion correction capability of the code. After removing the inserted column, the decoder examines the alternating sequence to check if the inserted row is in $\mathbf{U}$. If this is the case, the decoder uses the row parity check to verify which row is inserted and removes it. If the inserted row is not in $\mathbf{U}$, the decoder uses $\mathcal{VT}_{c,d}(n-1,n)$ and the column parity check to recover the exact index of the inserted row.

Case 3: Assume that the last column of $\mathbf{U}$ is not the alternating sequence. Thus, the last column is inserted. The decoder removes this column and detects which row is inserted, as explained in the previous case. □

We successfully showed Theorem 5.3. The redundancy of the construction $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ is

$$r^{\mathrm{cc}}_{(1,1)}(n) = 2n + 4\log_2(n) + 7 + 2\log_2(e)$$

as stated in Corollary 5.3. Note that in [BWS+21], a (non-asymptotic) lower bound on the

redundancy of any $(1,1)$-criss-cross insertion/deletion-correcting code is derived. In fact, for the simplified $(1,1)$-criss-cross error model, roughly the same minimal redundancy is needed as for the general 2-criss-cross model (see Section 5.4). Hence, the construction is approximately $2\log_2(n)$ bits away from the lower bound. The proof of Corollary 5.3 will be given in Section A.1. We provide a short intuition in the following. The two parity vectors for the rows and columns contribute $2n - \log_2(n)$ redundancy bits. Further, we fix the top right bits of length $\log_2(n) + 1$ in the rightmost column of the codeword arrays to the alternating sequence. For locating the deleted row and column, two binary extensions of n-ary VT codes are used with the additional constraint that no neighboring symbol is the same. Each of these subarrays contributes roughly $2\log_2(n)$ redundancy.

On an additional node, we remark that a slightly modified construction having an explicit encoder is given in [BWS+21], which is based on the systematic q-ary VT code construction from [Ten84].

5.6 Multiple Criss-Cross Construction

In this section, we present an existential construction of t-criss-cross insertion/deletion correcting codes. The construction follows a similar strategy as presented for the simplified error model. We start with an intuitive road map to our code construction and then formally define each ingredient.

5.6.1 Road Map

Our construction uses structured arrays so that the indices of the inserted/deleted rows and columns can be exactly recovered. Then, the set of structured arrays is intersected with arrays of a Gabidulin code (that can correct row/column erasures) to retrieve the arrays of the code. The structure is depicted in Figure 5.6.

We structure the $n \times n$ codewords $\mathbf{C}$ as follows. We protect the columns with indices between $t\log_2(n) + 1$ and $n - (t+1)^2$ using $t\log_2(n)$ codes where each one is a binary systematic t-insdel correcting code. These codes are divided into t blocks of size $\log_2(n)$. We impose a *window constraint* on the columns of the systematic part of every block. This constraint ensures that every $t+1$ consecutive columns are different. Therefore, the indices of the deleted columns within the systematic part can be located using all $\log_2(n)$ insdel correcting codes of any block (Claim 5.5). In the case of insertions, the indices of the inserted columns within the systematic part can be located up to an interval of length at most $2t$, containing at most t columns of the original array. This ambiguity arises when the inserted rows/columns are equal to collections of rows/columns of the original array within an interval. We call this phenomenon *block confusions* (see Section 5.6.2 and Claim 5.6).

In the redundancy part, runs may exist. Thus, the recovery of the index of the deleted columns is only guaranteed within the corresponding run. To recover the exact location of the deleted columns here, we protect the redundancy part of the codes by appending (from below) what we call a *locator array* that can detect the exact positions of column deletions within this part (Claim 5.3). We call this array $\mathbf{T}^{(1)}$. In case of insertions, the locator array is used to recover the index of the inserted columns up to block confusions of length at most

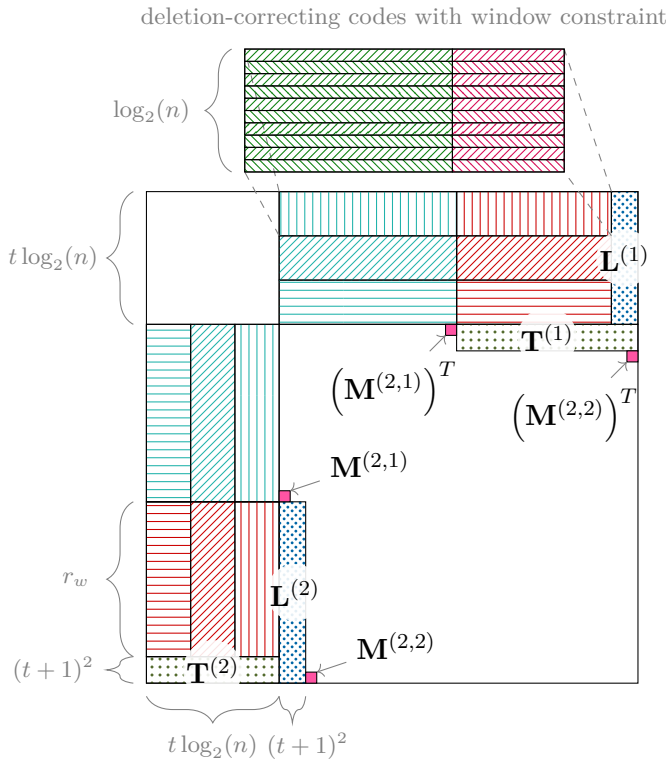


Figure 5.6: Illustration of an array contained in the locator set $\mathcal{L}_t(n)$ for $t = 3$. In the first $t\log_2(n)$ rows there are t blocks each consisting of a systematic part (cyan) and a redundancy part (red) with $r_w \leq (4t+1)\log_2(n)$. Each row is encoded using a systematic t-deletion correcting code (zoomed in part). In addition, a window constraint is imposed in the systematic part of each block. Those blocks are used to locate column deletions. This structure is protected with the arrays $\mathbf{L}^{(1)}$ (blue) against row deletions and $\mathbf{T}^{(1)}$ (brown) against column deletions. Lastly, to locate the borders of $\mathbf{T}^{(1)}$ we use the marker arrays $\mathbf{M}^{(2,1)}$ and $\mathbf{M}^{(2,2)}$ (pink). A symmetric structure locates row deletions.

$2t$ (Claim 5.4).

Note that for the window constraint to work efficiently, we must have all $\log_2(n)$ insdel correcting codes of the considered block. Therefore, we use the subarray $\mathbf{C}_{[1:t\log_2 n],[n-(t+1)^2:n]}$ as a locator array $\mathbf{L}^{(1)}$ that can detect the exact position of a deleted row within the first $t\log_2(n)$ rows (Claim 5.3). As a result, if all t deletions are row deletions within the first $t\log_2(n)$ rows, then the locator array is enough to recover all the indices (Lemma 5.4). Otherwise, we have at least one block of the t blocks unaffected by a row deletion. This block is used to recover the deleted columns with indices in the range $t\log_2(n)+1$ to $n-(t+1)^2$ (Lemma 5.5). The same arguments hold for the insertion case (Claim 5.4, Lemma 5.6 and Lemma 5.7). The only difference is the possibility of the inserted row/column causing ambiguities for the exact indices of the insertions.

One more step is needed. We must be able to locate the position of the locator arrays

within the resulting $(n-t_\mathrm{r})\times(n-t_\mathrm{c})$ array $\widetilde{\mathbf{C}}$. Therefore, we put four *marker arrays* for the locator arrays that are detectable even after t insertions or deletions. We call those arrays $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$.

The same structure (transposed) is used to index the rows. In addition, the columns with indices between 1 and $t\log_2(n)$ are protected by the locator array for safeguarding the insdel correcting codes indexing the rows. Note the claims and lemmas mentioned before also include the statements to recover the row indices.

The whole code is intersected with a linear Gabidulin code [Gab85] that can correct row/column erasures. Once the positions of the deleted rows and columns are known to the decoder, those positions are marked as erasures and corrected using the Gabidulin code. In the case of insertions, all inserted rows and columns with exactly recovered indices are removed. If inserted rows/columns create block confusions, we can delete all involved rows/columns and insert erasures. The window constraint guarantees that we do not delete more than t rows/columns of the original array by this procedure, hence assuring us that we do not exceed the erasure correction capability of the Gabidulin code.

In the following subsections, we formally define the five main ingredients of our code:

1. The locator arrays (see Section 5.6.2);
2. The binary systematic t-insdel correcting codes with window constraints (see Section 5.6.3);
3. The marker arrays (see Section 5.6.4);
4. The *locator set* which is the combination of all the previously mentioned parts (see Section 5.6.5);
5. A Gabidulin code [Gab85] which is used to correct row/column erasures.

Afterward, the formal definition of the code is given in Section 5.6.6. The proof will be presented by providing an explicit decoder in Section 5.7. Further, the redundancy of the construction will be analyzed in Section 5.8.

5.6.2 Locator Arrays

For a positive integer a, we denote by $\mathbf{I}_a$ the identity array of dimension $a\times a$ and by $\mathbf{1}_a$ and $\mathbf{0}_a$ the all-one row vector and all-zero row vector of length a, respectively. We use $\otimes$ to indicate the Kronecker product. We thus have the following definition from [Hag20].

Definition 5.3 (Locator arrays)**.** *We set* $\mathbf{L}'\in {\Sigma_2}^{(t+1)\times(t+1)^2}$ *as* $\mathbf{L}'\triangleq \mathbf{I}_{t+1}\otimes\mathbf{1}_{t+1}$*. More precisely,* $\mathbf{L}'$ *has the following structure*

$$\mathbf{L}'=\begin{pmatrix}\mathbf{1}_{t+1} & \mathbf{0}_{t+1} & \dots & \mathbf{0}_{t+1}\\ \mathbf{0}_{t+1} & \mathbf{1}_{t+1} & \dots & \mathbf{0}_{t+1}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{0}_{t+1} & \mathbf{0}_{t+1} & \dots & \mathbf{1}_{t+1}\end{pmatrix}.$$

Let s *be a multiple of* $(t+1)$ *such that* $s\geq\lceil\frac{t}{2}\rceil(t+1)$*. We define the locator array* $\mathbf{L}_s\in {\Sigma_2}^{s\times(t+1)^2}$ *as*

$$\mathbf{L}_s\triangleq \mathbf{1}_{\frac{s}{t+1}}^{T}\otimes\mathbf{L}'.$$

Moreover, we define the locator array $\mathbf{T}_s \in {\Sigma_2}^{(t+1)^2 \times s}$ *to be the transpose of* $\mathbf{L}_s$*, i.e.,*

$$\mathbf{T}_s \triangleq \mathbf{L}_s^T = \mathbf{1}_{\frac{s}{t+1}} \otimes {\mathbf{L}'}^T.$$

Throughout this chapter, we drop s in the notation $\mathbf{L}_s$ and $\mathbf{T}_s$ when the value of s is evident from the context.

Claim 5.3 (Deletion detection in $\mathbf{L}_s$ and $\mathbf{T}_s$)**.** *Let* $\mathbf{L}_s$ *be an array affected by* t_{r} *row and* t_{c} *column deletions such that* $t_{\mathrm{r}} + t_{\mathrm{c}} = t$*. Divide* $\mathbf{L}_s$ *into* $(t+1)$ *subarrays each consisting of* $(t+1)$ *consecutive columns of* $\mathbf{L}_s$*. By examining* $\widetilde{\mathbf{L}}_s$*, we can locate the exact positions of the deleted rows. We can also determine the number of column deletions that happened in each subarray of* $\mathbf{L}_s$*.*

Let $\mathbf{T}_s$ *be an array affected by* t_{r} *row and* t_{c} *column deletions such that* $t_{\mathrm{r}} + t_{\mathrm{c}} = t$*. The same statement above for* $\mathbf{L}_s$ *holds for* $\mathbf{T}_s$ *by switching rows for columns.*

Proof. We prove the first part of the claim, while the second part follows similarly since $\mathbf{T}_s = \mathbf{L}_s^T$ and row deletions in $\mathbf{T}_s$ can be seen as column deletions in $\mathbf{L}_s$ and vice versa.

By construction of $\mathbf{L}_s$, for any $i \in [s-t]$ and $j \in [t]$ it holds that $\mathbf{L}_{i,[(t+1)^2]} \neq \mathbf{L}_{i+j,[(t+1)^2]}$. This property holds even in the presence of at most t column deletions in $\mathbf{L}_s$. Thus, due to the fixed structure of $\mathbf{L}_s$, one can uniquely determine the exact indices of the deleted rows.

Moreover, we divide $\mathbf{L}_s$ into subarrays consisting of $(t+1)$ columns. For any $a \in [t+1]$ and $b \in [t+1]$, we have $\mathbf{L}_{[s],(a-1)(t+1)+1} = \mathbf{L}_{[s],(a-1)(t+1)+b}$. In other words, we have $(t+1)$ identical columns in a subarray. This property holds even if there were at most t row deletions in $\mathbf{L}_s$. Furthermore, for $s \geq \lceil \frac{t}{2} \rceil (t+1)$ and $a, b, c \in [t+1]$, it always holds that $\mathbf{L}_{[s],(a-1)(t+1)+1} \neq \mathbf{L}_{[s],(c-1)(t+1)+b}$, unless $a = c$. Therefore, we can determine the deleted columns within any subarray by counting the missing columns. □

We briefly elaborate on the dimension constraint of $\mathbf{L}_s$, i.e., $s \geq \lceil \frac{t}{2} \rceil (t+1)$. If $\mathbf{L}_s$ consists of less than $\lceil \frac{t}{2} \rceil$ copies of $\mathbf{L}'$, then a deletion of two consecutive rows in each block will lead to an impossibility in locating a column deletion within two adjacent subarrays. Recall that $\mathbf{1}_a$ is the all-one vector of length a and let $t = 3$ and $s = 4$, we assume the following deletion pattern:

$$\underbrace{\begin{pmatrix} \mathbf{1}_4 & \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{1}_4 & \mathbf{0}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{1}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{1}_4 \end{pmatrix}}_{\mathbf{L}'} \xrightarrow[\text{2nd column deletion}]{\text{1st \& 2nd row deletion}} \underbrace{\begin{pmatrix} \mathbf{0}_7 & \mathbf{1}_4 & \mathbf{0}_4 \\ \mathbf{0}_7 & \mathbf{0}_4 & \mathbf{1}_4 \end{pmatrix}}_{\widetilde{\mathbf{L}}'}$$

Even though we can locate the row deletions, we cannot locate the column deletion within a subarray of $(t+1)$ columns. Thus, it is necessary to have another copy of $\mathbf{L}'$ within $\mathbf{L}_s$ to guarantee a successful detection of column deletions. In the general case, this extends to needing at least $\lceil \frac{t}{2} \rceil$ copies of $\mathbf{L}'$ in $\mathbf{L}_s$. This example can be directly applied for $\mathbf{T}_s$ by switching in the argument rows and columns.

For the following statements and proofs in case of insertions we have to define the terminology of block confusions.

Block confusions: Consider an array $\mathbf{Z} \in {\Sigma_2}^{n\times m}$. Let $\widetilde{\mathbf{Z}}$ be the resulting array after t-criss-cross insertions in $\mathbf{Z}$. Given a subset $\mathcal{B} \subseteq [n]$, an integer $a \in [n]$, and a non-negative integer $c \leq n$, we define a row block confusion in $\widetilde{\mathbf{Z}}$ if $\widetilde{\mathbf{Z}}_{a+c',[m]} = \widetilde{\mathbf{Z}}_{a+b+c',[m]}$ for all $b \in \mathcal{B}$ and all non-negative c' such that $c' \leq c$. In other words, we declare a block confusion after insertions when a collection of c' consecutive rows in $\widetilde{\mathbf{Z}}$ are the same as other collections of consecutive rows within an interval determined by a and $\mathcal{B}$. This leads to the fact that an insertion locating algorithm cannot determine the exact location (up to the block confusion) of the inserted rows even when knowing $\mathbf{Z}$. For convenience, we provide the following illustration of a row block confusion.

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}}_{\mathbf{Z}} \xrightarrow[\text{1st \& 2nd row}]{\text{insertion in}} \underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}}_{\widetilde{\mathbf{Z}}}$$

Given both $\mathbf{Z}$ and $\widetilde{\mathbf{Z}}$, an insertion locating algorithm cannot distinguish whether the insertion happened in the first and second row, the second and third, or the third and fourth row. We call this a row block confusion with parameters $a = 1$, $\mathcal{B} = \{2\}$, and $c = 1$.

If $\mathbf{Z}$ satisfies predetermined properties, we can narrow down the parameter range of the possible block confusions. Let $\mathbf{Z} \in {\Sigma_2}^{n\times m}$ be an array satisfying $\mathbf{Z}_{i,[m]} \neq \mathbf{Z}_{i+j,[m]}$ for all $i \in [n-t]$ and $j \in [t]$. Then, it follows that $\mathcal{B} \subseteq [t]$, $a \in [n]$, and $0 \leq c \leq t$. Therefore, the window of a row block confusion, defined as $[a, \max(\mathcal{B}) + c]$, consists of at most $2t$ rows. In addition, within this window, there exists at least one and at most t rows of the original array $\mathbf{Z}$. Moreover, the bounds on the maximum number of original columns in the confusion and the *length* of the confusion, defined as the total number of rows in the block confusion, scale proportionally with the number of insertions responsible for the confusion. Note that the property mentioned above on the array $\mathbf{Z}$ is satisfied by $\mathbf{L}_s$.

We define a column block confusion similarly. The array $\mathbf{T}_s$ satisfies the desired properties on the columns to limit the parameter range of column block confusions. In the sequel, we will drop the row and column term when referring to block confusions when it is clear from the context.

Claim 5.4 (Insertion detection in $\mathbf{L}_s$ and $\mathbf{T}_s$)**.** *Let $\mathbf{L}_s$ be affected by t_r row and t_c column insertions such that $t_\mathrm{r} + t_\mathrm{c} = t$. Divide $\mathbf{L}_s$ into $(t+1)$ subarrays each consisting of $(t+1)$ consecutive columns of $\mathbf{L}_s$. By examining $\widetilde{\mathbf{L}}_s$, we can locate the positions of the inserted rows up to a row block confusion of length at most $2t$ containing at most t columns of the original array $\mathbf{L}_s$. We can also determine the number of column insertions (and possibly their positions up to a column block confusion within the subarray or within the adjacent subarray) that happened in each subarray of $\mathbf{L}_s$.*

Let $\mathbf{T}_s$ be affected by t_r row and t_c column insertions such that $t_\mathrm{r} + t_\mathrm{c} = t$. The same statement above for $\mathbf{L}_s$ holds for $\mathbf{T}_s$ by switching rows for columns.

Proof. The proof follows the same steps as in Claim 5.3. In terms of row insertions, the difference is that if the inserted rows in $\mathbf{L}_s$ create a block confusion, then we cannot distinguish between the original rows and the inserted rows in a window of length at most $2t$

rows. This follows from the construction of $\mathbf{L}_s$. In terms of column insertions, if the inserted column differs from the column run in which it is inserted and from both adjacent column runs, it can be precisely located. Otherwise, we can only count the number of insertions in each block up to a confusion with an adjacent block. This is due to possible column block confusions at the column runs located in adjacent subarrays. □

5.6.3 Insdel Correcting Codes with Window Constraints

First, we focus on two main components of this ingredient in our construction.

Insdel correcting codes: We use the construction of [SGB20b] for our binary systematic t-insdel correcting code (already briefly discussed in Section 3.4). We briefly recall the construction results of [SGB20b]. Given a sequence $\mathbf{k} \in {\Sigma_2}^k$, one can compute a redundancy vector $\mathbf{r_k} \in {\Sigma_2}^{r_k}$ with $r_k \leq 4t\log_2(k) + o(\log_2(k))$. The resulting sequence $(\mathbf{k}|\mathbf{r_k})$ can be uniquely recovered after t insdel errors. Note that $\mathbf{r_k}$ is a function of the information $\mathbf{k}$ and r_k is a function of the information length k and the number of insdel errors t.

Window constraint: We define the window constraint as the set $\mathcal{W}_t(\ell, w) \subseteq {\Sigma_2}^{\ell\times w}$, where for any $\mathbf{W} \in \mathcal{W}_t(\ell, w)$, $i \in [w-t]$ and $j \in [t]$, it holds that $\mathbf{W}_{[\ell],i} \neq \mathbf{W}_{[\ell],i+j}$.

For an array $\mathbf{W} \in \mathcal{W}_t(\ell, w)$, let $\mathbf{R_W} \in {\Sigma_2}^{\ell\times r_w}$ be the array formed such that for any $i \in [\ell]$ the i^{th} row of $\mathbf{R_W}$ is the redundancy vector corresponding to the i^{th} row of $\mathbf{W}$; computed using the systematic construction in [SGB20b]. We refer to the array $\mathbf{R_W} \in {\Sigma_2}^{\ell\times r_w}$ as the redundancy array. Let $m \triangleq w + r_w$, we define $\mathcal{D}_t^{(1)}(\ell, m)$ as the set of all arrays resulting from the concatenation of $\mathbf{W}$ and $\mathbf{R_W}$, i.e.,

$$\mathcal{D}_t^{(1)}(\ell, m) \triangleq \left\{\mathbf{D} \in {\Sigma_2}^{\ell\times m} : \mathbf{D} = (\mathbf{W} \parallel \mathbf{R_W}) \text{ s.t. } \mathbf{W} \in \mathcal{W}_t(\ell, w)\right\}.$$

In other words, $\mathcal{D}_t^{(1)}(\ell, m)$ is the set of binary systematic t-insdel correcting codes in which the systematic part satisfies the imposed window constraint. This set will be used to index the columns of our arrays in the constructed code. We define $\mathcal{D}_t^{(2)}(\ell, m) \triangleq \left\{\mathbf{D}^T : \mathbf{D} \in \mathcal{D}_t^{(1)}(\ell, m)\right\}$. This set is going to be used to index the rows.

Claim 5.5. *Let $\mathbf{D} = (\mathbf{W} \parallel \mathbf{R_W}) \in \mathcal{D}_t^{(1)}(\ell, m)$ be an array affected by t column deletions and no row deletions, we can locate the exact positions of the deleted columns in the subarray $\mathbf{W}$.*

The same holds for any array in $\mathcal{D}_t^{(2)}(\ell, m)$ by switching in the argument rows and columns.

Proof. Assume $\widetilde{\mathbf{D}} = (\widetilde{\mathbf{W}} \parallel \widetilde{\mathbf{R}}_{\mathbf{W}})$ is the array obtained after the deletions. For each row in $\widetilde{\mathbf{W}}$, we can use the corresponding redundancy in $\widetilde{\mathbf{R}}_{\mathbf{W}}$ to correct the deletions that happened in this row [SGB20b]. We start by looking at the position of the first recovered bit in each row. In each row, this position may be unique or may be in an interval of possible positions (run). The exact location of the column is then determined by the unique position in which all runs (of all rows) intersect. The intersection is guaranteed to be unique by the imposed window constraint since for any $i \in [w-t]$ and $j \in [t]$, it holds that $\mathbf{W}_{[\ell],i} \neq \mathbf{W}_{[\ell],i+j}$. This process is repeated for all recovered bits until all t positions are determined.

A similar argument follows for the second statement of the claim. □

Claim 5.6. *Given an array* $\mathbf{D} = (\mathbf{W} \parallel \mathbf{R}_{\mathbf{W}}) \in \mathcal{D}_t^{(1)}(\ell, m)$ *affected by* t *column insertions and no row insertions, we can locate the positions of the inserted columns in the subarray* $\mathbf{W}$ *up to column block confusion of length at most* $2t$ *containing at most* t *columns of the original array* $\mathbf{D}$.

The same holds for any array in $\mathcal{D}_t^{(2)}(\ell, m)$ *by switching in the argument rows and columns.*

Proof. The proof follows the same technique as the proof of Claim 5.5. The only exception arises if the inserted columns create a column block confusion. However, the window constraint guarantees that in the worst case, the block confusion occurs in a window of length at most $2t$. Moreover, within this block confusion, there exists at most t columns of the original array. □

5.6.4 Marker Arrays

We define the following arrays of dimension $(t+1) \times (t+1)$ which will operate as *markers* to locate the position of the locator arrays in the resulting $\widetilde{\mathbf{C}}$. Recall that we use four locator arrays in our construction, namely $\mathbf{L}^{(1)}$, $\mathbf{L}^{(2)}$, $\mathbf{T}^{(1)}$, and $\mathbf{T}^{(2)}$ (see Figure 5.6). We only need marker arrays for $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(2)}$. The position of $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(2)}$ can be then determined.

The first marker array $\mathbf{M}^{(2,1)}$, put on top of $\mathbf{L}^{(2)}$, consists of the first $t+1$ columns of $\mathbf{L}'$. The second marker array $\mathbf{M}^{(2,2)}$, put on the right of $\mathbf{L}^{(2)}$, consists of the complement of the last $t+1$ columns of $\mathbf{L}'$. The marker arrays $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$ are the transpose of $\mathbf{M}^{(2,1)}$ and $\mathbf{M}^{(2,2)}$, respectively.

5.6.5 Locator Set

We formally define the sets of arrays in $\Sigma_2^{n\times n}$ that form our code. Let $\mathbf{X} \in \Sigma_2^{n\times n}$, we start with the set of arrays that are used to index the columns. This set is denoted by $\mathcal{H}_t(\ell, n)$. The arrays in this set have the first $t\ell$ columns divided into t blocks. The columns whose indices are between $t\ell+1$ and $n-(t+1)^2$ of each row consist of a systematic t-deletion correcting code in which the systematic part satisfies the window constraint. We can write

$$\mathcal{H}_t(\ell, n) \triangleq \left\{ \mathbf{X} : \mathbf{X}_{[(a-1)\ell+1:a\ell],[t\ell+1:n-(t+1)^2]} \in \mathcal{D}_t^{(1)}(\ell, n-t\ell-(t+1)^2) \ \forall\, a \in [t] \right\} .$$

The set of arrays $\mathcal{V}_t(\ell, n)$ that are used to index the rows is defined similarly to $\mathcal{H}_t(\ell, n)$ by replacing columns with rows

$$\mathcal{V}_t(\ell, n) \triangleq \left\{ \mathbf{X} : \mathbf{X}_{[t\ell+1:n-(t+1)^2],[(b-1)\ell+1:b\ell]} \in \mathcal{D}_t^{(2)}(\ell, n-t\ell-(t+1)^2) \ \forall\, b \in [t] \right\} .$$

For a value of r_w that divides[7] $t+1$, the set of arrays $\mathcal{E}_t(\ell, n)$ that contain the locator

[7] If the value of r_w does not divide $t+1$, then one can expand the dimension of the locator arrays in $\mathcal{E}_t(\ell, n)$ to the next multiple of $t+1$ that is greater than $r_w + (t+1)^2$.

arrays in the positions shown in Figure 5.6 is defined as follows.

$$\mathcal{E}_t(\ell, n) \triangleq \left\{ \mathbf{X} : \begin{array}{l} \mathbf{X}_{[1:t\ell],[n-(t+1)^2+1:n]} = \mathbf{L}_{t\ell}\,, \\ \mathbf{X}_{[t\ell+1:t\ell+(t+1)^2],[n-r_w-(t+1)^2+1:n]} = \mathbf{T}_{r_w+(t+1)^2}\,, \\ \mathbf{X}_{[n-(t+1)^2+1:n],[1:t\ell]} = \mathbf{T}_{t\ell}\,, \\ \mathbf{X}_{[n-r_w-(t+1)^2+1:n],[t\ell+1:t\ell+(t+1)^2]} = \mathbf{L}_{r_w+(t+1)^2}\,, \end{array} \right\}.$$

The set of arrays that contain the marker arrays in the positions shown in Figure 5.6 is defined as follows.

$$\mathcal{M}_t(\ell, n) \triangleq \left\{ \mathbf{X} : \begin{array}{l} \mathbf{X}_{[t\ell+1:t\ell+(t+1)],[n-r_w-(t+1)^2-(t+1)+1:n-r_w-(t+1)^2]} = \mathbf{M}^{(1,1)}\,, \\ \mathbf{X}_{[t\ell+(t+1)^2+1:t\ell+(t+1)^2+(t+1)],[n-(t+1)+1:n]} = \mathbf{M}^{(1,2)}\,, \\ \mathbf{X}_{[n-r_w-(t+1)^2-(t+1)+1:n-r_w-(t+1)^2],[t\ell+1:t\ell+(t+1)]} = \mathbf{M}^{(2,1)}\,, \\ \mathbf{X}_{[n-(t+1)+1:n],[t\ell+(t+1)^2+1:t\ell+(t+1)^+(t+1)]} = \mathbf{M}^{(2,2)}\,, \end{array} \right\}.$$

We can conclude this subsection by defining the *locator set* that is the set of all arrays that have the structure required by our code to recover the indices of the inserted or deleted columns and rows. The locator set is the intersection of all the previously defined sets.

Definition 5.4 (Locator Set)**.** *We define the following set:*

$$\mathcal{L}_t(n) \triangleq \mathcal{H}_t(\ell, n) \cap \mathcal{V}_t(\ell, n) \cap \mathcal{E}_t(\ell, n) \cap \mathcal{M}_t(\ell, n).$$

The defining parameters of $\mathcal{L}_t(n)$ are only t and n. By fixing those, all other parameters can be obtained from the imposed constraints. The most noteworthy parameters are w and r_w, which are functions of n and t. Moreover, we point out that a good choice for the parameter ℓ is $\log_2(n)$, which is mainly motivated by the redundancy optimization of the construction, thoroughly discussed in Section 5.8. Additionally, due to the aforementioned constraints on the locator arrays, $t\ell$ and $r_w + (t+1)^2$ need to be multiples of $(t+1)$ and $t\ell \geq \lceil \frac{t}{2} \rceil (t+1)$. We refer to Figure 5.6 for an illustration of such arrays. Our construction works only when $\log_2(n)$ is a multiple of $t+1$ since we choose $\ell = \log_2(n)$.

5.6.6 Construction

We write $\mathcal{C}_{\text{Gab}}(n, t) \subseteq \mathbb{F}_2^{n \times n}$ to refer to a linear[8] Gabidulin code which is able to correct any pattern of t_r row and t_c column erasures in an $n \times n$ array as long as $t_\text{r} + t_\text{c} = t$ [GP08]. This is equivalent to stating that its minimum rank distance[9] is at least $t+1$. Now, we can present our existential construction.

[8] Note that such a Gabidulin code can be represented as a set of vectors in $\mathbb{F}_{2^n}^n$ as well and is linear in $\mathbb{F}_{2^n}$. For our application, it is sufficient that such a Gabidulin code is also $\mathbb{F}_2$-linear, and we will always represent the codewords as binary $n \times n$ matrices.

[9] For the definition of the rank distance, see [Gab85]

Construction 5.2. *The code* $\mathcal{C}_{t,n}^{\mathrm{cc}} \subseteq \Sigma_2^{n\times n}$ *is the set of arrays that belong to*

$$\mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(n,t)\,.$$

Theorem 5.4. *The code* $\mathcal{C}_{t,n}^{\mathrm{cc}}$ *described in Construction 5.2 is a* t*-criss-cross insertion/deletion correcting code.*

A rough concept of our construction is as follows. We assume that the decoder knows whether a t-criss-cross deletion or insertion happened from the received array's dimension. In our codewords, we first introduce the structure $\mathcal{L}_t(n)$ to locate the indices of the inserted or deleted columns and rows. With this knowledge, we can introduce erasures into the missing rows and columns and convert the deletion problem into an erasure problem, which the Gabidulin code $\mathcal{C}_{\mathrm{Gab}}(n,t)$ [GP08] can solve. We call this type of decoding the *locate-decode strategy*. The Theorem 5.4 will be proven by providing a generic decoding strategy in the next section.

5.7 Decoder for Code Construction

Assume a codeword $\mathbf{C} \in \mathcal{C}_{t,n}^{\mathrm{cc}}$ is transmitted and let t_{c} and t_{r} be such that $t_{\mathrm{c}} + t_{\mathrm{r}} = t$. The decoder receives an array $\widetilde{\mathbf{C}} \in \Sigma_2^{(n-t_{\mathrm{r}})\times(n-t_{\mathrm{c}})}$ obtained from $\mathbf{C}$ by t_{r} row and t_{c} column deletions or an array $\widetilde{\mathbf{C}} \in \Sigma_2^{(n+t_{\mathrm{r}})\times(n+t_{\mathrm{c}})}$ obtained from $\mathbf{C}$ by t_{r} row and t_{c} column insertions. The dimension of the received array is assumed to be known to the decoder. As mentioned, we first focus on locating the indices of inserted/deleted rows and columns. Given the indices, we can transform the problem into an erasure-correcting problem. A decoder flow for correcting deletions is depicted in Figure 5.7.

5.7.1 Locating the indices

Let us denote the set of indices of the rows and columns that got inserted or deleted by $\mathcal{I}^{(t_{\mathrm{r}})} \subset [n+t_{\mathrm{r}}]$ and $\mathcal{I}^{(t_{\mathrm{c}})} \subset [n+t_{\mathrm{c}}]$, respectively, with $|\mathcal{I}^{(t_{\mathrm{r}})}| + |\mathcal{I}^{(t_{\mathrm{c}})}| = t_{\mathrm{r}} + t_{\mathrm{c}} = t$. For clarity of presentation, we first present the decoding strategy for locating deletions. Subsequently, we present the decoding strategy for locating insertions.

Claim 5.7. *Given the array* $\widetilde{\mathbf{C}}$ *affected by* t_{r} *row and* t_{c} *column deletions, the marker arrays* $\widetilde{\mathbf{M}}^{(1,1)}$, $\widetilde{\mathbf{M}}^{(1,2)}$, $\widetilde{\mathbf{M}}^{(2,1)}$, *and* $\widetilde{\mathbf{M}}^{(2,2)}$ *can be located.*

Proof. We will focus on locating the arrays $\widetilde{\mathbf{M}}^{(2,1)}$ and $\widetilde{\mathbf{M}}^{(2,2)}$. A similar proof can be given to find the arrays $\widetilde{\mathbf{M}}^{(1,1)}$ and $\widetilde{\mathbf{M}}^{(1,2)}$ by exploiting the symmetric properties of $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(1)}$.

Using Claim 5.3 we can locate the leftmost column of $\widetilde{\mathbf{L}}^{(2)}$. Moreover, the bottom row of $\widetilde{\mathbf{L}}^{(2)}$ is given directly by the position of $\mathbf{L}^{(2)}$ in the codeword itself.

Note that we only have to detect row or column deletions rather than exactly locating them. Our goal is to locate the rightmost column of $\widetilde{\mathbf{L}}^{(2)}$. Recall that in the last $t+1$ rows of $\widetilde{\mathbf{L}}^{(2)}$ there is for sure an $\widetilde{\mathbf{L}'}$ which originates from $\mathbf{L}'$ affected by possible row and column deletions. By a similar argument as in Claim 5.3, one can detect the number of column deletions that happened in each subarray of $\mathbf{L}'$ of $t+1$ consecutive columns, starting from the left. This stems from the fact that number of column runs in $\widetilde{\mathbf{L}}'$ is $t+1$ minus the number

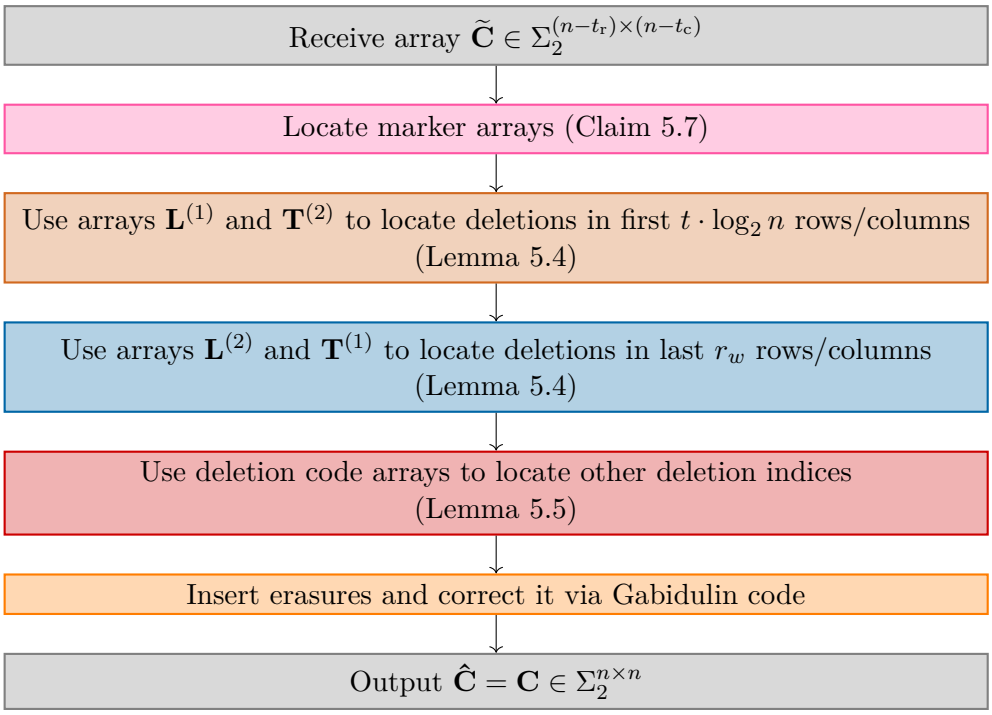


Figure 5.7: The decoder flow for correcting a t-criss-cross deletion in codewords $\mathbf{C} \in \mathcal{C}_{t,n}^{\text{cc}}$.

of row deletions in $\mathbf{L}'$. Recall that from Claim 5.3, we know the number of row deletions in $\mathbf{L}'$. The remaining ingredient is to know when $\widetilde{\mathbf{L}}'$ ends. This is guaranteed since the columns of the marker $\mathbf{M}^{(2,2)}$ are the complement of the last $t+1$ columns in $\mathbf{L}'$. Therefore, we are guaranteed to have at least one column of $\widetilde{\mathbf{M}}^{(2,2)}$ marking the end of $\widetilde{\mathbf{L}}'$ even in the presence of row or column deletions.

Given the rightmost column of $\widetilde{\mathbf{L}}^{(2)}$ we focus on locating the marker $\widetilde{\mathbf{M}}^{(2,1)}$ and therefore the topmost row of $\widetilde{\mathbf{L}}^{(2)}$. Recall that $\mathbf{L}^{(2)}$ consists of $\frac{s}{t+1}$ arrays $\mathbf{L}'$ stacked on top of each other. Using the same argument as in Claim 5.3, we can locate every row deletion in $\mathbf{L}^{(2)}$ until the topmost subarray $\mathbf{L}'$. This is true even in the presence of column deletions since column deletions do not change the fact that every $t+1$ consecutive rows in $\mathbf{L}^{(2)}$ are different. By the choice of the marker $\mathbf{M}^{(2,1)}$, we ensure that the number of row deletions in the topmost $\mathbf{L}'$ can be detected. This is true because the marker has the same structure as the first $t+1$ columns of $\mathbf{L}'$. □

Lemma 5.4. *Given the array $\widetilde{\mathbf{C}}$ affected by t_r row and t_c column deletions, any row with index $i \in \mathcal{I}^{(t_r)}$ such that $1 \leq i \leq t\ell$ or $n - r_w - (t+1)^2 < i \leq n$ can be exactly recovered. Similarly, any column index $j \in \mathcal{I}^{(t_c)}$ such that $1 \leq j \leq t\ell$ or $n - r_w - (t+1)^2 < j \leq n$ can be exactly recovered.*

Proof. We focus on recovering the indices of the deleted rows such that $i \in \mathcal{I}^{(t_r)}$ and the indices of the deleted columns such that $j \in \mathcal{I}^{(t_c)}$ that satisfy $1 \leq i \leq t\ell$ and $n - r_w - (t+1)^2 < j \leq n$. Recovering the remaining indices of the statement follows by the symmetry of the construction.

From Claim 5.7, the location of $\widetilde{\mathbf{L}}^{(1)}$ and $\widetilde{\mathbf{T}}^{(1)}$ in $\widetilde{\mathbf{C}}$ can be exactly recovered. Therefore, by Claim 5.3 we can locate any column deletions with indices $n - r_w - (t+1)^2 < j \leq n$ by decoding $\widetilde{\mathbf{T}}^{(1)}$. Consequently, having the location of $\widetilde{\mathbf{L}}^{(1)}$ and using Claim 5.3, we can recover the indices of the deleted rows that satisfy $1 \leq i \leq t\ell$. Similarly, we can obtain the indices with $n - r_w - (t+1)^2 < i \leq n$ and $1 \leq j \leq t\ell$. □

Lemma 5.5. *Given the array $\widetilde{\mathbf{C}}$ affected by t_r row and t_c column deletions, any row index $i \in \mathcal{I}^{(t_\mathrm{r})}$ such that $t\ell < i \leq n - r_w - (t+1)^2$ can be exactly recovered. Similarly, any column index $j \in \mathcal{I}^{(t_\mathrm{c})}$ such that $t\ell < j \leq n - r_w - (t+1)^2$ can be exactly recovered.*

Proof. We start by proving that the column indices can be recovered. We want to leverage the structure imposed by the set $\mathcal{H}_t(\ell, n)$. For an array $\mathbf{C} \in \mathcal{H}_t(\ell, n)$, each row of the subarray $\mathbf{C}_{[1:t\ell],[t\ell+1:n-(t+1)^2]}$ is encoded using a binary systematic t-insdel-correcting code. In addition, the columns $\mathbf{C}_{[1:t\ell],j}$ such that $t\ell < j \leq n - r_w - (t+1)^2$ are the systematic part of this code. Recall that the rows are divided into t blocks, each of size ℓ, where in each block the columns $t < j \leq n - r_w - (t+1)^2$ satisfy the window constraint. We assume that at least one column in this interval is deleted. Therefore, at most $(t-1)$ rows can be deleted in $\mathbf{C}$. This means that at least one block of ℓ rows is unaffected by any row deletion. We can locate this block for the deletion case by Lemma 5.4. By Claim 5.5, we can recover the indices of the columns deleted within the range $t < j \leq n - r_w - (t+1)^2$. Similarly, we can obtain the indices with $t\ell < i \leq n - r_w - (t+1)^2$ by leveraging the structure imposed by $\mathcal{V}_t(\ell, n)$ using Claim 5.5. □

We now present the equivalent results for the insertion case. In the following statements we only highlight the differences from the deletion case. The main difference is that we must tackle possible insertion patterns that can create block confusions in our received array $\widetilde{\mathbf{C}}$. In general, the strategy is as follows. Since all rows and columns of the codeword $\mathbf{C}$ satisfy the window constraint, even after t-criss-cross insertions, we can exploit the fact that insertions can only create a confusion of length at most $2t$ with at most t rows/columns of the original array. Therefore, we can delete the block confusions and turn the insertion locating problem into a deletion locating problem, which we have shown earlier how to solve.

Claim 5.8. *Given the array $\widetilde{\mathbf{C}}$ affected by t_r row and t_c column insertions, the marker array $\widetilde{\mathbf{M}}^{(1,1)}$ can be located up to row and column block confusion of length at most $2t$, and $\widetilde{\mathbf{M}}^{(1,2)}$ can be located up to column block confusion of length at most $2t$ and row block confusion of length at most $2t+1$. The marker array $\widetilde{\mathbf{M}}^{(2,1)}$ can be located up to row and column block confusion of length at most $2t$, and $\widetilde{\mathbf{M}}^{(1,2)}$ can be located up to column block confusion of length at most $2t$ and row block confusion of length at most $2t+1$.*

Proof. We have the same preliminary information as in Claim 5.4 and 5.7. However, suppose the inserted rows/columns create block confusions exactly at the border of the arrays $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(1)}$, or $\mathbf{L}^{(2)}$ and $\mathbf{T}^{(2)}$. In that case, we can locate the desired arrays only up to the length of the block confusion. By construction, the length of the block confusion is limited to the parameter range given in the statement of the claim. Moreover, we use the following strategy for locating the border between $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(1)}$ in the presence of block confusions, where the same can be applied to the border of $\mathbf{L}^{(2)}$ and $\mathbf{T}^{(2)}$ by symmetry of the construction. The

idea is to use a block of insdel codes to resolve the row block confusion. Note that there must be at least one insdel code block that is not affected by insertions due to the existence of t insdel code subarrays. Moreover, we can determine the affected arrays by Claim 5.4. We declare the border of the insdel code to be the first expected row index of $\mathbf{T}^{(1)}$. Note that by declaring a wrong border, we add at most t insertions to the insdel code. By Claim 5.6, we can decode the insdel code. The correct border can then be determined by the first index, which is not marked as an insertion in the insdel code subarray. If no insertions are declared at the beginning of the subarray, then the chosen border is correct.

Moreover, in case $\widetilde{\mathbf{M}}^{(1,2)}$ and $\widetilde{\mathbf{M}}^{(2,2)}$ are located up to a column block confusion and a row block confusion, respectively, one can simply ignore the confusion to locate $\widetilde{\mathbf{M}}^{(1,1)}$ and $\widetilde{\mathbf{M}}^{(2,1)}$ since the inserted columns must follow the structure of $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(2)}$ to create a confusion. □

Observe that the marker arrays $\widetilde{\mathbf{M}}^{(1,2)}$ and $\widetilde{\mathbf{M}}^{(2,2)}$ can be located up to a row, or respectively a column block confusion of length at most $2t+1$. The confusion may contain more than t original rows/columns of the original array. However, since the indices of the block confusions are within the index range of the insdel codes, we can use those as stated in Lemma 5.7 to tackle this problem.

Lemma 5.6. *Given the array $\widetilde{\mathbf{C}}$ affected by t_{r} row and t_{c} column insertions, any row with index $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ such that $1 \leq i \leq t\ell$ or $n - r_w - (t+1)^2 < i \leq n$ can be recovered up to a row block confusion of length at most $2t$ consisting of at most t rows of the original array $\mathbf{C}$. Similarly, any column index $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ such that $1 \leq j \leq t\ell$ or $n - r_w - (t+1)^2 < j \leq n$ can be recovered up to a column block confusion of length at most $2t$ consisting of at most t columns of the original array $\mathbf{C}$.*

Proof. We focus on recovering the indices of the inserted rows such that $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ and the indices of the inserted columns such that $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ that satisfy $1 \leq i \leq t\ell$ and $n - r_w - (t+1)^2 < j \leq n$. Recovering the remaining indices of the statement follows by symmetry of the construction. In general, we can use the same strategy as presented for the deletion case using Claims 5.4 and 5.8. In case the marker arrays are located up to a block confusion, one can ignore the rows/columns that created a confusion, i.e., we only consider one collection of the rows/columns of the confusion since the inserted rows/columns must satisfy the fixed structure of the locator arrays $\mathbf{T}^{(1)}$, $\mathbf{L}^{(1)}$, $\mathbf{T}^{(2)}$, and $\mathbf{L}^{(2)}$. □

Lemma 5.7. *Given the array $\widetilde{\mathbf{C}}$ affected by t_{r} row and t_{c} column insertions, any row index $i \in \mathcal{I}^{(t_{\mathrm{r}})}$ such that $t\ell < i \leq n - r_w - (t+1)^2$ can be recovered up to a row block confusion of length at most $2t$ consisting of at most t rows of the original array $\mathbf{C}$. Similarly, any column index $j \in \mathcal{I}^{(t_{\mathrm{c}})}$ such that $t\ell < j \leq n - r_w - (t+1)^2$ can be recovered up to a column block confusion of length at most $2t$ consisting of at most t columns of the original array $\mathbf{C}$.*

Proof. We start by proving that the column indices can be recovered. We want to leverage the structure imposed by the set $\mathcal{H}_t(\ell, n)$. We assume that at least one column in this interval is inserted. This means that at least one block of ℓ rows is unaffected by any row insertion. By Lemma 5.6 we can locate this block by remarking that at least one insertion is needed to create a block confusion and therefore affect the block. By Claim 5.6, we can recover the

indices of the columns inserted within the range $t < j \leq n - r_w - (t+1)^2$ up to a block confusion of length at most $2t$ containing at most t columns of the original array. Similarly, we can obtain the indices with $t\ell < i \leq n - r_w - (t+1)^2$ by leveraging the structure imposed by $\mathcal{V}_t(\ell, n)$, see Lemma 5.6 and Claim 5.6. □

5.7.2 Recovering the Transmitted Array

Now, we can present the full proof of our code construction.

Proof of Theorem 5.4. If a t-criss-cross deletion happened, we can apply Lemmas 5.4 and 5.5 to determine the sets of indices $\mathcal{I}^{(t_\mathrm{r})}$ and $\mathcal{I}^{(t_\mathrm{c})}$. Then, for all $i \in \mathcal{I}^{(t_\mathrm{r})}$ and $j \in \mathcal{I}^{(t_\mathrm{c})}$, the decoder inserts row or column erasures in $\widetilde{\mathbf{C}}$ starting from the smallest index. Now, the decoder applies a Gabidulin criss-cross erasure decoder to determine the values of the erased symbols [GP08].

In case of a t-criss-cross insertion we apply Lemmas 5.6 and 5.7. The decoder deletes the inserted rows/columns whose positions are exactly recovered. For each block confusion, the decoder deletes the whole block confusion. This deletion strategy deletes at most t rows/columns of the original array since all rows/columns follow the window constraint. Thus, the decoder inserts row/column erasures in $\widetilde{\mathbf{C}}$ starting from the smallest index and applies a Gabidulin criss-cross erasure decoder to determine the values of the erased symbols [GP08]. □

5.8 Redundancy for Code Construction

In this section, we analyze the redundancy of our code presented in Section 5.6 denoted by $r^{\mathrm{cc}}(n,t)$. We will refer to the redundancy of each individual set $\mathcal{C}_{\mathrm{Gab}}(t,n)$, $\mathcal{L}_t(n)$, $\mathcal{H}_t(\ell,n)$, $\mathcal{V}_t(\ell,n)$, $\mathcal{E}_t(\ell,n)$, $\mathcal{W}_t(\ell,w)$ and $\mathcal{M}_t(\ell,n)$ by $r_*(n,t)$, where $*$ is replaced with the corresponding set letter. In the following, we give an intuition behind the computations of the redundancy.

Since $\mathcal{C}^{\mathrm{cc}}_{t,n} = \mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(t,n)$ and since the Gabidulin code is a linear code, we can compute the code redundancy as

$$r^{\mathrm{cc}}(n,t) \leq r_{\mathcal{L}}(n,t) + r_{\mathrm{G}}(n,t)\,.$$

Moreover, since the intersected sets in the locator set $\mathcal{L}_t(n)$ impose constraints on disjoint positions in the $n \times n$ arrays, we can further split the redundancy as follows.

$$r_{\mathcal{L}}(n,t) = r_{\mathcal{H}}(n,t) + r_{\mathcal{V}}(n,t) + r_{\mathcal{E}}(n,t) + r_{\mathcal{M}}(n,t)\,.$$

The sets $\mathcal{H}_t(\ell,n)$ and $\mathcal{V}_t(\ell,n)$ impose similar constraints: t disjoint subarrays constrained with the window constraint where each row is protected by a systematic t-insdel correcting code from [SGB20b].

Claim 5.9. *The redundancy resulting from the constraints imposed by the two sets $\mathcal{H}_t(\ell, n)$ and $\mathcal{V}_t(\ell, n)$ is bounded as*

$$r_{\mathcal{H}}(n,t) + r_{\mathcal{V}}(n,t) \leq 2t(r_{\mathcal{W}}(\ell, w) + \log_2(n) \cdot r_w)\,,$$

where $w = n - t\log_2(n) - r_w - (t+1)^2$ and $r_w \leq 4t\log_2(n) + o(\log_2(n))$.

Proof. The sets $\mathcal{H}_t(\ell, n)$ and $\mathcal{V}_t(\ell, n)$ impose the same constraints, i.e., each array belonging to any of these sets has t subarrays protected by deletion correcting codes with window constraints. Therefore, we have

$$r_{\mathcal{H}}(n,t) + r_{\mathcal{V}}(n,t) = 2t(r_{\mathcal{W}}(\ell, w) + \log_2(n) \cdot r_w)\,,$$

where r_w is the length of the redundancy vector used to protect a vector of length

$$w = n - t\log_2(n) - r_w - (t+1)^2 \tag{5.4}$$

against t deletions, and $\log_2(n)$ is the number of protected vectors in each subarray. Recall that for any positive integer k, the redundancy for protecting a vector of length k is bounded by $r_k \leq 4t\log_2(k) + o(\log_2(k))$ [SGB20b]. Thus, we bound this value by

$$r_w \leq (4t+1)\log_2(n)\,. \tag{5.5}$$

We now focus on computing $r_{\mathcal{W}}(\ell, w)$. To compute an upper bound on the redundancy imposed by the window constraint $\mathcal{W}_t(\ell, w)$, we require a lower bound on w. Note that using a lower bound on w only increases the redundancy imposed by $\mathcal{W}_t(\ell, w)$. This will be clear from the following calculations. From Equations (5.4) and (5.5) we obtain

$$w \geq n - (5t+1)\log_2(n) - (t+1)^2\,.$$

We calculate a lower bound on $|\mathcal{W}_t(\ell, w)|$. On a high level, our calculations are interpreted as going through each column of an array in $\mathcal{W}_t(\ell, w)$ and counting the number of choices for this specific column. The first column is arbitrary and thus has 2^ℓ choices. The second column is not allowed to be the same as the one before. Thus, it has $(2^\ell - 1)$ choices. The third column has $(2^\ell - 2)$ choices since it cannot be the same as the two preceding columns. This process continues until we reach the $(t+2)^{\text{nd}}$ column. The number of choices for this vector is $(2^\ell - (t+1))$. Since the restriction is imposed on an interval of $(t+1)$ vectors, each remaining column has $(2^\ell - (t+1))$ choices. Thus, for the window constraint, the following holds.

$$\begin{aligned}
|\mathcal{W}_t(\ell, w)| &\geq 2^\ell \cdot (2^\ell - 1) \cdot \ldots \cdot (2^\ell - (t+1)) \cdot (2^\ell - (t+1))^{w-t-2} \\
&\geq (2^\ell - (t+1))^w \\
&= 2^{\ell w}\left(1 - \frac{t+1}{2^\ell}\right)^w\,.
\end{aligned}$$

We denote the redundancy resulting from the constraints imposed by the window constraint

as $r_{\mathcal{W}}(\ell, n - t\ell - r_w)$. We continue the calculations, recalling that $\ell = \log_2(n)$.

$$\begin{aligned}
r_{\mathcal{W}}(\ell, n - t\ell - r_w) &\leq \ell(n - t\ell - r_w) - \log_2\left(|\mathcal{W}_t(\ell, n - t\ell - r_w)|\right) \\
&\leq \log_2\left(\left(1 - \frac{t+1}{2^\ell}\right)^{n-t\ell-r_w}\right) \\
&\leq \log_2\left(\left(1 - \frac{t+1}{n}\right)^{n}\right) - \log_2\left(\left(1 - \frac{t+1}{n}\right)^{(5t+1)\log_2(n)}\right) \\
&\overset{\text{(d)}}{\leq} \log_2(e^{(t+1)}) - (5t+1)\log_2(n) \cdot \log_2\left(\left(1 - \frac{t+1}{n}\right)\right) \\
&\overset{\text{(e)}}{\leq} (t+1)\log_2(e) + (5t+1)\log_2(n) \\
&\leq (5t+1)\log_2(n) + 2(t+1)\,.
\end{aligned}$$

We used in Step (d) the inequality $(1 - \frac{x}{n})^n \leq e^x$ and exploited in Step (e) the fact that $\frac{1}{2} \leq (1 - \frac{t+1}{n}) \leq 1$ for our choice of parameters and for sufficiently large n.

Recall that any array in $\mathcal{D}_t^{(1)}(\ell, n - t\ell - (t+1)^2)$ consists of $\log_2(n)$ binary t-deletion correcting codes. Therefore, we have

$$\begin{aligned}
r_{\mathcal{H}}(n,t) &\leq t \cdot \left(\underbrace{(5t+1)\log_2(n) + 2(t+1)}_{\text{window constraint}} + \underbrace{\log_2(n) \cdot (4t+1)\log_2(n)}_{\text{binary deletion correcting codes}}\right) \\
&= (4t^2 + t)\log^2(n) + (5t^2 + t)\log_2(n) + 2t(t+1)^2\,.
\end{aligned}$$

Since the arrays in $\mathcal{V}_t(\ell, n)$ have a similar structure imposed (only transposed) and the regions of the imposed constraints are disjoint, one can conclude that

$$r_{\mathcal{H}}(n,t) + r_{\mathcal{V}}(n,t) \leq 2(4t^2 + t)\log^2(n) + 2(5t^2 + t)\log_2(n) + 4t(t+1)^2\,.$$

□

Observe that the constraints for the remaining sets fix values for certain subarray boundaries. Therefore, the following can be obtained.

Claim 5.10. *The redundancy $r_{\mathcal{L}}(n,t)$ resulting from the constraints imposed by the set $\mathcal{L}_t(n)$ is bounded as*

$$r_{\mathcal{L}}(n,t) \leq (8t^2 + 2t)\log^2(n) + o(\log^2(n))\,.$$

Proof. We argued that since the different constraints are imposed on disjoint subarrays in $\mathcal{L}_t(n)$, then the redundancy $r_{\mathcal{L}}(n,t)$ can be written as

$$r_{\mathcal{L}}(n,t) = r_{\mathcal{H}}(n,t) + r_{\mathcal{V}}(n,t) + r_{\mathcal{E}}(n,t) + r_{\mathcal{M}}(n,t)$$

The redundancy imposed by the locator and marker arrays equals the dimension of the

subarrays with fixed entries. We can then write

$$r_{\mathcal{E}}(n,t) \leq (6t^3 + 13t^2 + 8t + 1)\log_2(n)$$

and

$$r_{\mathcal{M}}(n,t) = 4(t+1)^2\,.$$

The other terms of the redundancy in $r_{\mathcal{L}}(n,t)$ were computed in Claim 5.9. □

We can conclude this section with the statement on the redundancy $r^{\mathrm{cc}}(n,t)$ of the code $\mathcal{C}^{\mathrm{cc}}_{t,n}$ presented in Construction 5.2. Note that the redundancy added by the Gabidulin code is tn.

Lemma 5.8. *The redundancy of the code $\mathcal{C}^{\mathrm{cc}}_{t,n}$ is bounded as*

$$r^{\mathrm{cc}}(n,t) \leq tn + (8t^2 + 2t)\log^2(n) + o(\log^2(n))\,.$$

Proof. By construction we have that $\mathcal{C}^{\mathrm{cc}}_{t,n} = \mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(n,t)$. By Claim 5.10 we have that

$$|\mathcal{L}_t(n)| \geq \frac{2^{n^2}}{n^{((8t^2+2t)+o(1))\log_2(n)}}.$$

From [GP08] we have that $|\mathcal{C}_{\mathrm{Gab}}(n,t)| = \frac{2^{n^2}}{2^{tn}}$. Further, because $\mathcal{C}_{\mathrm{Gab}}(n,t)$ is a linear code, a coset exists such that the following is satisfied by the pigeonhole principle.

$$|\mathcal{C}^{\mathrm{cc}}_{t,n}| \geq 2^{n^2} \cdot \underbrace{\frac{1}{2^{tn}}}_{\text{Gabidulin Code}} \cdot \underbrace{\frac{1}{n^{((8t^2+2t)+o(1))\log_2(n)}}}_{\text{Locator Set}}.$$

Hence, we can conclude that the total redundancy of the $\mathcal{C}^{\mathrm{cc}}_{t,n}$ satisfies

$$r^{\mathrm{cc}}(n,t) = n^2 - \log_2(|\mathcal{C}^{\mathrm{cc}}_{t,n}|) \leq tn + (8t^2 + 2t)\log^2(n) + o(\log^2(n)).$$

□

5.9 Conclusion

In this chapter, we have considered the t-criss-cross insertion/deletion problem in binary arrays. First, we have shown that the one-dimensional insertion-deletion equivalence also holds in the two-dimensional array setting. Moreover, we have shown that the asymptotic lower bound on the redundancy for any t-criss-cross correcting code is $r^{\mathrm{cc}}(n,t) \geq tn + t\log(n) - \mathcal{O}(1)$. First, we have presented a construction for a simplified channel model of a fixed $(1,1)$-criss-cross insertion/deletion error. Subsequently, we have presented our t-criss-cross insertion/deletion correcting code construction, which is based on transforming the insertion/deletion problem into an erasure problem. The redundancy of the constructed t-criss-cross insertion/deletion correcting code is $\mathcal{O}(t^2\log^2(n))$ far from the derived lower

bound. The construction is proven by providing an explicit decoder. We note that given an order-optimal systematic construction of n-ary t-insdel correcting codes, we could improve our construction such that the redundancy is only $\mathcal{O}(t^3 \log(n))$ far from the derived lower bound. This results from replacing the $2t \log n$ binary t-insdel correcting codes indexing the columns/rows by $2t$ n-ary t-insdel correcting codes.

On a final note, improvements on the coding problem for insdel errors in arrays remain possible. We generalized in [SWB+22] the one-dimensional equivalence between deletions and insdels errors to arrays and even extended the equivalences to hold for any d-dimensional arrays. Moreover, [CHK21] presents an optimal construction for the simplified model and provides an alternate construction for the t-criss-cross insertion/deletion problem with redundancy $tn + o(n)$. Further, it would be interesting to generalize the problem to study possible combinations of simultaneous insertions and deletions in arrays. The next step would be finding a code construction, with redundancy close to the lower bound derived in this work, that can correct a mixture of row and column insdel errors. One step in this direction was given in [Hag23], where a construction correcting combinations of row/column deletions, substitutions, and erasures in (not necessarily squared) arrays is presented. Further research topics in this direction include studying the characteristics of an array's insdel spheres.

Part II

Coding Schemes for the DNA Storage Channel with a Probabilistic Decoder

Chapter 6

Preliminaries on Insertion/Deletion Error-Correction for Stochastic Channels

This part of the dissertation covers end-to-end coding solutions for DNA-based data storage by assuming stochastic transmission channel models and using probabilistic decoding methods, especially in the presence of insertion, deletion, and substitution errors. Consequently, in this chapter, we first cover essentials introducing the applied methodologies. First, we briefly cover the applied probabilistic decoding principles [Mac03; RL09; CT06]. Subsequently, we focus on two channels: the independently identically distributed (i.i.d.) insertion, deletion, and substitution (IDS) channel (see Section 6.2.2); and the trace-reconstruction channel with IDS errors (see Section 6.2.3).

In the probabilistic scenario, we describe the channel procedure as a stochastic process with probabilistic inference on the receiver's side. In the following, we let input and output formally be described by the random vectors $\mathbf{X}$ and $\mathbf{Y}$, respectively. Further, the channel is described by the general channel law $\mathbb{P}(\mathbf{Y}|\mathbf{X})$. The decoder aims to minimize the failure probability of deciding on the incorrect codeword. This is equivalent to maximizing the *a posteriori* probability (APPs); hence, a maximum a posteriori (MAP) decoder decides as

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{C}} \mathbb{P}(\mathbf{x} \mid \mathbf{y}) = \arg\max_{\mathbf{x}\in\mathcal{C}} \frac{\mathbb{P}(\mathbf{y},\mathbf{x})}{\mathbb{P}(\mathbf{y})} \propto \arg\max_{\mathbf{x}\in\mathcal{C}} \mathbb{P}(\mathbf{y} \mid \mathbf{x})\,\mathbb{P}(\mathbf{x}) \ .$$

For uniform input, this decoding rule is equivalent to the *maximum likelihood* (ML) decoding rule given as

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{C}} \mathbb{P}(\mathbf{y} \mid \mathbf{x}) \ .$$

Another decoding approach is the symbolwise MAP (s-MAP) decoder which decides symbol by symbol according to

$$\hat{x}_t = \arg\max_{x_t\,:\,\mathbf{x}\in\mathcal{C}} \mathbb{P}(x_t \mid \mathbf{y}) \ .$$

Popular decoders facilitating the s-MAP rule are the sum-product algorithms based on message-passing on factor graphs [KFL01] also known as the forward-backward or BCJR

algorithm [BCJ+74; HOP96], e.g., for trellis codes, and belief-propagation, e.g., for low-density-parity-check codes [Gal62].

Given a channel with $\mathbb{P}(\mathbf{Y}|\mathbf{X})$, we may wonder what is the highest coding rate R at which one can ensure reliable transmission.

Definition 6.1 ([SH22; CT06])**.** *A rate R is called achievable if there exists a sequence of codes $\{\mathcal{C}_n\}$, each of length n and of rate R, such that the decoding error probability tends to 0 as $n \to \infty$. The capacity is the supremum of the achievable rates.*

Another important quantity is the *mutual information rate* (MIR) which is expressed as

$$\frac{1}{n}\mathrm{I}(\mathbf{X};\mathbf{Y}) \ .$$

For various channels, this quantity is an achievable rate for $n \to \infty$, e.g., see Section 6.2.2. Further, in cases the MIR is infeasible to compute a lower bound may be derived as described in Section 6.1.

In a finite-length setting, a popular figure of merit for a given coding system is the average *frame-error-rate* (FER) p_{fer} which is approximated using Monte-Carlo methods. We sample a large number of times Φ realizations $\mathbf{x}$ from the input space and pass it via the channel to obtain a realization $\mathbf{y}$ over the output space, and letting a decoder infer $\hat{\mathbf{x}}$ given $\mathbf{y}$. Thus, we compute

$$p_{\mathrm{fer}} \triangleq \frac{1}{\Phi}\sum_{\phi=1}^{\Phi}\mathbb{P}(\hat{\mathbf{x}}_\phi \neq \mathbf{x}_\phi|\mathbf{x}_\phi) \ .$$

6.1 Mismatched Decoding and the BCJR-once Rate

In various scenarios, it may be that the true channel law $\mathbb{P}(\mathbf{Y}|\mathbf{X})$ is unknown, the maximizing input distribution $\mathbb{P}(\mathbf{X})$ is unknown, or simply the optimal MAP/ML/s-MAP decoding rule is strictly too complex to implement. The concept of *mismatched decoding* allows us to quantify achievable rates for decoders which follow an auxiliary channel law $\mathbb{Q}(\mathbf{Y}|\mathbf{X})$ instead of the true model $\mathbb{P}(\mathbf{Y}|\mathbf{X})$ [SFSB+20; KS93; MKL+94]. In the following, we focus on simplified decoders and present the so-called *BCJR-once rate*, denoted as $R_{\text{BCJR-once}}$ [KMM03; MG04; SPS07] which is defined as the symbolwise mutual information between the input process with uniform distribution and the output of an s-MAP decoder (e.g, the BCJR algorithm [BCJ+74], see Section 6.2.2). Note that the BCJR-once rate is closely related to the concept of *Generalized Mutual Information* introduced by [KS93; MKL+94].

We assume an s-MAP decoder to output non-negative symbolwise APPs $\mathbb{Q}(\mathsf{X}_t|\mathbf{y})$ such that $\sum_{x_t}\mathbb{Q}(x_t|\mathbf{y}) = 1$ which may not be the true APPs $\mathbb{P}(\mathsf{X}_t|\mathbf{y})$. We may mitigate under- or overconfidence of the s-MAP output by an order-preserving transformation of the symbolwise APPs such that $\mathbb{Q}(\mathsf{X}_t|\mathbf{y}) = \mathbb{Q}(\mathsf{X}_t|\mathbf{y})^s$ for $s > 0$. Further, let $\mathbb{Q}(\mathbf{X}|\mathbf{Y})$ be an appropriate decoding metric. More precisely, we consider a memoryless metric given by

$$\mathbb{Q}(\mathbf{x}|\mathbf{y}) \propto \prod_{t=1}^{n}\mathbb{Q}(x_t|\mathbf{y})^s \ ,$$

where the proportional constant is chosen such that $\sum_{\mathbf{x}} \mathbb{Q}(\mathbf{x}|\mathbf{y}) = 1$ for all $\mathbf{y}$. Consequently, for the mutual information $\mathrm{I}(\mathbf{X};\mathbf{Y})$, it holds that

$$\begin{aligned}
\mathrm{I}(\mathbf{X};\mathbf{Y}) &= \mathrm{H}(\mathbf{X}) - \mathrm{H}(\mathbf{X}|\mathbf{Y}) \\
&= \mathbb{E}_{(\mathbf{X},\mathbf{Y})}\left[-\log_2 \mathbb{P}(\mathbf{x}) + \log_2 \mathbb{P}(\mathbf{x}|\mathbf{y})\right] \\
&= \mathbb{E}_{(\mathbf{X},\mathbf{Y})}\left[-\log_2 \mathbb{P}(\mathbf{x}) + \left(\log_2 \mathbb{Q}(\mathbf{x}|\mathbf{y}) + \log_2 \frac{\mathbb{P}(\mathbf{x}|\mathbf{y})}{\mathbb{Q}(\mathbf{x}|\mathbf{y})}\right)\right] \\
&\geq \mathbb{E}_{(\mathbf{X},\mathbf{Y})}\left[-\log_2 \mathbb{P}(\mathbf{x}) + \log_2 \mathbb{Q}(\mathbf{x}|\mathbf{y})\right] , \\
&= \mathbb{E}_{(\mathbf{X},\mathbf{Y})}\left[\sum_{t=1}^{n} \left(-\log_2 \mathbb{P}(x_t) + \log_2 \mathbb{Q}(x_t|\mathbf{y})^s\right)\right] ,
\end{aligned}$$

where the expectation is over the true joint distribution $\mathbb{P}(\mathbf{X},\mathbf{Y})$ of the RVs. The inequality holds by identifying that the third summand corresponds to a non-negative Kullback-Leibler divergence, which can be interpreted as the *mismatch*. The last equality holds due to the assumed memoryless metric and identical uniform input distribution. Assuming that the above expectation obeys ergodicity, we formally define the BCJR-once rate as follows.

Definition 6.2. *Given an identical uniformly input distribution and let $\mathbb{Q}(x_t|\mathbf{y})$ be the symbolwise APP output of a s-MAP decoder, the BCJR-once information rate $R_{BCJR\text{-}once}$, measured in bits per channel use, is defined as*

$$R_{BCJR\text{-}once} \triangleq \max_{s>0} \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} \left(-\log_2 \mathbb{P}(x_t) + \log_2 \mathbb{Q}(x_t|\mathbf{y})^s\right) .$$

An achievability result for $R_{\text{BCJR-once}}$ is shown for example in [SPS07]. The rate can be numerically approximated by sampling a long input sequence $\mathbf{x}$ and its corresponding output sequences $\mathbf{y}$ due to the Shannon–McMillan–Breiman theorem [CT06].

6.2 Channels with Random Synchronization Errors

In this section, we introduce two stochastic channel models which induce insertion, deletion, and substitution noise. First, we give a literature review on channels affected by synchronization errors for single and multiple sequence transmission (Section 6.2.1). Then, we introduce the *independently identically distributed (i.i.d.) insertion, deletion, and substitution (IDS) channel*, which is a single sequence transmission channel (Section 6.2.2). Subsequently, we expand this model to multiple sequences, called the *trace reconstruction channel with i.i.d. IDS noise* (Section 6.2.3). More precisely, a single input sequence is transmitted over multiple independent i.i.d. IDS channels, each producing an output sequence, that can be leveraged jointly by a decoder to infer the transmitted input.

6.2.1 Review on Coding for Synchronization Errors for Single and Multiple Traces

The studies of probabilistic channels with synchronization errors date back as early as 1961. In [Gal61], Gallagher discussed a channel in which each channel input symbol independently gets deleted, transmitted correctly, or replaced by two random symbols. He proposed to use convolutional codes, offset by a pseudo-random sequence that is known at the receiver's side to enable better synchronization, which is decoded using sequential decoding techniques. The same has been analyzed in [Zig69] for a similar channel. In 1967, Dobrushin proved Shannon's capacity theorem which he showed for memoryless channels for a big variety of synchronization channels [Dob67], which later has been independently extended by [Koz71; AW71]. Bahl and Jelinek modeled the synchronization channel as a finite-state machine and presented inference mechanisms using stack algorithm in [BJ75]. Building upon Gallagher's approach, convolutional coding schemes to tackle synchronization errors have been discussed in [BFC00; CFS06; MT10; BF15] and extended for turbo codes in [MT12].

A break-through work of Davey and MacKay [DM01] in 2001 proposed a concatenated coding scheme based on LDPC codes and so-called watermark codes as inner codes, which are sparse codebooks that are offset by a pseudo-random sequence. The inner code is responsible for maintaining synchronization, whereas the outer code corrects residual errors. Moreover, they modeled the synchronization channel as a hidden Markov model (HMM) and derived forward-backward recursions to infer symbolwise APPs even in the presence of insertion and deletion errors. This scheme has been subject of several improvements and optimizations in the 2010s [BSW10; BB11; JA11; WFD11; BBW14; SHY20]. Most-noteworthy is the symbol-level decoding strategy [BSW10] where trellis sections of the joint code and channel trellis are combined consisting of multiple consecutive channel symbols (e.g., the length of an inner codeword) and likelihoods are computed using the metric from [BJ75]. Inspired by Seller's marker codes from the 1960s [Sel62], concatenated coding schemes similar to the Davey-MacKay scheme have been proposed in [CMC$^+$03; Rat05; WFD10], where fixed (or adaptive [IK12]) markers are inserted after a fixed number of outer code word symbols. In these schemes, synchronization is also achieved by a forward-backward algorithm with an HMM. The work [Rat05] was the first to employ turbo iterations between the outer and inner marker codes to gradually improve synchronization performance with the help of the outer code. Deep-learning approaches for these types of concatenated schemes have been studied in [MJM$^+$24]. Standalone codes (i.e., without an inner code) such as LDPC codes in [GK18; SHY19] and polar codes in [KK20; TFV21; TPF$^+$21; PT21] were studied to correct synchronization errors. We again refer to [MBT10] for an elaborative survey on synchronization channels.

The problem of inferring the original sequence given multiple noisy versions arose in the field of computational biology as the *multiple sequence alignment problem* [CL88]. It was first mathematically introduced by [Lev01] in the adversarial setting as the *sequence reconstruction problem* and shortly after by [BKK$^+$04] as the *trace reconstruction problem* in the probabilistic setting. These and numerous follow-up works – we focus on the probabilistic setting – aim to characterize the minimum number of traces needed to recover the original sequences in the worst-case, the average-case, and with coded input. We refer to [BPR$^+$20] for a nice overview. We focus on the scenario for approximate inference with a relatively small

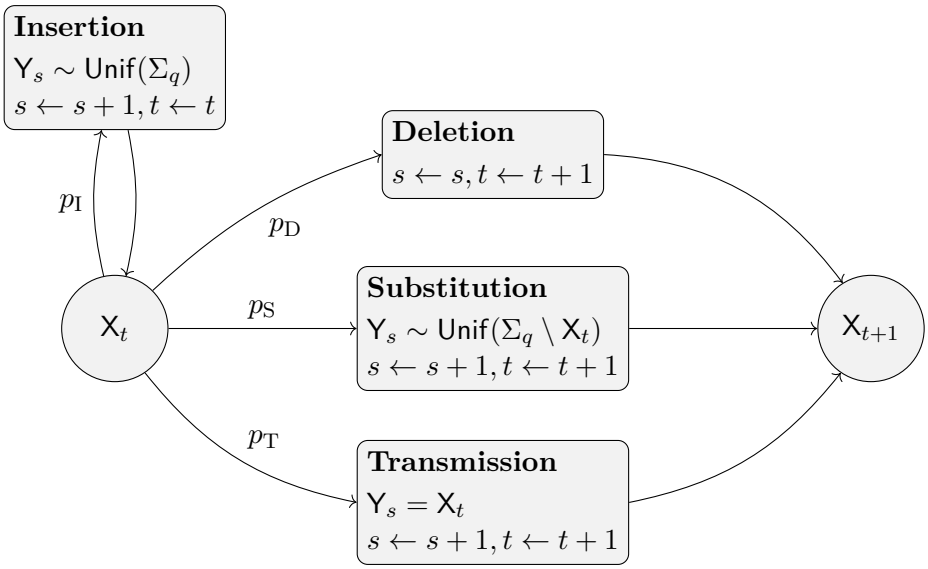


Figure 6.1: State-based IDS channel model [BJ75].

budget of traces, i.e., roughly only $2-35$ traces (surely with a constant number of traces with respect to the sequence length), using a practical algorithm. In the field of biology, in this use case, MSA algorithms have been excessively applied [CG17] following a consensus decision by making a majority decision on the produced alignment. The work of [SYS+24] reconstructs the original by using dynamic programming. In a more information- and coding-theoretic fashion, approximate ML solutions have been studied in [SYY20; SDD+18]. [SK20] derived an s-MAP decoder, which is an extension of the HMM approach from the single sequence case of [DM01] to a multi-dimensional HMM. However, decoding using this HMM is practically infeasible due to an exponential growth of the state space according to the number of traces. Building upon this (and concurrently proposed), coded versions have been studied in [MLW+22] and [SGP+21], where both works consider a concatenated coding scheme setting, where the inner code is incorporated in the multi-dimensional HMM. The inner codes were inspired in the single sequence case by convolutional and time-varying codes from [BF15; BBW14] and [Rat05], respectively. Both of the works, [MLW+22] and [SGP+21], proposed similar suboptimal decoders with linear complexity (with respect to the number of traces), which showed competitive performance with the MSA algorithms and the algorithm from [SYS+24] in the regime of $2-10$ traces. In the same line of thought to reduce the decoding complexity, sequential decoding techniques on the multi-dimensional HMM trellis have been investigated in [BLWZ24; BWA+24].

6.2.2 I.i.d. Insertion/Deletion/Substitution Channel

We introduce the i.i.d. IDS channel as a finite-state machine model in line with [BJ75]. In this section, we follow closely the descriptions from [MLW+22; SGP+21].

Let the vector $\mathbf{X} = (\mathsf{X}_1, \ldots, \mathsf{X}_n)$ be the input and the vector $\mathbf{Y} = (\mathsf{Y}_1, \ldots, \mathsf{Y}_{n'})$ be the output of the channel, both with symbols from Σ_q. We define four non-negative numbers

$p_\mathrm{I}, p_\mathrm{D}, p_\mathrm{S}$, and p_T to be the insertion, deletion, substitution, and (correct) transmission probability, respectively, such that $p_\mathrm{I} + p_\mathrm{D} + p_\mathrm{S} + p_\mathrm{T} = 1$. For an input $\mathbf{X}$, we generate the output $\mathbf{Y}$ symbol by symbol. Let t and s be the integer pointing to the current input and output symbols, respectively, initialized as $t = s = 1$. Then, we successively sample from the following four events:

1. **Insertion with probability** p_I
   A random symbol gets inserted. Thus, the symbol Y_s is chosen uniformly at random from Σ_q. We increase the pointer $s \leftarrow s + 1$, whereas t remains unchanged.

2. **Deletion with probability** p_D
   The symbol X_t is deleted. Hence, we set the pointers to $t \leftarrow t + 1$, and s remains unchanged.

3. **Substitution with probability** p_S
   The current input X_t gets substituted. Thus, the symbol Y_s is chosen uniformly at random from $\Sigma_q \setminus \{\mathsf{X}_t\}$, and we increase both pointers as $t \leftarrow t + 1$ and $s \leftarrow s + 1$.

4. **(Correct) Transmission with probability** p_T
   The symbol X_t gets transmitted and we set $\mathsf{X}_t = \mathsf{Y}_s$. Like in the substitution event, we increase both pointers as $t \leftarrow t + 1$ and $s \leftarrow s + 1$.

We stop when $t = n + 1$ and $s = n' + 1$. Note that n' is random and depends on the channel realization.[1] As can be seen by the different pointers t and s, the symbols in $\mathbf{Y}$ are not synchronized anymore, hence, Y_t could be the result of transmitting a symbol $\mathsf{X}_{t'}$ with $t' \neq t$. We can interpret the synchronization loss as infinite memory introduced by the channel. Nonetheless, we can describe the channel as an HMM by introducing the RV *drift* D_t [DM01], defined as the number of insertions minus the number of deletions that occurred at time t, as a hidden state variable. Using the input and output pointers, we have that $d_t = s - t$ for a given channel realization. In line with this thought, the output vector $\mathbf{Y}$ requires a different interpretation, i.e., $\mathbf{Y}$ is a concatenation of n vectors $\mathbf{Y}_t \in \left\{\Sigma_q^* \cup \{\}\right\}$ such that $\mathbf{Y} = (\mathbf{Y}_1 \parallel \ldots \parallel \mathbf{Y}_n)$. For each symbol X_t, a string $\mathbf{Y}_t$ of length $\mathsf{D}_t - \mathsf{D}_{t-1} + 1$ is formed according to the channel probabilities and state transitions for the fixed pointer t. The essence of the drift states is that the sequence $\mathbf{D} = (\mathsf{D}_0, \ldots, \mathsf{D}_n) \in \mathbb{Z}^{n+1}$ forms a Markov chain, and the outputs after time $t + 1$ are independent of the previous states, enabling the Markov property:

$$\mathbb{P}(\mathbf{X}, \mathbf{Y}, \mathbf{D}) = \mathbb{P}(\mathsf{D}_0) \prod_{t=1}^{n} \mathbb{P}(\mathbf{Y}_t, \mathsf{D}_t \mid \mathsf{X}_t, \mathsf{D}_{t-1}) \, \mathbb{P}(\mathsf{X}_t) \ . \tag{6.1}$$

We have that $\mathbb{P}(\mathsf{D}_0 = 0) = 1$, $\mathbb{P}(\mathsf{X}_t)$ is given by the input distribution, and $\mathbb{P}(\mathbf{Y}_t, \mathsf{D}_t \mid \mathsf{X}_t, \mathsf{D}_{t-1})$ is the channel law with the hidden variable. We can describe the HMM by a factor graph as depicted in Figure 6.2.

This fact enables the formulation of forward-backward recursions for a BCJR decoder [BCJ+74; DM01; BF15]. For a given transmission, a BCJR decoder infers symbolwise APPs

[1] For ease of notation, we do not write n' as an RV.

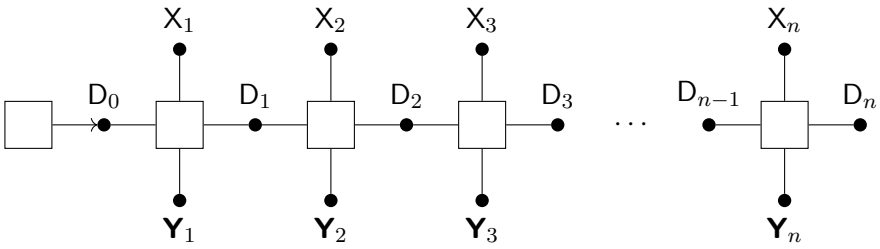


Figure 6.2: Factor graph of the i.i.d. IDS channel when described as an HMM. The black dots represent variable nodes, and the squares are the factor nodes, which are defined by the transition probabilities $\mathbb{P}\left(\mathsf{D}_t, \mathsf{Y}_t | \mathsf{X}_t, \mathsf{D}_{t-1}\right)$. For simplicity, the factors for the symbol input distribution $\mathbb{P}\left(\mathsf{X}_t\right)$ are not illustrated.

as

$$\mathbb{P}\left(x_t|\mathbf{y}\right) = \frac{\mathbb{P}\left(x_t, \mathbf{y}\right)}{\mathbb{P}\left(\mathbf{y}\right)} \propto \sum_{d_{t-1}, d_t \in \mathbb{Z}^{n+1}} \mathbb{P}\left(x_t, \mathbf{y}, d_{t-1}, d_t\right) .$$

Further, we can split the joint probability due to the Markov property as

$$\mathbb{P}\left(x_t, \mathbf{y}, d_{t-1}, d_t\right) = \underbrace{\mathbb{P}\left(\mathbf{y}_1^{(t-1)+d_{t-1}}, d_{t-1}\right)}_{\triangleq \alpha_{t-1}(d_{t-1})} \cdot \underbrace{\mathbb{P}\left(\mathbf{y}_{(t-1)+d_{t-1}+1}^{t+d_t}, x_t, d_t \middle| d_{t-1}\right)}_{\triangleq \gamma_t(d_{t-1}, d_t, x_t)} \cdot \underbrace{\mathbb{P}\left(\mathbf{y}_{t+d_t+1}^{n'} \middle| d_t\right)}_{\triangleq \beta_t(d_t)} .$$

We define the terms forward metric $\alpha_{t-1}(d_{t-1})$, backward metric $\beta_t(d_t)$, and branch metric $\gamma_t(d_{t-1}, d_t, x_t)$ as above. The forward and backward metric can be calculated recursively:

$$\alpha_t(d_t) = \sum_{d_{t-1}} \sum_{x_t} \alpha_{t-1}(d_{t-1}) \gamma_t(d_{t-1}, d_t, x_t) , \qquad \alpha_0(d_0) = \begin{cases} 1 & , d_0 = 0 , \\ 0 & , d_0 \neq 0 ; \end{cases}$$

$$\beta_{t-1}(d_{t-1}) = \sum_{d_t} \sum_{x_t} \gamma_t(d_{t-1}, d_t, x_t) \beta_t(d_t) , \qquad \beta_n(d_n) = \begin{cases} 1 & , d_n = n' - n , \\ 0 & , d_n \neq n' - n . \end{cases}$$

The branch metric can be decomposed as

$$\gamma_t(d_{t-1}, d_t, x_t) = \mathbb{P}\left(\mathbf{y}_{(t-1)+d_{t-1}+1}^{t+d_t}, d_t | d_{t-1}, x_t\right) \mathbb{P}\left(x_t\right) ,$$

The first term can be calculated using the channel probabilities. Let $\Delta_d \triangleq d_t - d_{t-1}$, $\dot{\mathbf{y}} \triangleq \mathbf{y}_{(t-1)+d_{t-1}+1}^{t+d_t}$, we have

$$\mathbb{P}\left(\dot{\mathbf{y}}, d_t | d_{t-1}, x_t\right) = \begin{cases} p_{\mathrm{D}} & , \Delta_d = -1 , \\ \left(\frac{p_{\mathrm{I}}}{q}\right)^{\Delta_d} \left(\frac{p_{\mathrm{I}}}{q} p_{\mathrm{D}} + p_{\mathrm{T}}\right) & , \Delta_d \geq 0, x_t = \dot{y}_{\Delta_d + 1} , \\ \left(\frac{p_{\mathrm{I}}}{q}\right)^{\Delta_d} \left(\frac{p_{\mathrm{I}}}{q} p_{\mathrm{D}} + \frac{p_{\mathrm{S}}}{q-1}\right) & , \Delta_d \geq 0, x_t \neq \dot{y}_{\Delta_d + 1} . \end{cases} \tag{6.2}$$

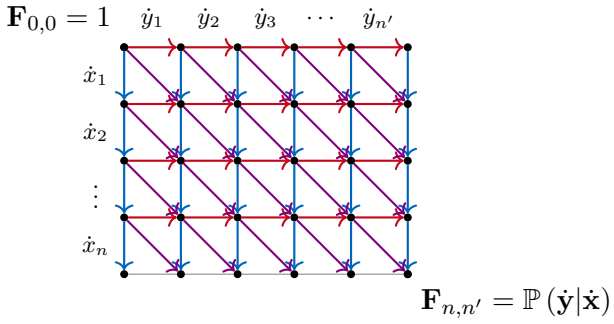


Figure 6.3: Lattice structure of $\mathbf{F}$ to compute $\mathbb{P}(\dot{\mathbf{y}}|\dot{\mathbf{x}})$ for the i.i.d. IDS channel. Assuming we are at vertex $\mathbf{F}_{\iota,\mu}$: The horizontal edge ($\rightarrow$) models an insertion with probability p_{I}/q; The vertical edge ($\rightarrow$) models a deletion with probability p_{D}; A diagonal edge ($\rightarrow$) models either a substitution with probability $p_{\mathrm{S}}/_{q-1}$ when $\dot{x}_\iota \neq \dot{y}_\mu$ or transmission with probability p_{T} whenever $\dot{x}_\iota = \dot{y}_\mu$.

Note that this calculation can be extended to compute vector likelihoods $\mathbb{P}(\dot{\mathbf{y}}, d_t|d_{t-1}, \mathbf{x}_t)$ using the lattice computation from [BJ75], which is the probabilistic expansion of the dynamic programming approach to compute the edit distance between two vectors [Vin68; WF74; NW70]. Assume a vector $\dot{\mathbf{x}} \in \Sigma_q^n$ is transmitted over the i.i.d. IDS channel with output $\dot{\mathbf{y}} \in \Sigma_q^{n'}$, we can compute $\mathbb{P}(\dot{\mathbf{y}}|\dot{\mathbf{x}})$ over a two-dimensional lattice $\mathbf{F}$ with dimensions $(n+1)\times(n'\times 1)$, where we index the lattice points starting from zero. A horizontal transition in the lattice represents an insertion with probability $\frac{p_{\mathrm{I}}}{q}$ and a vertical transition represents a deletion with probability p_{D}. A diagonal transition is either represented by a substitution with probability p_{S} when $\dot{x}_\iota \neq \dot{y}_\mu$, and by a transmission with probability p_{T} otherwise. We compute $\mathbf{F}_{n,n'} \triangleq \mathbb{P}(\dot{\mathbf{y}}|\dot{\mathbf{x}})$ by marginalizing over all paths starting from $\mathbf{F}_{0,0}$. We can compute this quantity for all $0 < \iota < n$ and $0 < \mu < n'$ iteratively by the update rule

$$\mathbf{F}_{\iota,\mu} = \begin{cases} \frac{p_{\mathrm{I}}}{q}\mathbf{F}_{\iota,\mu-1} + p_{\mathrm{D}}\mathbf{F}_{\iota-1,\mu} + \frac{p_{\mathrm{S}}}{q-1}\mathbf{F}_{\iota-1,\mu-1} & , \dot{x}_\iota \neq \dot{y}_\mu \,, \\ \frac{p_{\mathrm{I}}}{q}\mathbf{F}_{\iota,\mu-1} + p_{\mathrm{D}}\mathbf{F}_{\iota-1,\mu} + p_{\mathrm{T}}\mathbf{F}_{\iota-1,\mu-1} & , \dot{x}_\iota = \dot{y}_\mu \,, \end{cases}$$

where we initialize $\mathbf{F}_{0,0} = 1$ and all other vertices to zero. Since we model insertions as self-loops, i.e., before we move according to another event in time, we do not allow insertion transition in the last row of the lattice. Hence, for the last row, we update by

$$\mathbf{F}_{n,\mu} = \begin{cases} p_{\mathrm{D}}\mathbf{F}_{n-1,\mu} + \frac{p_{\mathrm{S}}}{q-1}\mathbf{F}_{n-1,\mu-1} & , \dot{x}_n \neq \dot{y}_\mu \,, \\ p_{\mathrm{D}}\mathbf{F}_{n-1,\mu} + p_{\mathrm{T}}\mathbf{F}_{n-1,\mu-1} & , \dot{x}_n = \dot{y}_\mu \,. \end{cases}$$

We illustrate the lattice in Figure 6.3. In the same line of thought, a BCJR decoder may also work on coded input by combining the channel code trellis with the HMM trellis and using the lattice computation to compute the branch metric, e.g., as in [BF15; MLW+22].

In theory, the length n' is unbounded since we model the insertion length to follow a geometric distribution. This makes an optimal decoder infeasible [DM01] and restricts applying known information-theoretic methodologies (e.g., the extension of Shannon's capacity

theorem [Dob67; AW71; Sha48; CR20]). To make decoding feasible, the decoder works on an auxiliary channel model, where one limits maximum insertion length per symbol to $I^{\max}$ and restricting the maximum drift values to an interval such that $\mathsf{D}_t \in [-d_t^{\max}, d_t^{\max}]$. To take this into account, we normalize the transition probabilities given in Equation (6.2) with a constant $\frac{1}{1-p_{\mathrm{I}}^{I^{\max}}}$ and consider a maximum drift transition $\Delta_d = I^{\max} + 1$ [DM01]. The maximum drift $d_t^{\max}$ is usually limited to

$$d^{\max} \triangleq \mathbb{E}_{\mathsf{D}_t}\left[\mathsf{D}_t\right] + 5 \cdot \sqrt{\mathbb{V}_{\mathsf{D}_n}\left[\mathsf{D}_n\right]} = t \cdot \frac{p_{\mathrm{I}} - p_{\mathrm{D}}}{1 - p_{\mathrm{I}}} + 5 \cdot \sqrt{n \frac{(1-p_{\mathrm{D}})(p_{\mathrm{D}} + p_{\mathrm{I}})}{(1-p_{\mathrm{I}})^2}}\,, \tag{6.3}$$

as proposed in [DM01]. Consequently, the auxiliary channel can be described as a factor graph with finite state space, i.e., with input $\mathbf{X} \in \Sigma_q^n$, output $\mathbf{Y} = (\mathbf{Y}_1 \parallel \ldots \parallel \mathbf{Y}_n)$, where $\mathbf{Y}_t \in \left\{\Sigma_q^{1+I^{\max}} \cap \{\}\right\}$, and state sequence $\mathbf{D} \in [-d^{\max}, d^{\max}]^{n+1}$ (in contrast to $\mathbb{Z}^{n+1}$), using the factorization as in Equation (6.1). Moreover, the state process fulfills ergodicity, i.e., we have that $\mathbb{P}\left(\mathsf{D}_t | \mathsf{D}_0\right) > 0$ for all values $\mathsf{D}_0, \mathsf{D}_t$ for all sufficiently large t [ALV+06].

For such an IDS channel, according to [Dob67; AW71; CR20] the capacity is given by

$$\lim_{n\to\infty} \frac{1}{n} \max_{\mathbb{P}(\mathbf{X})} \mathrm{I}\left(\mathbf{X}; \mathbf{Y}\right)\,,$$

under the similar assumption that whenever $\mathbb{P}\left(\mathbf{Y}_t | \mathsf{X}\right) > 0$ it holds that $0 \leq |\mathbf{Y}_t| \leq \Gamma$, where $|\mathbf{Y}_t|$ is the length of $\mathbf{Y}_t$ and Γ is a constant. In other words, this means that the expected length of the output is bounded (which is true for the auxiliary model of the decoder above). However, there is no closed-form expression for the exact capacity, and the maximizing input distribution $\mathbb{P}\left(\mathbf{X}\right)$ is unknown [CR20]. Multiple lower and upper bounds on the capacity exist; for a survey on the capacity of synchronization channels, we refer to [Mit09; CR20].

6.2.3 Trace Reconstruction Channel with i.i.d. Insertion/Deletion/Substitution Noise

We introduce the trace reconstruction channel with i.i.d. IDS noise. In line with [SK20; MLW+22; SGP+21], we consider that a sequence is transmitted over a constant (with respect to the sequence length) number T independent identical IDS channels presented in Section 6.2.2. Formally, let $\mathbf{X} \in \Sigma_q^n$ be the input, the channel outputs T sequences $\mathbf{Y}_1, \ldots, \mathbf{Y}_T$, each independently transmitted through an IDS channel with parameters $p_{\mathrm{I}}, p_{\mathrm{D}}$, and p_{S}. We illustrate the channel model in Figure 6.4.

Similar as in the single sequence transmission described in Section 6.2.2, the channel can be expressed as a T-state HMM with the hidden drift vectors $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_T$ for the corresponding output sequences $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_T$ [SK20; MLW+22; SGP+21]. A straightforward s-MAP decoding approach is to formulate BCJR recursions on the trellis produced by the T-state HMM. However, with an increasing number of sequences, this quickly becomes infeasible since the number of possible states grows exponentially with T. In [MLW+22], we proposed a low-complexity decoding strategy, the so-called *separate decoding* strategy, which computes mismatched APPs by decoding each trace independently and combining the APPs by simple multiplication. The idea originates from decoding multiple memoryless channels where

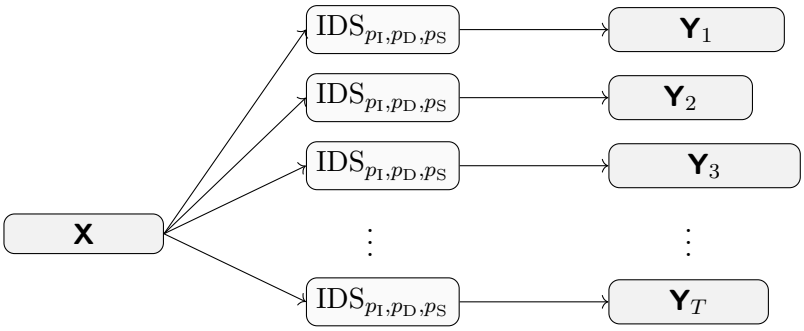


Figure 6.4: Multiple traces i.i.d. IDS channel.

this approximation holds exactly. Concurrently, a similar approach was also developed by [SGP+21]. In this work, the authors used techniques from extrinsic information-processing [KFL01] to share information between the individual trace decoders and to post-process the combined APPs. We focus on the approach from [MLW+22] since this approach requires less hyperparameter tuning. A combined *mismatch* symbolwise APP can be computed by

$$\mathbb{Q}\left(x_t|\mathbf{y}_1,\ldots,\mathbf{y}_T\right) \triangleq \frac{1}{\mathbb{P}\left(x_t\right)^{T-1}} \prod_{i=1}^{T} \mathbb{P}\left(x_t|\mathbf{y}_i\right) , \tag{6.4}$$

where $\mathbb{P}\left(x_t|\mathbf{y}\right)$ is computed by a BCJR decoder presented in Section 6.2.2. Note that in a subsequent chapter, we may mitigate under-/overconfidence of the computed APPs by further computing $\mathbb{Q}\left(x_t|\mathbf{y}_1,\ldots,\mathbf{y}_T\right)^{s(T)}$, where $s(T)$ is chosen such that the BCJR-once rate is maximized. This is inspired by the idea of mismatched decoding (see Section 6.1) and is also applied in [SGP+21]. Since this is an order-preserving transformation, this approach is only beneficial for a coding scheme in which a decoder can handle soft input (APPs).

Chapter 7

An End-to-End Coding Scheme for DNA Storage With Nanopore-Sequenced Reads

Abstract

We consider error-correcting coding for DNA storage using nanopore sequencing. The DNA storage channel is modeled as a sampling noise channel where the input data is chunked into M short DNA strands, which are copied a random number of times, and the channel outputs a random selection of N noisy DNA strands. The retrieved DNA reads are prone to strand-dependent insertion, deletion, and substitution (IDS) errors. We construct an index-based concatenated coding scheme consisting of the concatenation of an outer code, an index code, and an inner code. We further propose a low-complexity (linear in N) maximum a posteriori probability decoder that takes into account the strand-dependent IDS errors and the randomness of the drawing to infer symbolwise a posteriori probabilities for the outer decoder. The proposed system is evaluated by Monte-Carlo simulations for information-outage probabilities and frame error rates for different channel setups on experimental data. We finally evaluate the overall system performance using the read/write cost trade-off. A powerful combination of tailored channel modeling and soft information processing allows us to achieve excellent performance even with error-prone nanopore-sequenced reads outperforming state-of-the-art schemes.

This chapter is based on the work [WML+23; WSH+24]. Lorenz Welter contributed to the conception of the projects, the derivation of the results, the data analysis of the raw experimental data, and the writing of the manuscripts.

7.1 Introduction

To enable commercial application of DNA storage, it is essential to develop practical end-to-end coding solutions which ensure reliable but also cost-efficient storage. Initially, research focused on general proofs-of-concept, whereas nowadays, the focus has shifted to experimental and theoretical investigation of particular aspects of the storage solution. In Section 2.1, we have provided an elaborative survey on DNA storage with a focus on the system components and provided a state-of-the-art experiment review.

In short, the DNA storage system considered in this chapter consists of three components:

1) *Writing and Synthesis:* Data is translated to several M short DNA strands of length L (typically in the range of $100 - 200\,\mathrm{nt}$), each by chaining the four DNA nucleotides $(\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T})$.

2) *Storage:* The DNA strands are stored spatially unordered within an appropriate environment. Multiple copies of each strand are directly stored, and the number of copies depends on the applied synthesis method.
3) *Reading and Sequencing:* Before reading, the DNA strands are typically copied by PCR. Then, a random subset of strands of cardinality N is drawn from the storage pool, and a sequencing device retrieves the data of the drawn DNA strands, also called DNA reads; we focus specifically on *Oxford Nanopore* sequencing.

For data consisting of multiple files, we adapt the method for random file access from [OAC+18], i.e., we utilize different primer sequences attached to the file-specific DNA strands and use primer-targeted PCR amplification for the readout. Further, we recap briefly the flow of nanopore sequencing. First, the stored double-helix DNA strands are split by an enzyme into the two single-stranded DNA strands, the so-called *forward* and *backward* strands due to their complementary characteristic. Then, the single strands are passed in random order through a pore placed on a membrane across which an electric current is measured. The change of current indicates the present nucleotide passing through the pore, where the inference is performed by a machine-learning algorithm called the *basecaller*. Due to the geometrical nature of the pore and the variying speed the strand passes through the pore, the inference of a single nucleotide is not only dependent on its own but also on the surrounding nucleotides as well. Erroneous inference may lead to insertion, deletion, and substitution errors.

The *DNA storage channel* is the fundamental theoretical end-to-end model of storing M strands of length L and reading randomly selected N noisy copies. This channel offers challenging impairments like random sampling but also inherent redundancy opportunities due to the multiple noisy copies of a single strand. Theoretical research on the DNA storage channel, which influenced our work, started with [HSR+17], where the authors derived the capacity of the DNA storage channel in the noise-free case. The authors showed that the unordered nature of the storage and the random subset selection induce a nonrecurring capacity loss even in the absence of errors. Follow-up works included substitution errors during the storage process [LSWZ+19; LSWZ+20] and various drawing distributions [LSWZ+22] in their capacity analysis. These capacity results were recently extended to any memoryless error channel within the DNA storage channel model [WM22] for a uniform drawing distribution. Interestingly, [WM22] showed that the decrease in theoretical decoding error probability is dominated by the number of stored DNA strands M, and not by the total number $M \cdot L$ of stored nucleotides. This implies that at the receiver side, reordering the DNA strands and recovering the data of erased, i.e., not drawn, strands is a crucial point. This is also underlined by its follow-up work [Wei22], where the author investigates a concatenated coding scheme based on random coding arguments with a coded index for strand reordering and additional low-complexity decoding, such as decoding strands individually before regrouping. Furthermore, noteworthy from a primarily theoretical perspective are the code constructions for the DNA storage channel with erasure noise [LHS22] and substitution noise [LWP20; HD23a]. A profound review of the theoretical aspects of the DNA storage channel is given in [SH22].

We have discussed error-correction and modeling approaches for the i.i.d. IDS channel given single or multiple reads in Chapter 6, respectively. Most relevant are the approaches

from [MLW+22; SGP+21], thoroughly discussed in Sections 6.2.2 and 6.2.3. These works model the (multiple) IDS channels as a (multi-dimensional) HMM and propose in the multiple sequence case a sub-optimal but low-complexity symbolwise APP estimator by decoding the traces first individually and combine the information by simple APP multiplication. However, as already discussed, the errors within a strand occur, in fact, in a non-i.i.d. fashion. In particular, the concatenated coding scheme for multiple reads from [MLW+22] has been extended to non-i.i.d. errors, explicitly targeting nanopore read channels, by [MRA23; HD23b]. Other modeling and coding approaches focusing on the single-read nanopore channel have been investigated in [MDK18; HCW21; MVS24; BYWZ+24]. These works model nanopore sequencing as three cascaded noise channels: intersymbol interference due to the presence of multiple nucleotides in the pore, duplication and deletion noise due to the random speed, and measurement noise. The work [CNT+20] also focuses on non-i.i.d. errors as the decoder is based on a deep-learning algorithm together with Viterbi decoding and evaluated on experimental data, which naturally contains non-i.i.d. errors.

Other related work arises in the field of population recovery, where instead of the trace reconstruction problem, the objective is to infer a set of input sequences given multiple input traces produced by a drawing distribution [BCF+19; BCS+19], yet more focused on asymptotic results. Furthermore, coding for channels that arbitrarily permute input sequences and then transmit each via a noise channel is discussed in [WG08; HSS+12].

Our contribution. With this work, we aim to advance the research on reliable DNA storage with *Oxford Nanopore sequencing* by incorporating more realistic modeling based on experimental data and introducing a flexible end-to-end/decoding solution designed for efficient error-free data retrieval. We model the DNA storage channel explicitly, including strand-dependent IDS errors, narrowing the modeling gap to the actual DNA storage channel. We call this model the *sampling KMER channel*.

We employ an index-based concatenated coding scheme and provide random access through primer-targeted PCR. In our proposed coding scheme, the information for one file is encoded by an outer code to provide overall error protection, mainly tackling unresolved errors and strand erasures of the data. Subsequently, the resulting codeword is split into M data blocks. In each data block, its index information is embedded after being encoded by an index code, whereas the data block itself is encoded by another (inner) code. Both codes are specially tailored to tackle IDS errors. The collection of M strands is then transmitted via the DNA storage channel incorporating strand erasures, multiple read copies, and strand-dependent IDS noise.

At the receiver side, each noisy strand is decoded separately by an s-MAP decoder tailored explicitly to the nanopore channel, outputting symbolwise APPs. This low-complexity strategy initially ignores the possible multi-copy gain; however, to exploit this gain, we later combine the APPs according to the subsequent clustering and index decoding steps. We optionally cluster the received strands based on the obtained APPs. An index decision is made for each cluster by jointly considering the received strands within a cluster. Accurate clustering is computationally expensive and scales polynomially in the number of reads N [RMR+17]. Hence, instead of complete clustering after the individual strand decoding, we leverage the soft output of our index decisions by grouping only reads with a clear index decision. Subsequently, we assign the remaining reads by comparing them to a constant number

of the pre-grouped reads. By that strategy, we keep the decoding complexity to scale linearly in the number of total reads N but benefit from an additional grouping performance gain.

We analyze our proposed scheme in terms of outage and FERs on experimental data. Further, we provide comparisons based on the read/write cost trade-off to other experiments and coding setups in the literature. The created experimental dataset with uniform input data is available at [SW24a].

To achieve good performance, our scheme relies on channel estimations obtained from a random dataset. Moreover, given the targeted system parameters, mainly the number of stored strands M and the targeted coverage c, we can use the random dataset to optimize the code components and decoding parameters.

We summarize our main contributions as follows:

- Competitive and practical coding scheme for an end-to-end DNA storage system.
- Flexible design of the scheme, which can be optimized to various DNA storage channel parameters.
- Incorporating strand-dependent IDS noise in the channel model and the decoder.
- Efficient decoding algorithm with complexity scaling only linearly in the number of total reads N.
- Comprehensive analysis of our scheme, providing valuable insights for the DNA storage channel.
- Verification of the correctness of our scheme on experimental data.
- Experimental random dataset with nanopore-sequenced DNA reads.

We remark that our coding approach is also applicable to storage methods without nanopore sequencing. For example, strand-dependent errors may happen as well during the synthesis process. Nonetheless, the channel model is inspired by the physical process of nanopore sequencing. Further, the optimization of our proposed system, i.e., channel estimation and decoder optimization, is based specifically on nanopore data.

The remainder of the chapter is organized as follows. In Section 7.2, we introduce the considered DNA storage channel model. We present our coding scheme in Section 7.3 and the decoding procedure in Section 7.4. Since we have conducted laboratory experiments with random data, we describe our experimental setup in Section 7.5. The optimization procedure is laid out in Section 7.6. Then, we present our results in Section 7.7, including the simulated end-to-end performance of the optimized scheme for an example target scenario. Finally, we draw some conclusions in Section 7.8.

We remark that to stay in line with the literature on the DNA storage channel and the DNA experiments, in this chapter, we use small letters n to denote lengths of data sequences and codes; we, use capital letters L to describe lengths which are related to a single DNA strand. Further, the overall coding rate is measured in information bits per nucleotide (bit/nt).

7.2 DNA Storage Channel Model

For one file, we consider the practical scenario in which a long data sequence is divided into short DNA strands, which are individually synthesized, and the PCR amplification and sequencing processes produce a random number of noisy reads for each short DNA strand.

This channel is called the *sampling noisy channel*. In the following, we focus first on the single-strand noise channel by which each DNA strand is impaired during the whole storage process. After that, we discuss the sampling nature of our channel model in detail.

7.2.1 KMER Channel Model

We consider a variant of the state-based memory-k nanopore channel model introduced in [HD23b], which we abbreviate simply as the $\texttt{KMER}_k$ channel. (We describe the modifications we introduced to the model in [HD23b] below.) The essential characteristic of this channel model is that it incorporates memory effects for errors within DNA strands—i.e., it forgoes the common assumption that errors are i.i.d. Introducing memory effects in the channel model allows us to match more closely the empirically observed behavior of nanopore sequencing, e.g., a bias toward higher deletion rates in homopolymers [DN21]. In particular, the occurrence of an error event (insertion, deletion, or substitution)—and hence the error-free transmission event as well—depends in the model on the surrounding k input symbols and on the previous event (insertion $\texttt{Ins}$, deletion $\texttt{Del}$, substitution $\texttt{Sub}$, or transmission $\texttt{Tra}$). The $\texttt{KMER}_k$ channel we use in this work is depicted schematically in Figure 7.1 and introduced formally below. The description follows the same principles used to introduce the i.i.d. IDS channel in Section 6.2.2.

Let $\mathbf{X} = (\mathsf{X}_1, \ldots, \mathsf{X}_L)$, $\mathsf{X}_t \in \Sigma_{\text{DNA}} = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, be the DNA strand to be transmitted and $\mathbf{Y} = (\mathsf{Y}_1, \ldots, \mathsf{Y}_{L'})$ the output DNA strand, where due to insertions and deletions the length of the output strand L' is random and depends on the channel realizations.[1] We define the RV

$$\mathsf{K}_t^k \triangleq (\mathsf{X}_{t-\lfloor \frac{k}{2} \rfloor}, \ldots, \mathsf{X}_t, \ldots, \mathsf{X}_{t+\lfloor \frac{k}{2} \rfloor})$$

to be the surrounding $k$mer at time instance $t \in \{\left\lfloor \frac{k}{2} \right\rfloor + 1, \ldots, L - \left\lfloor \frac{k}{2} \right\rfloor\}$ and denote as $k\text{mer}_t$ its realization. For any other possible instance t, we define K_t^k to be the longest possible but symmetric substring surrounding X_t, e.g., for $k > 1$, $\mathsf{K}_1^k = (\mathsf{X}_1)$, $\mathsf{K}_2^k = (\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3)$, etc. Due to the definition of $k$mer, we consider only odd values for k. Let $\mathsf{E} \in \{\texttt{Ins}, \texttt{Del}, \texttt{Sub}, \texttt{Tra}\}$ denote the previous channel event. Note that the total number of channel events is at least L depending on the number of insertions and at least L' due to the number of deletions.

For an input $\mathbf{X}$, we generate the output $\mathbf{Y}$ symbol by symbol. Let t and s be the integer pointing to the current input and output symbol, respectively, initialized as $t = s = 1$. Then, we successively sample from the event distribution:

1. **Insertion with probability** $\mathbb{P}\left(\texttt{Ins} | \mathsf{K}_t^k, \mathsf{E}\right)$
   A random nucleotide gets inserted. Thus, we choose $\mathsf{Y}_s \sim \mathsf{Unif}(\Sigma_{\text{DNA}})$ and set $\mathsf{E} = \texttt{Ins}$. We increase the pointer $s \leftarrow s + 1$, whereas t remains unchanged.

2. **Deletion with probability** $\mathbb{P}\left(\texttt{Del} | \mathsf{K}_t^k, \mathsf{E}\right)$
   The nucleotide X_t is deleted. Hence, we set the pointers to $t \leftarrow t + 1$ and s remains unchanged.

[1] Like in the case for the i.i.d. IDS channel (see Section 6.2.2), we do not write the output length L' as a RV.

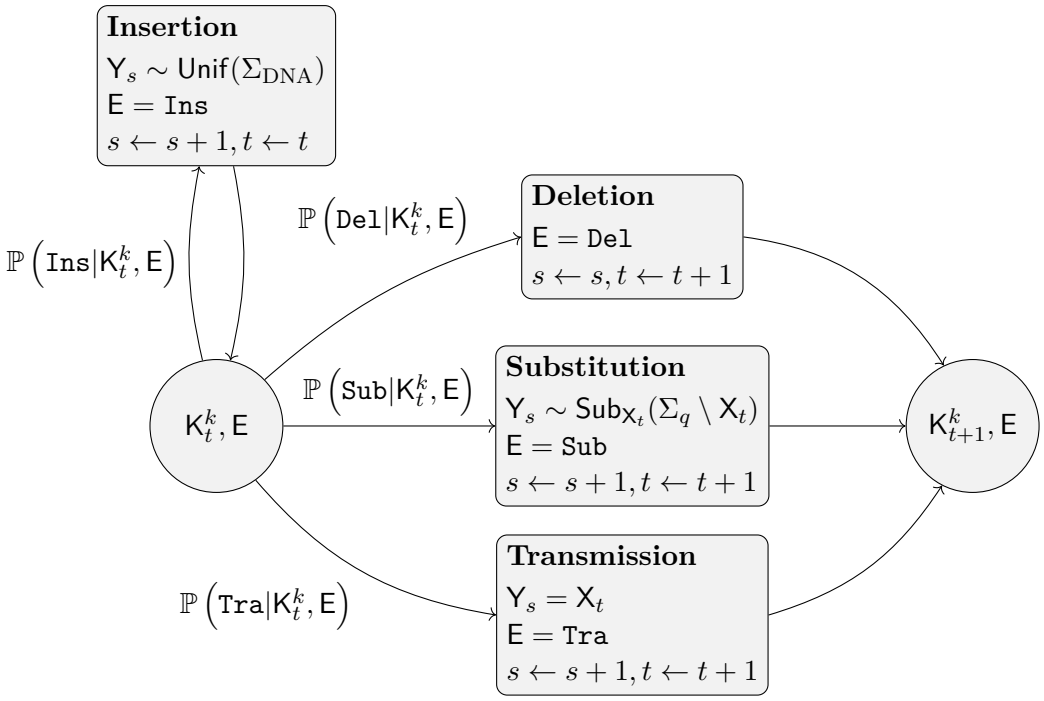


Figure 7.1: Adapted state-based memory-k nanopore ($\mathtt{KMER}_k$) channel inspired from [HD23b]. We have $\mathsf{K}_t^k = (\mathsf{X}_{t-\lfloor \frac{k}{2} \rfloor}, \ldots, \mathsf{X}_t, \ldots, \mathsf{X}_{t+\lfloor \frac{k}{2} \rfloor})$.

3. **Substitution with probability** $\mathbb{P}\left(\mathtt{Sub}|\mathsf{K}_t^k, \mathsf{E}\right)$
   The current input X_t gets substituted by another nucleotide. We choose the symbol $\mathsf{Y}_s \sim \mathsf{Sub}_{\mathsf{X}_t}(\Sigma_{\mathrm{DNA}} \setminus \mathsf{X}_t)$, where $\mathsf{Sub}_{\mathsf{X}_t}(\Sigma_q \setminus \mathsf{X}_t)$ is a distribution which is defined by the probability mass function $\mathbb{P}\left(\mathsf{Y}|\mathsf{X} = x_t, \mathsf{E} = \mathtt{Sub}\right)$ where $\mathsf{Y} \in \Sigma_{\mathrm{DNA}} \setminus \mathsf{X}_t$. We increase both pointers as $t \leftarrow t+1$ and $s \leftarrow s+1$.

4. **(Correct) Transmission with probability** $\mathbb{P}\left(\mathtt{Tra}|\mathsf{K}_t^k, \mathsf{E}\right)$
   The symbol X_t gets transmitted without error, thus, we set $\mathsf{X}_t = \mathsf{Y}_s$. Like in the substitution event, we increase both pointers as $t \leftarrow t+1$ and $s \leftarrow s+1$.

We stop when $t = L+1$ and $s = L'+1$. Like in the i.i.d. IDS channel, due to the insertion and deletion events the input and output strands are not synchronized anymore, hence, Y_t could be the result of transmitting a symbol $\mathsf{X}_{t'}$ with $t' \neq t$.

The main differences between the model described above, and the model originally introduced in [HD23b] are:

1. *The definition of the window for the kmer dependencies:* Instead of conditioning on k previously transmitted symbols, we condition on k *surrounding* symbols, in more realistic correspondence with the physical nature of the nanopore sequencing process.

2. *Modeling of the insertions:* The model in [HD23b] assumes that a deletion must follow (a burst of) insertions. In contrast, we allow for insertions to be followed by any channel event, see Figure 7.1.

7.2.2 Sampling KMER Channel Model

The data sequence is divided into M short DNA strands $\mathbf{X}_1, \ldots, \mathbf{X}_M$, each of length L, which are synthesized and stored multiple times in a *pool*. We define the input list $\underline{\mathbf{X}} \triangleq (\mathbf{X}_1, \ldots, \mathbf{X}_M)$. We assume that the overall storage process, i.e., synthesis, PCR amplification, and nanopore sequencing, generates N^{f} forward and N^{b} backward reads from the strands in the pool, where each read is a noisy (forward) or noisy reverse-complemented (backward) copy of an input strand $\mathbf{X}_i$. A reverse-complement $\mathbf{x}^{\mathrm{b}} \in \Sigma_{\mathrm{DNA}}^{L}$ of a strand $\mathbf{x}^{\mathrm{f}}$ is defined as $\mathbf{x}^{\mathrm{b}} = (\bar{x}_L^{\mathrm{f}}, \ldots, \bar{x}_1^{\mathrm{f}})$, where $\bar{x} \in \Sigma_{\mathrm{DNA}}$ denotes the complement of the symbol x with the respective complement relation A – T and C – G (Watson-Crick pairing).

Consequently, the number of total strands N is described as

$$N = N^{\mathrm{f}} + N^{\mathrm{b}} = cM\,,$$

for a constant $c \in \mathbb{R}^{+}$. In the literature, the factor c is called the *coverage depth*.

The channel between the input to be synthesized, $\underline{\mathbf{X}}$, and the output of the sequencing process, $\underline{\mathbf{Y}}$, can be split into forward and backward reads as $\underline{\mathbf{Y}} = [\underline{\mathbf{Y}}^{\mathrm{f}}, \underline{\mathbf{Y}}^{\mathrm{b}}]$. In the following, we describe the channel model in four phases.

1. **Draw:**
   N draws are performed from the list $\mathbf{X}$ at random with replacement according to the drawing probability $\mathbb{P}(\mathsf{I}_j = i)$, where $\mathsf{I}_j \in [M]$, $j \in [N]$, denotes the RV of the index of the j^{th} drawn input strand. Let D_i be the RV corresponding to the number of times strand X_i is drawn and define the $\mathbf{D} \triangleq (\mathsf{D}_1, \ldots, \mathsf{D}_M)$, with $\sum_i \mathsf{D}_i = N$, and Q_d the RV giving the number of sequences which are drawn d times. Then, $\mathbf{D}$ is multinomially distributed as $\mathbf{D} \sim \mathsf{Multinom}\,(M, N, \mathbb{P}(\mathsf{I} = i))$.

2. **Reverse-complement:**
   Each drawn strand $\mathsf{X}_{\mathsf{I}_j}$ is randomly reverse-complemented according to a Bernoulli RV $\mathsf{B} \sim \mathsf{Ber}(p^{\mathrm{rc}})$, $\mathsf{B} \in \{\mathrm{f}, \mathrm{b}\}$ and labeled accordingly as a forward "f" or backward "b" read.

3. **Transmit:**
   The drawn strands are transmitted through independent channels, where the corresponding forward $\mathtt{KMER}_k^{\mathrm{f}}$ or backward $\mathtt{KMER}_k^{\mathrm{b}}$ channel is applied. We denote by $\mathbf{Z}_j^{\mathrm{f/b}}$ the output of the j^{th} $\mathtt{KMER}_k^{\mathrm{f}}/\mathtt{KMER}_k^{\mathrm{b}}$ channel, $j \in \{1, \ldots, N\}$, and define $\underline{\mathbf{Z}} \triangleq (\mathbf{Z}_1^{\mathrm{f}}, \ldots, \mathbf{Z}_{N^{\mathrm{f}}}^{\mathrm{f}}, \mathbf{Z}_1^{\mathrm{b}}, \ldots, \mathbf{Z}_{N^{\mathrm{b}}}^{\mathrm{b}})$ of total length N.

4. **Permute:**
   To obtain the final output $\underline{\mathbf{Y}} = [\underline{\mathbf{Y}}^{\mathrm{f}}, \underline{\mathbf{Y}}^{\mathrm{b}}]$, the list $\underline{\mathbf{Z}}$ is permuted uniformly at random and subsequently split into forward reads $\underline{\mathbf{Y}}^{\mathrm{f}}$ and backward reads $\underline{\mathbf{Y}}^{\mathrm{b}}$ according to their respective labels.

We illustrate the channel model in Figure 7.2. We define the channel parameter $\beta = \frac{\log_2(M)}{L}$ that relates the total number and the length of the strands and can be interpreted as the penalty, in bit/nt, that the channel induces by the loss of order during the *permute* step.

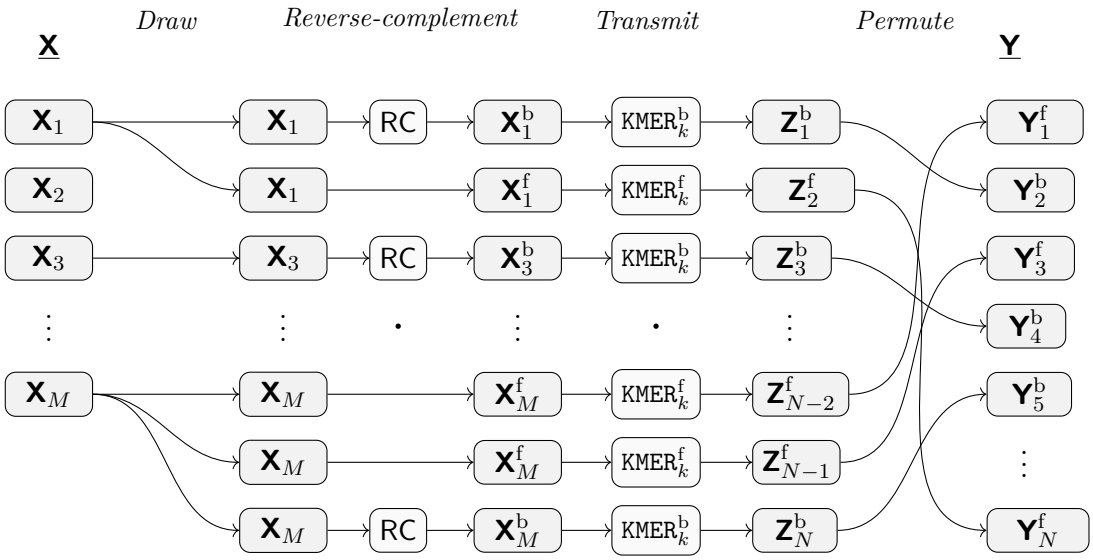


Figure 7.2: Sampling KMER channel model. First, each input strand gets copied randomly. Second, the strands get reverse-complemented (RC) according to a Bernoulli RV and get consequently labeled as forward or backward reads (see $(\cdot)^{\mathrm{f}}/(\cdot)^{\mathrm{b}}$). Third, the resulting strands are passed through independent $\mathtt{KMER}_k^{\mathrm{f}}/\mathtt{KMER}_k^{\mathrm{b}}$ channels. Fourth, the strands are randomly permuted.

We remark that in [WM22; LSWZ+22; SH22], the *draw* and *permute* steps are considered as a joint process referred to as *sampling*. We split the step for the sake of presentation similar as in the capacity proof of [LSWZ+22]; our channel model is essentially the same, except that here we consider the noise channel as a $\mathtt{KMER}_k$ channel, which induces non-i.i.d. IDS errors, and an arbitrary drawing distribution (which is however invariant to permuation).

7.2.3 Extended Trace Reconstruction Channel Model

This channel model simplifies the sampling channel model by fixing the drawing step and removing the permutation step. In particular, let the input be M strands of length L. Each input strand is drawn exactly T times. The event of reverse-complementing follows the same Bernoulli RV $B \sim \mathsf{Ber}(p^{\mathrm{rc}})$. By analyzing this simplified channel, we can gain insights into the behavior of different codes in the presence of IDS errors and perform optimization procedures for the sampling channel (see Section 7.6). We use the terminology *extended trace reconstruction channel* to highlight the close relationship to the simple *trace reconstruction channel*, as introduced in Section 6.2.3.

7.3 Index-Based Coding Scheme

We propose an index-based concatenated coding scheme consisting of three codes: an outer spatially-coupled low-density parity-check (SC-LDPC) code, providing overall protection to

the data and against non-drawn strands; an index code, which counteracts the loss of order in the presence of IDS errors; and an inner code whose primary goal is to maintain synchronization.

The information $\mathbf{u} \in \mathbb{F}_4^{k_\mathsf{o}}$ is encoded by an $[n_\mathsf{o}, k_\mathsf{o}, o_\mathrm{sc}, m_\mathrm{sc}]_4$ SC-LDPC code of output length n_o, input length k_o, coupling length o_sc, and coupling memory m_sc over the finite field $\mathbb{F}_4$ [FZ99; LSC+10]. Note that there is a one-to-one mapping of the field $\mathbb{F}_4 = \mathbb{F}_{2^2}$ to the DNA alphabet $\Sigma_\mathrm{DNA} = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. We interpret the alphabet Σ_DNA as a field $\mathbb{F}_4$ to allow linear operations over vectors in the DNA alphabet. The resulting codeword is passed through an interleaver which uniformly at random permutes the codeword symbols forming $\mathbf{w} = (w_1, \ldots, w_{n_\mathsf{o}}) \in \mathbb{F}_4^{n_\mathsf{o}}$. We split $\mathbf{w}$ into M equal-length data blocks as $\mathbf{w} = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)})$ such that $\mathbf{w}^{(i)} = (w_1^{(i)}, \ldots, w_{L_\mathsf{o}}^{(i)}) \in \mathbb{F}_4^{L_\mathsf{o}}$, $L_\mathsf{o} = \frac{n_\mathsf{o}}{M}$. Independently, the data blocks $\mathbf{w}^{(i)}$ are encoded by an $[n_\mathsf{i}, k_\mathsf{i}]_4$ inner code over $\mathbb{F}_4$ of output length n_i, input length k_i, and rate $R_\mathsf{i} = \frac{k_\mathsf{i}}{n_\mathsf{i}}$, generating an inner data codeword of length $L_\mathsf{o} R_\mathsf{i}^{-1}$. We consider the marker-repeat (MR) codes presented in [SGP+21; IK12] for the inner code, where the k_i-th input symbol is repeated once such that $n_\mathsf{i} = k_\mathsf{i} + 1$ and $R_\mathsf{i} = 1 - \frac{1}{n_\mathsf{i}}$. The MR codes have shown good performance in high-rate and low-error regimes for i.i.d. trace reconstruction channels using real DNA strands [SGP+21].[2] Further, each index i is encoded by an $[n_\mathsf{ix}, k_\mathsf{ix}]_4$ index code over $\mathbb{F}_4$ of even output length n_ix and even input length $k_\mathsf{ix} \geq \log_4(M)$ to an index codeword. Our decoding technique provides higher error protection at the beginning and at the end of the DNA strands. Thus, we split the index codeword in half and insert the two halves at the beginning and the end of the inner codeword of the respective data block $\mathbf{w}^{(i)}$. Subsequently, a random offset sequence is added to the generated strand. It supports the synchronization capability of the index and inner codes and ensures that the nucleotides of the stored DNA strands are uniformly distributed over Σ_DNA. Finally, the fixed file-specific front and back primer sequences $\mathbf{p}_1, \mathbf{p}_2 \in \Sigma_\mathrm{DNA}^{L_\mathrm{p}}$ are attached, resulting in the final encoded output strand $\mathbf{x}_i$, of length $L = L_\mathsf{o} R_\mathsf{i}^{-1} + n_\mathsf{ix} + 2L_\mathrm{p}$. Note that the primer sequences and the random offset are known to the decoder. The final codeword list is then described by $\underline{\mathbf{X}} = (\mathbf{x}_1, \ldots, \mathbf{x}_M)$.

The overall code rate, or information density (without primers), is

$$R = R_\mathrm{o} R_\mathrm{i} \left(2 - \frac{\beta}{R_\mathrm{ix}}\right), \quad \text{[bit/nt]}$$

with individual rates $R_\mathrm{o} = 2\frac{k_\mathsf{o}}{n_\mathsf{o}}$, $R_\mathrm{ix} = 2\frac{k_\mathsf{ix}}{n_\mathsf{ix}}$, and recall $R_\mathrm{i} = 2\frac{k_\mathsf{i}}{n_\mathsf{i}}$, all normalized to the unit bit/nt.

For decoding, it is convenient to introduce the following equivalent interpretation of the primer, index, and inner encoding as a joint process. A final encoded output strand $\mathbf{x}_i$ consists of segments encoded with different codes. This allows us to perform decoding on a single trellis, which we call the *joint-code* trellis.[3] We transform the index integer i to an index vector $\mathbf{ind}(i) = (\mathrm{ind}(i)_1, \ldots, \mathrm{ind}(i)_{k_\mathsf{ix}}) \in \mathbb{F}_4^{k_\mathsf{ix}}$. This index vector is split in half into $\mathbf{ix}_1^{(i)}$ and

[2] We consider a slight variant where we place the redundancy symbols evenly spaced within the data regions of the strand. This is due to the termination effects of our decoder at the beginning and end of the strands; hence, we obtain equal distributed synchronization capabilities along the strand.

[3] We emphasize that the *joint-code* does not include the outer code since we encode/decode the outer code separately.

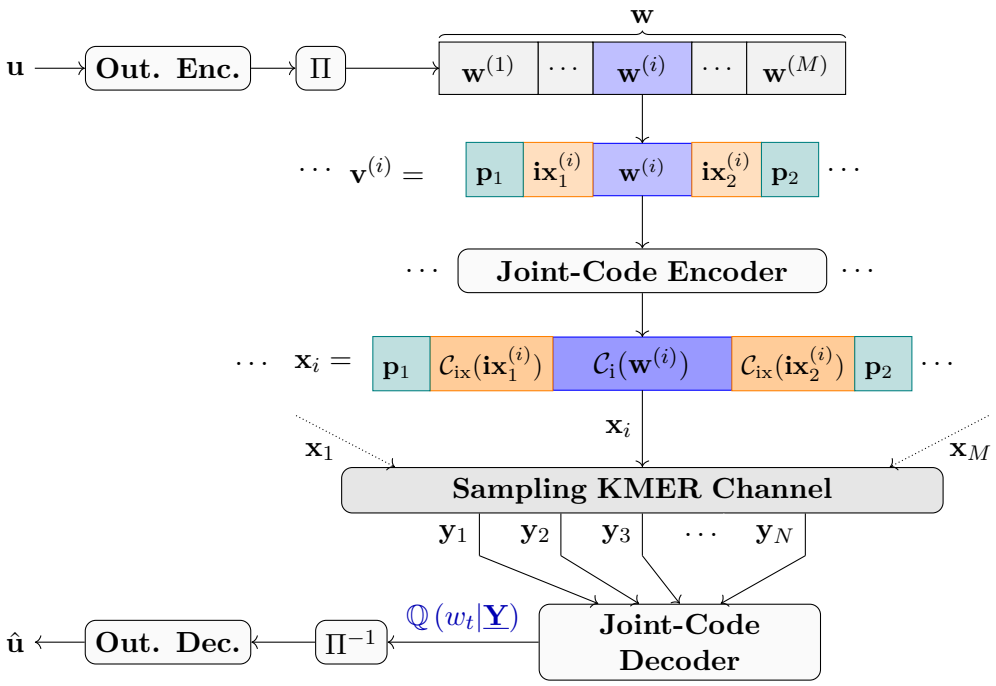


Figure 7.3: Concatenated coding scheme for communication over the sampling KMER channel. For each block $\mathbf{w}^{(i)}$, the corresponding index vector is split into two parts and inserted at the beginning and the end of the block together with the primers $\mathbf{p}_1$ and $\mathbf{p}_2$. *Joint-code* refers to the combination of primer, index, and inner code. We use $\mathcal{C}_{\text{ix}}(\cdot)$ and $\mathcal{C}_{\text{i}}(\cdot)$ as the encoding functions for the index code and inner code, respectively. This includes also the addition of the random offset symbols. The reads $\mathbf{y}_1, \ldots, \mathbf{y}_N$ are either labeled forward or backward reads. The term $\mathbb{Q}(w_t|\underline{\mathbf{Y}})$ denotes the mismatched symbol APPs computed by the joint-code decoder. Further, the symbols Π and Π^{-1} represent the random interleaving and its reversal operation, respectively.

$\mathbf{ix}_2^{(i)}$, and placed together with the corresponding primer sequence $\mathbf{p}_1$, $\mathbf{p}_2$ at the beginning and end of the vector $\mathbf{w}^{(i)}$ resulting in the vector $\mathbf{v}^{(i)} = (v_1^{(i)}, \ldots, v_{k_{\text{ix}}+L_{\text{o}}+2L_{\text{p}}}^{(i)})$, of length $k_{\text{ix}} + L_{\text{o}} + 2L_{\text{p}}$. Moreover, the $[n_{\text{ix}}, k_{\text{ix}}]_4$ index code is generated by a serial concatenation of a equal $[\frac{n_{\text{ix}}}{a}, \frac{k_{\text{ix}}}{a}]_4$ codes, where a is a multiple of 2 and divides both k_{ix} and n_{ix}. Subsequently, the vector $\mathbf{v}^{(i)}$ is encoded at the primer positions with a rate-one code, at the index positions by the $[\frac{n_{\text{ix}}}{a}, \frac{k_{\text{ix}}}{a}]_4$ codes, and symbolwise by the inner MR code at the positions corresponding to $\mathbf{w}^{(i)}$.

To further assist the reader's understanding, the coding/decoding scheme is depicted in Figure 7.3. Additionally, we zoom in on a single strand to highlight the encoding parameters in Figure 7.4.

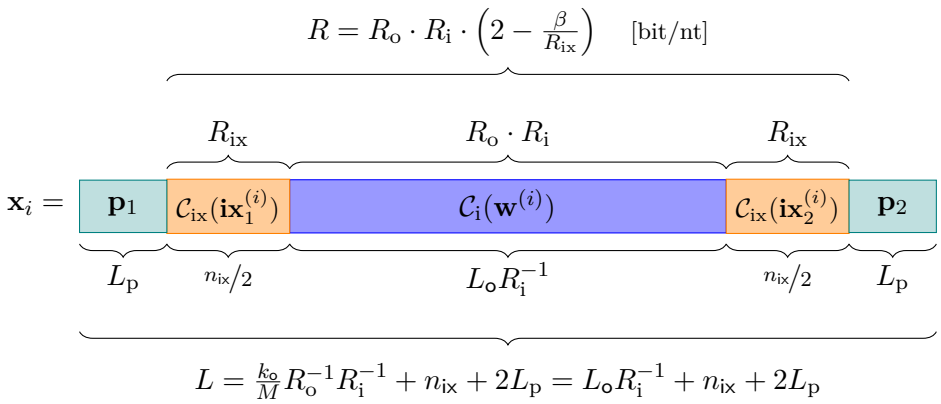


Figure 7.4: The encoding parameters for a single encoded DNA strand $\mathbf{x}_i$, $i \in [M]$. We use $\mathcal{C}_\mathrm{ix}(\cdot)$ and $\mathcal{C}_\mathrm{i}(\cdot)$ as the encoding functions for the index code and inner code, respectively. This includes also the addition of the random offset symbols. Note that $\mathcal{C}_\mathbf{w}^{(i)}$ is already encoded by the outer code, since this is part of the outer codeword $\mathbf{w} = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)})$. We indicate parameters for the length below the strand and highlight the rates above the strand.

7.4 Reads Decoding

The channel introduces three main challenges (see Figure 7.2):

1. The *randomness of the drawing* may cause an absence of input strands. However, it is also an inherent redundancy source since strands may be drawn multiple times. For unbiased drawing distributions, the coverage effectively controls the interplay of these two effects. As the coverage depth grows, the amount of provided redundancy increases; thus, the probability of absent strands decreases.

2. The insertion and deletion events during the single-strand transmission lead to a *loss of synchronization* within the respective strand.

3. The random permutation effect leads to a *loss of order* within the input strands.

Optimal decoding would consider these effects jointly to produce an estimate of the encoded information $\hat{\mathbf{u}}$ given a list of all received DNA strands $\underline{\mathbf{Y}}$; such decoding is infeasible in practice. Instead, we propose a sub-optimal decoding scheme based on *mismatched decoding* principles (see Section 6.1). In essence, we first perform inner strand-level decoding for each read, then group the generated estimations into clusters and reconstruct their ordering, combine estimations within each cluster, and finally feed these consensus estimations to the outer decoder to output an estimate $\hat{\mathbf{u}}$. We describe our decoding procedure in detail below, and for a better grasp, a simplified flowchart is depicted in Figure 7.5.

First, we infer APPs using an s-MAP decoder which works on the joint-code (recall combined primer, index, and inner code) and the forward/backward channel trellis by employing the BCJR algorithm [BCJ+74] for each received strand $\mathbf{y}_1, \ldots, \mathbf{y}_N$ independently (Section 7.4.1). The channel labels the received strands at no cost as a forward or backward

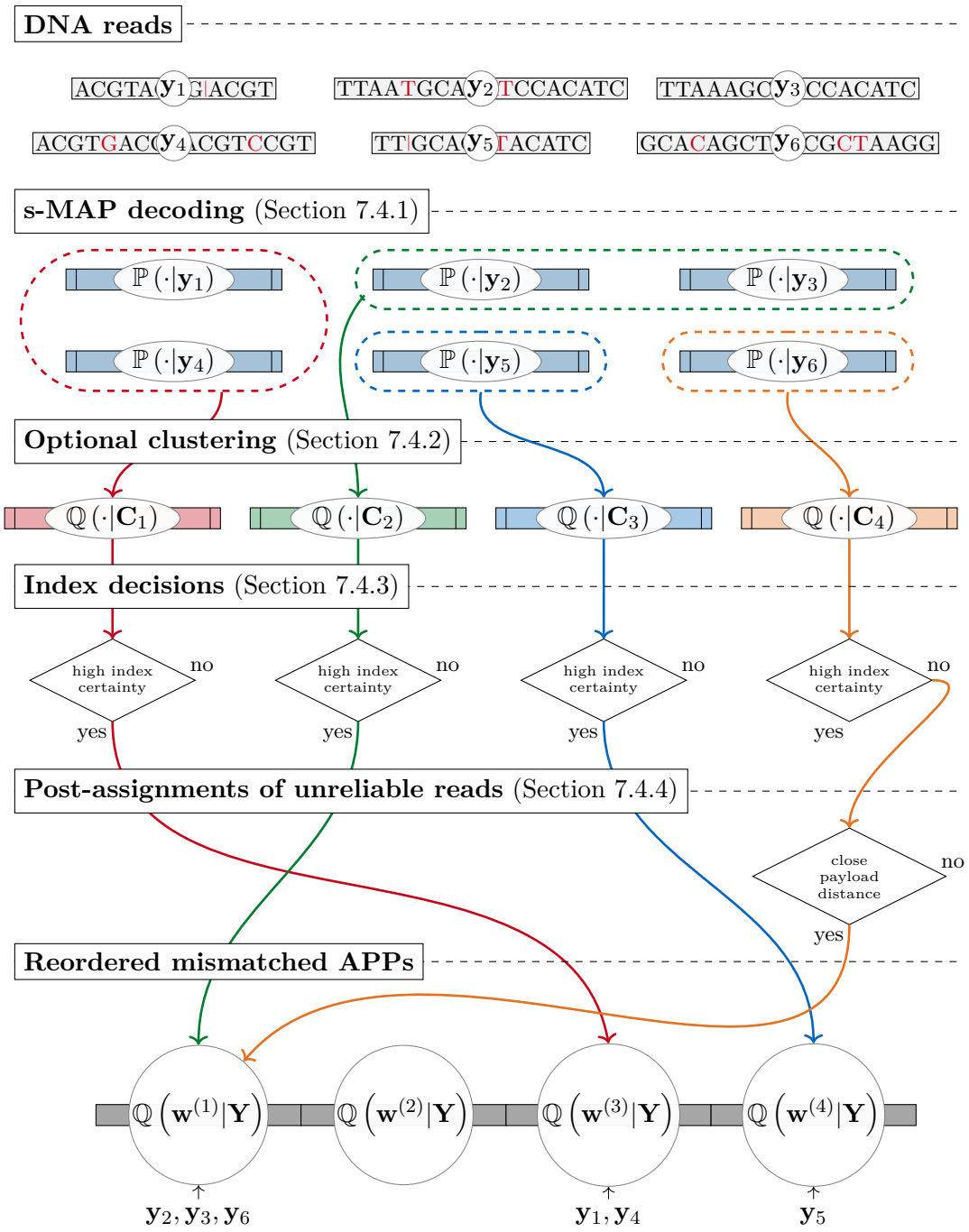


Figure 7.5: Simplified decoding flow (no outer decoding). For simplicity, we do not differentiate forward or backward reads. The arrows in the last row of the figure indicate the resulting assignments of the DNA reads $\mathbf{y}_1, \ldots, \mathbf{y}_6$ to the respective outer codeword blocks.

(i.e., reverse-complemented) read/[4]. Hence, we use separate channel estimations for forward and backward reads in our decoding. Optionally, clustering of the DNA strands using the APPs is performed (Section 7.4.2) For combining the APPs within a cluster, the APPs are symbolwise multiplied, generating the mismatched APPs for one cluster. By making another mismatched (ignoring APP correlations) ML decision on the index APPs for a given cluster (Section 7.4.3), we assign the block index $\hat{i}$ within $\mathbf{w}$ whenever the decoder-assigned probability of $\hat{i}$ is higher than a predefined threshold; otherwise, we keep the reads of the cluster for post-processing. Therefore, assuming optimal clustering, we can benefit from the multi-copy gain of the channel for index estimation. We later attempt to assign the reads that were ruled out based on the index decision threshold by comparing their *payload* distances, i.e., aggregated APP distance, to already-assigned reads (Section 7.4.4). The APPs of the data block belonging to the same index $\hat{i}$ are multiplied, generating the mismatched APPs $\mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right)$. These APPs are further processed to mitigate underconfidence/overconfidence bias following ideas from the theory of generalized mutual information (for more information, see Definition 6.2). Finally, the overall ordered APPs $\mathbb{Q}\left(w_t|\underline{\mathbf{Y}}\right)$ are passed to the outer decoder, reversing the interleaving operation and neglecting any possible correlations between the symbolwise APPs. The combination of the outer code and the interleaver is designed to handle *block-fading* effects (i.e., the loss of strands), as described in Section 7.6.5.

Combining APPs belonging to different strands by symbolwise multiplication is inspired by the well-performing *separate decoding* strategy of [MLW+22], already discussed in Section 6.2.3.

7.4.1 Symbolwise MAP Decoding of a DNA Read

In this subsection, we consider the case of decoding a single output forward read $\mathbf{y} \in \underline{\mathbf{Y}}^{\mathrm{f}}$ of length L' given an input $\mathbf{v}$ of length $k_{\mathsf{ix}} + L_{\mathsf{o}} + 2L_{\mathrm{p}}$, which includes the information words of the primer sequences, the index, and the coded data, using the BCJR decoder [BCJ+74] over the trellis combination of the $\mathtt{KMER}_k^{\mathrm{f}}$ channel with the joint-code trellis. Processing the backward reads follows similarly by reverse-complementing the chain of codebooks of the joint-code trellis and combining it afterward with the backward channel trellis. We drop the superscript of the sequence $\mathbf{v}$, i.e., its index information, due to the permutation effect of the channel. We follow the basic concept for decoding synchronization errors introduced in Section 6.2.2 for the i.i.d. IDS channel. Approaches that already adapted this concept for slightly different $\mathtt{KMER}_k$ channels have been independently proposed in [MRA23] and [HD23b].

Due to the synchronization loss, we model the combination of the joint-code and channel graph as an 3-state HMM with input $\mathbf{V}$ and output $\mathbf{Y}$. For each $t \in [0, k_{\mathsf{ix}} + L_{\mathsf{o}} + 2L_{\mathrm{p}}]$, the three states are the current $k$mer K_t^k, the channel event E_t, and the current drift D_t (see Section 6.2.2). We introduce the joint state as $\mathsf{S}_t \triangleq (\mathsf{K}_t^k, \mathsf{E}_t, \mathsf{D}_t)$. Let l_t denote the number of code symbols at trellis section t and $L_{t-1} = \sum_{i=1}^{t-1} l_i$. A transition from time $t-1$ to t corresponds to a transmission of code symbols $\dot{\mathbf{X}}_t \triangleq (\mathsf{X}_{L_{t-1}+1}, \ldots, \mathsf{X}^{L_t})$ corresponding to symbol V_t over the $\mathtt{KMER}_k^{\mathrm{f}}$ channel. Hence, at time t, the HMM models the emission of $l_t + \mathsf{D}_t - \mathsf{D}_{t-1}$ symbols, denoted as $\dot{\mathbf{Y}}_t$, depending only on the transition probabilities of

[4]The decoder relies on its knowledge of the primer sequences $\mathbf{p}_1, \mathbf{p}_2$ (see Figure 7.3) to distinguish between forward and backward reads.

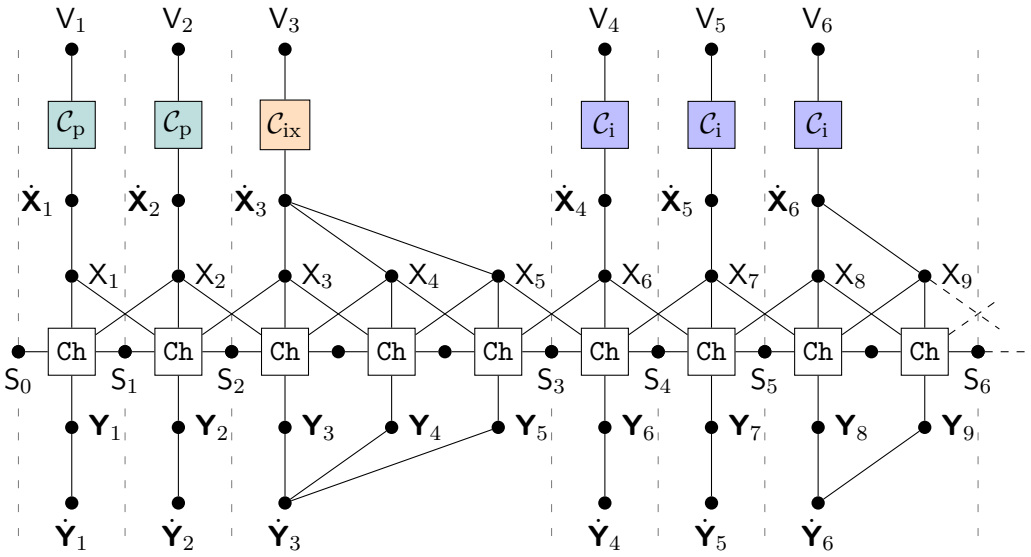


Figure 7.6: Factor graph of the combined joint-code and $\texttt{KMER}_k$ trellis when described as an HMM. The black dots represent variable nodes and the squares the factor nodes. We set $L_{\mathsf{o}} = 2$, $k_{\text{ix}}/a = 1$, $n_{\text{ix}}/a = 3$, $R_{\text{i}} = 3/4$, and $k = 3$. We distinguish between factor nodes for the codewords for the different codebooks $\mathcal{C}_{\text{p}}$, $\mathcal{C}_{\text{ix}}$, $\mathcal{C}_{\text{i}}$ and the channel $\texttt{Ch}$ ($\texttt{KMER}_k$). The output is formed by $\mathbf{Y} = (\mathbf{Y}_1 \parallel \ldots \parallel \mathbf{Y}_9 \parallel \ldots) = (\dot{\mathbf{Y}}_1 \parallel \dot{\mathbf{Y}}_2 \parallel \ldots \parallel \dot{\mathbf{Y}}_8 \parallel \ldots)$ where the boundaries are unknown.

current state S_t and the previous state S_{t-1}. We remark that within this transmission, there could be intermediate channel states not included in the combined joint-code and channel trellis since it is possible that $l_t \geq 1$, in contrast to the uncoded i.i.d. IDS HMM presented in Section 6.2.2. Moreover, due to $k$mer and event dependency, there is a memory effect of up to k symbols and of one previous channel event. Thus, we say that the HMM, emitting the state sequence $\mathbf{S} = (\mathsf{S}_0, \ldots, \mathsf{S}_{L_{\mathsf{o}}+k_{\text{ix}}+2L_{\text{p}}})$, has memory k. We depict a simplified factor graph of the HMM in Figure 7.6.

Due to modeling as an HMM, we can infer the APP of a symbol v_t given a channel realization $\mathbf{y}$ similarly as in the i.i.d. IDS chnannel (Section 6.2.2 by means of a BCJR decoder [BCJ+74]. We recap the steps and introduce the differences throughout the process.

The symbol APP can be obtained by demarginalizing as

$$\mathbb{P}(v_t|\mathbf{y}) = \frac{\mathbb{P}(v_t, \mathbf{y})}{\mathbb{P}(\mathbf{y})} \propto \sum_{s_{t-1}, s_t} \mathbb{P}(\mathbf{y}, v_t, s_{t-1}, s_t) \ ,$$

Futher the joint probability $\mathbb{P}(\mathbf{y}, v_t, s_{t-1}, s_t)$ can be split into the three terms forward metric $\alpha_{t-1}(s_{t-1})$, the branch metric $\gamma_t(s_{t-1}, s_t, v_t)$, and the backward metric $\beta_t(s_t)$ as follows:

$$\begin{aligned}\mathbb{P}(\mathbf{y}, v_t, s_{t-1}, s_t) &= \mathbb{P}\left(\mathbf{y}_1^{L_{t-1}+d_{t-1}}, s_{t-1}\right) \mathbb{P}\left(\mathbf{y}_{L_{t-1}+d_{t-1}+1}^{L_t+d_t}, v_t, s_t \middle| s_{t-1}\right) \mathbb{P}\left(\mathbf{y}_{L_t+d_t+1}^{L'} \middle| s_t\right) \\ &= \alpha_{t-1}(s_{t-1}) \cdot \gamma_t(s_{t-1}, s_t, v_t) \cdot \beta_t(s_t) \,,\end{aligned}$$

where d_t (similarly d_{t-1}) is a member of the joint state realization $s_t = (k\text{mer}_t, e_t, d_t)$. We can deduce the forward and backward recursions

$$\alpha_t(s_t) = \sum_{s_{t-1}} \sum_{x_t} \alpha_{t-1}(s_{t-1})\gamma_t(s_{t-1}, s_t, x_t)\,, \qquad \beta_{t-1}(s_{t-1}) = \sum_{s_t} \sum_{x_t} \gamma_t(s_{t-1}, s_t, x_t)\beta_t(s_t)\,,$$

with the starting conditions:

$$\alpha_0(s_0) = \begin{cases} \frac{1}{4} & , s_0 = (x, \texttt{Beg}, 0)\,, \\ 0 & \text{otherwise}\,; \end{cases}$$

$$\beta_{L_\text{o}+k_\text{ix}+2L_\text{p}}(s_{L_\text{o}+k_\text{ix}+2L_\text{p}}) = \begin{cases} \frac{1}{12} & , s_{L_\text{o}+k_\text{ix}+2L_\text{p}} = (x, e, L' - L)\,, \\ 0 & \text{otherwise}\,; \end{cases}$$

for any $x \in \Sigma_\text{DNA}$ and $e \in \{\texttt{Del}, \texttt{Sub}, \texttt{Tra}\}$, since we cannot have a final state `Ins`.

The branch metric

$$\gamma_t(s_{t-1}, s_t, v_t) \propto \mathbb{P}\left(v_t\right) \mathbb{P}\left(\mathbf{y}_{L_{t-1}+d_{t-1}+1}^{L_t+d_t}, s_t | s_{t-1}, v_t\right)$$

represents the likelihood of emitting symbols $\dot{\mathbf{y}}_t = \mathbf{y}_{L_{t-1}+d_{t-1}+1}^{L_t+d_t}$ given the codesymbols $\dot{\mathbf{x}}_t$ corresponding to symbol v_t and the state transition from s_{t-1} to s_t. This can be computed by a three-dimensional expansion of the lattice structure introduced discussed in Section 6.2.2 [BJ75]. As discussed before, the decoding trellis of the joint-code and the channel only considers channel states at time transitions t. Hence, the following lattice can be interpreted as an intermediate forward computation over multiple channel states for the same v_t. We define the lattice $\mathbf{F}$ of dimensions $(l_t + 1) \times (l_t + d_t - d_{t-1} + 1) \times 5$, where the first dimension corresponds to the length (plus one) of the codeword $\dot{\mathbf{x}} = (\dot{x}_1, \ldots, \dot{x}_{l_t})$ associated with the symbol v_t, the second dimension to the given window $\dot{\mathbf{y}} = (\dot{y}_1, \ldots, \dot{y}_{l_t+d_t-d_{t-1}})$ according to the considered drift changes (plus one), and the third dimension to the possible channel event states $\mathcal{E} = \{\texttt{Beg}, \texttt{Ins}, \texttt{Del}, \texttt{Sub}, \texttt{Tra}\}$.[5] The lattice $\mathbf{F}$ is initialized as $\mathbf{F}_{\iota,\mu,\epsilon} = 0$ for all possible ι, μ, ϵ, except we set $\mathbf{F}_{0,0,e_{t-1}} = 1$. Note that the lattice computation depends on the considered incoming state s_{t-1} and outgoing state s_t; thus, the possible choices of current state $k\text{mer}_{t-1}$ and future state $k\text{mer}_t$ depend partly on the choice of $\dot{\mathbf{x}}$, or vice versa.

We will compute

$$\mathbb{P}\left(\mathbf{y}_{L_{t-1}+d_{t-1}+1}^{L_t+d_t}, s_t | s_{t-1}, v_t\right) = \mathbf{F}_{l_t, l_t+d_t-d_{t-1}, e_t}\,,$$

where e_t is the final event after the transmission of $\dot{\mathbf{x}}$ of the information symbol v_t. Note that e_t cannot be the event `Ins` since we model the insertions always before a time change. The value $\mathbf{F}_{l_t, l_t+d_t-d_{t-1}, e_t}$ can be computed by marginalizing over all possible paths originating from $\mathbf{F}_{0,0,e_{t-1}} = 1$ and ending in $\mathbf{F}_{l_t, l_t+d_t-d_{t-1}, e_t}$. This can be iteratively computed. For

[5] For the sake of presentation, we omit the time dependency t for $\dot{\mathbf{x}}$, $\dot{\mathbf{y}}$, and $\mathbf{F}_t$.

increasing ι and μ, we have the following iterative update rules,

$$\begin{aligned}
\mathbf{F}_{\iota,\mu,\texttt{Ins}} &= \sum_{\epsilon\in\mathcal{E}} \frac{1}{4}\cdot\mathbb{P}\left(\texttt{Ins}|k\text{mer}_\iota,\epsilon\right)\cdot\mathbf{F}_{\iota,\mu-1,\epsilon} \\
&\qquad 0\leq\iota<l_t \text{ and } 1\leq\mu\leq l_t+d_t-d_{t-1}\,, \\
\mathbf{F}_{\iota,\mu,\texttt{Del}} &= \sum_{\epsilon\in\mathcal{E}} \mathbb{P}\left(\texttt{Del}|k\text{mer}_\iota,\epsilon\right)\cdot\mathbf{F}_{\iota-1,\mu,\epsilon} \\
&\qquad 1\leq\iota\leq l_t \text{ and } 0\leq\mu\leq l_t+d_t-d_{t-1}\,, \\
\mathbf{F}_{\iota,\mu,\texttt{Sub}} &= \sum_{\epsilon\in\mathcal{E}} \mathbb{P}\left(\texttt{Sub}|k\text{mer}_\iota,\epsilon\right)\cdot\mathbb{P}\left(\dot{y}_{\mu-1}|\dot{x}_{\iota-1},\texttt{Sub}\right)\cdot\mathbf{F}_{\iota-1,\mu-1,\epsilon} \\
&\qquad 1\leq\iota\leq l_t \text{ and } 1\leq\mu\leq l_t+d_t-d_{t-1}\,, \\
\mathbf{F}_{\iota,\mu,\texttt{Tra}} &= \sum_{\epsilon\in\mathcal{E}} \mathbb{P}\left(\texttt{Tra}|k\text{mer}_\iota,\epsilon\right)\cdot\mathbf{F}_{\iota-1,\mu-1,\epsilon} \\
&\qquad 1\leq\iota\leq l_t \text{ and } 1\leq\mu\leq l_t+d_t-d_{t-1}\,.
\end{aligned}$$

Observe that we consider a substitution matrix but model the insertions as random symbols. We illustrate the lattice structure in Figure 7.7.

For complexity reasons, we prune the decoding trellis. In each trellis section t, we limit the maximum and minimum considered drift around its expected value exactly like in the i.i.d. case by some multiple of its total variance. The limit is computed according to Equation (6.3) using the overall average insertion and deletion probabilities. Further, the decoder considers at most $I_{\max}$ consecutive insertions per channel state transition. Moreover, we could have decided to combine multiple symbols/codebooks to possibly further enhance the decoding performance [BSW10; BF15], but the decoding complexity is dependent on the cardinality of the created codebook and additionally grows quadratically in the length of the trellis, which renders decoding infeasible for larger trellis sections.

During decoding, the primer sequences $\mathbf{p}_1$ and $\mathbf{p}_2$ are known. We use them to create 'soft' boundaries for the strand's index and payload parts. By incorporating the primers into the trellis, we avoid the negative effects of making a hard decision on primer boundaries when extracting DNA strands from the basecaller output.

7.4.2 Reads Clustering

Typical approaches involve clustering the received strands before performing error-correction decoding. These methods usually use approximation techniques for the edit distance (minimum number of IDS operations required to change one sequence into another, see Section 2.3), e.g., q-gram distance [ALD+20; RMR+17] to reduce the distance computation cost. We present a clustering approach with complexity $O(N^2L)$, which computes pairwise distances in $O(L)$ based on the resynchronized soft information the strand-by-strand decoding provides, which we call *payload* distance. We use the aggregated statistical distance between the two APP outputs as a distance metric between two reads $\mathbf{y}_{j_1}$ and $\mathbf{y}_{j_2}$. In

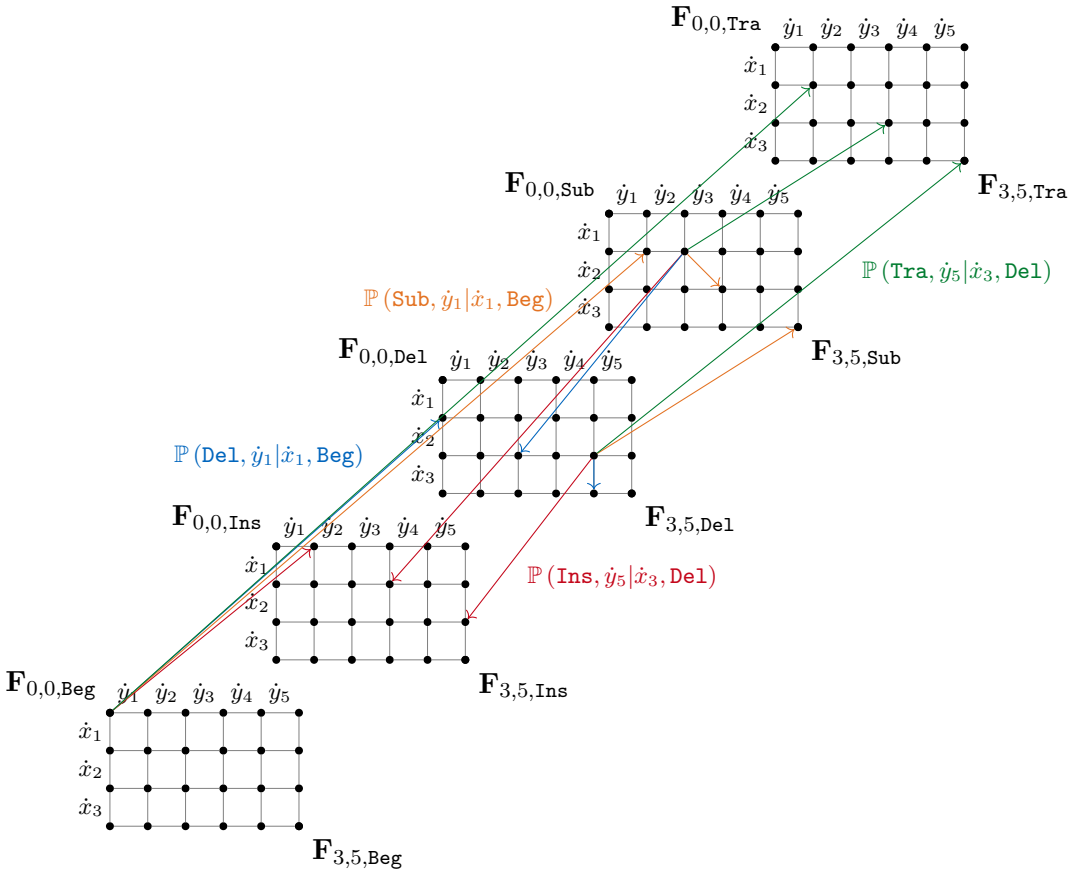


Figure 7.7: Lattice structure of $\mathbf{F}$ to compute $\mathbb{P}(\dot{\mathbf{y}}, s_t|\dot{\mathbf{x}}, s_{t-1})$ for $k = 1$. Assuming we are at vertex $\mathbf{F}_{\iota,\mu,\epsilon}$: The event `Ins` connects an edge ($\rightarrow$) to vertex $\mathbf{F}_{\iota,\mu+1,\texttt{Ins}}$; The event `Del` connects an edge ($\rightarrow$) to vertex $\mathbf{F}_{\iota+1,\mu,\texttt{Del}}$; The event `Sub` connects an edge ($\rightarrow$) to $\mathbf{F}_{\iota+1,\mu+1,\texttt{Sub}}$; The event `Tra` connects an edge ($\rightarrow$) to $\mathbf{F}_{\iota+1,\mu+1,\texttt{Tra}}$. Note that due to the channel model not all vertices can be reached, e.g., $\mathbf{F}_{3,\mu\geq 0,\texttt{Ins}}$, $\mathbf{F}_{\iota>0,\mu>0,\texttt{Beg}}$, $\mathbf{F}_{0,\mu\geq 0,\texttt{Del}}$, etc.

particular, we define

$$d_{\mathrm{C}}(\mathbf{y}_{j_1}, \mathbf{y}_{j_2}) \triangleq \sum_{t=L_{\mathrm{p}}+1}^{k_{\mathrm{ix}}+L_{\mathrm{o}}+L_{\mathrm{p}}} \frac{1}{2} \sum_{v_t} \left| \mathbb{P}(v_t|\mathbf{y}_{j_1}) - \mathbb{P}(v_t|\mathbf{y}_{j_2}) \right| ,$$

where the second sum is over all possible symbols at time t. Two reads are considered to stem from the same DNA input strand if their pairwise distance is lower than a threshold δ_{C}. For a given setup, this threshold is optimized such that the probability of clustering two reads originating from different DNA input strands is at most 10^{-6} based on simulations.

In particular, we follow a hierarchical clustering approach using the single-linkage method. Hence, we start by computing the pairwise distances of the reads and considering each read a single cluster. We iteratively merge clusters with the minimum current cluster distance,

where the cluster distance is defined as the minimum distance of any two reads of the two clusters. We stop merging whenever no cluster distance is below the threshold δ_{C}.

Let the r^{th} cluster, $1 \leq r \leq M'$, be formed as $\mathbf{C}_r \triangleq (\mathbf{y}_{r,1}, \ldots, \mathbf{y}_{r,|\mathbf{C}_r|})$, where $\mathbf{y}_{r,j}$ is the j^{th} strand placed in cluster $\mathbf{C}_r$ and $|\mathbf{C}_r| \neq 0$. We denote the overall clustering output as the list $\mathbf{C} = (\mathbf{C}_1, \ldots, \mathbf{C}_{M'})$, with $1 \leq M' \leq N$ and $\sum_{r=1}^{M'} |\mathbf{C}_r| = N$.

In general, this or other reads clustering approaches enable the system to take advantage of the multi-copy gain introduced by the channel to improve the subsequent step of index recovery of the received strands. The disadvantage of most reads clustering approaches is the associated computing cost for large M and N. We have also considered trading complexity against clustering accuracy similar to [RMR+17], where one clusters iteratively random small subsets of reads. Due to either the impracticability for large N or non-significant performance gains in the latter for a fixed coverage, we emphasize that this clustering step is optional (there are good performance gains for small N, see [WML+23]).

7.4.3 Index Decoding

The index code and recovering the index information of each strand/cluster at the receiver side are the crucial points for tackling the permutation effect and leveraging the multi-copy gain of the channel.

For all strands $\mathbf{y}_{r,j}$ of each cluster $\mathbf{C}_r$, we obtain the soft information of the indices by extracting

$$\begin{aligned}&\mathbf{p}\left(\mathbf{ind}(i)|\mathbf{y}_{r,j}\right) = \left(\mathbb{P}\left(\mathrm{ind}(i)_1|\mathbf{y}_{r,j}\right), \ldots, \mathbb{P}\left(\mathrm{ind}(i)_{k_{\mathsf{ix}}}|\mathbf{y}_{r,j}\right)\right)\\ &\triangleq \left(\mathbb{P}\left(v_{L_{\mathrm{p}}+1}|\mathbf{y}_{r,j}\right), \ldots, \mathbb{P}\left(v_{L_{\mathrm{p}}+(k_{\mathsf{ix}}/2)}|\mathbf{y}_{r,j}\right), \mathbb{P}\left(v_{L_{\mathrm{p}}+L_{\mathrm{o}}+(k_{\mathsf{ix}}/2)+1}|\mathbf{y}_{r,j}\right), \ldots, \mathbb{P}\left(_{L_{\mathrm{p}}+L_{\mathrm{o}}+k_{\mathsf{ix}}}|\mathbf{y}_{r,j}\right)\right)\end{aligned}$$

from the respective strand APPs $\mathbb{P}\left(v_t|\mathbf{y}_{r,j}\right)$. For a cluster $\mathbf{C}_r$, we generate a vector of symbolwise mismatched index APPs defined as

$$\mathbf{q}\left(\mathbf{ind}(i)|\mathbf{C}_r\right) \triangleq \left(\mathbb{Q}\left(\mathrm{ind}(i)_1|\mathbf{C}_r\right), \ldots, \mathbb{Q}\left(\mathrm{ind}(i)_{k_{\mathsf{ix}}}|\mathbf{C}_r\right)\right) .$$

For any $1 \leq \kappa \leq k_{\mathsf{ix}}$, the entries are computed according to

$$\mathbb{Q}\left(\mathrm{ind}(i)_\kappa|\mathbf{C}_r\right) \propto \left[\prod_{j=1}^{|\mathbf{C}_r|} \mathbb{P}\left(\mathrm{ind}(i)_\kappa|\mathbf{y}_{r,j}\right)\right]^{s(|\mathbf{C}_r|)} ,$$

where $s(\cdot) > 0$ is a scaling parameter inspired by the theory of generalized mutual information analysis that allows to mitigate overconfidence/underconfidence bias of the mismatched rule and is, in our scenario, dependent on the number of combined traces (see Sections 6.1 and 6.2.3).

For all possible strand indices $1 \leq i \leq M$, we derive a mismatched APP $\mathbb{Q}\left(i|\mathbf{C}_r\right)$ given $\mathbf{q}\left(\mathbf{ind}(i)|\mathbf{C}_r\right)$ using the memoryless mismatched decoding rule, i.e., assuming independence

between the above computed index APPs $\mathbb{Q}\left(\mathrm{ind}(i)_\kappa|\mathbf{C}_r\right)$. In detail, we compute

$$\mathbb{Q}\left(i|\mathbf{C}_r\right) \propto \prod_{\kappa=1}^{k_{\mathrm{ix}}} \mathbb{Q}\left(\mathrm{ind}(i)_\kappa|\mathbf{C}_r\right) ,$$

inspired by ML decoding for memoryless channels. This method naturally provides confidence levels for each cluster belonging to specific indices (if no clustering is performed for each read). We assign all reads of a cluster $\mathbf{C}_r$ to index $\hat{i}$ only if the maximum confidence is higher than a threshold δ_{I} as

$$\hat{i}(\mathbf{C}_r) = \begin{cases} \underset{1\le i\le M}{\arg\max}\ \mathbb{Q}\left(i|\mathbf{C}_r\right) & \text{if } \underset{1\le i\le M}{\max}\ \mathbb{Q}\left(i|\mathbf{C}_r\right) \ge \delta_{\mathrm{I}} , \\ \text{None} & \text{otherwise} . \end{cases}$$

Introducing the index reliability threshold δ_{I} reduces the negative effect of misassigning reads/clusters to the wrong index. This stems from the idea that read erasures are easier for the outer decoder to handle than wrongly assigned reads, especially in low-coverage scenarios. The threshold δ_{I} is optimized to yield the highest average MIR (see Section 7.6.3). This strategy can be compared to adding error-detection redundancy like a cyclic-redundancy-check to the index or the whole DNA strand, as employed, e.g., in [LCR+23], but by leveraging the soft-output decoder instead of additional redundancy.

Rather than discarding the non-assigned reads, we can store them in the set

$$\mathcal{R} = \left\{ \mathbf{y} \in \mathbf{C}_r : \underset{1\le i\le M}{\max}\ \mathbb{Q}\left(i|\mathbf{C}_r\right) < \delta_{\mathrm{I}},\ \forall\, 1 \le r \le M' \right\}$$

for post-processing as described in Section 7.4.4.

We remark that computing $\mathbb{Q}\left(i|\mathbf{C}_r\right)$ for any $i \in M$ is computationally expensive and only feasible for small enough M (e.g., we consider $M = 30\,589$ feasible, see Section 7.7). Instead one can perform hard decisions on the symbolwise index APPs $\mathbf{q}(\mathbf{ind}(i)|\mathbf{C}_r)$, e.g., as exectuted in [WML+23]. This method is less expensive but less accurate. Moreover, it does not allow for the use of the confidence threshold δ_{I} and the later presented post-processing.

Let $\mathcal{S}_i$ be the set of reads assigned to index i, i.e.,

$$\mathcal{S}_i = \left\{ \mathbf{y} \in \mathbf{C}_r : i = \hat{i}(\mathbf{C}_r),\ \forall\, 1 \le r \le M' \right\} ,$$

where $0 \le |\mathcal{S}_i| \le N$ and $1 \le i \le M$. For all data positions of the i^{th} block, i.e., for symbols of $\mathbf{w}^{(i)}$, we compute the mismatched APPs according to

$$\mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right) \propto \left[\prod_{\mathbf{y}_j \in \mathcal{S}_i} \mathbb{P}\left(w_t^{(i)}|\mathbf{y}_j\right) \right]^{s(|\mathcal{S}_i|)} ,$$

where again $s(\cdot)$ is the scaling parameter for mitigating underconfidence/overconfidence bias and depends on the number of combined traces. Further, we set $\mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right) = \frac{1}{4}$ (or equally $s(0) = 0$) when $\mathcal{S}_i = \emptyset$.

7.4.4 Post-Assignment of Reads

Assigning the reads according to the maximum soft information of the decoded index is a practical way of grouping the reads. Further, filtering according to the reliability of the actual decision using the index threshold δ_{I} can be considered a straightforward way of tackling misassignments. However, this method solely focuses on the index part of the strand and hence has the drawback of throwing away reads with possible bad index region quality but good payload quality.

The general idea of the post-assignment step is to assign the so-far-ignored reads in $\mathcal{R}$ using the *payload* distance to the already-grouped reads. In that way, we can consider the index decoding step as a sort of pre-grouping and use the computed soft information for each block $\mathbf{w}^{(i)}$ as the group representative for additional read assignments. To further reduce computational complexity, we fix the performed comparison to the top index candidates $\mathcal{T}_{\mathbf{y}}$ according to the already computed probabilities $\mathbb{Q}\left(i|\mathbf{y}\right)$ by the index decoder for each read $\mathbf{y} \in \mathcal{R}$, where $|\mathcal{T}_{\mathbf{y}}|$ is constant with respect to N.

In more detail, recall that $\mathbf{v}^{(i)}$ is the vector associated with block $\mathbf{w}^{(i)}$ plus the corresponding index and primer symbols. For $\mathbf{y} \in \mathcal{R}$, we use the distance measure $d_{\mathrm{P}}(\mathbf{w}^{(i)}, \mathbf{y})$ which is defined similarly as the clustering distance as

$$d_{\mathrm{P}}(\mathbf{w}^{(i)}, \mathbf{y}) \triangleq \sum_{t=L_{\mathrm{P}}+1}^{k_{\mathrm{ix}}+L_{\mathrm{o}}+L_{\mathrm{P}}} \frac{1}{2} \sum_{v_t^{(i)}} \left| \mathbb{Q}\left(v_t^{(i)}|\underline{\widetilde{\mathbf{Y}}}\right) - \mathbb{Q}\left(v_t^{(i)}|\mathbf{y}\right) \right| ,$$

where the computed APPs of the pre-grouped reads $\underline{\widetilde{\mathbf{Y}}}$, i.e., all reads $\mathbf{y} \notin \mathcal{R}$, are denoted as $\mathbb{Q}\left(v_t^{(i)}|\underline{\widetilde{\mathbf{Y}}}\right)$. Further, in the following, we formally define the top candidate list $\mathcal{T}_{\mathbf{y}}$ for any $\mathbf{y} \in \mathcal{R}$. Denote as $\mathbf{T}_{\mathbf{y}} = [i_1, \ldots, i_M]$ the list of indices sorted according to their soft information, i.e., for all $i_i < i_j$ it holds $\mathbb{Q}\left(i_i|\mathbf{y}\right) > \mathbb{Q}\left(i_j|\mathbf{y}\right)$. Then, the top candidate list $\mathcal{T}_{\mathbf{y}}$ is defined as the first $|\mathcal{T}_{\mathbf{y}}|$ elements of $\mathbf{T}_{\mathbf{y}}$. The final index assignment of the read $\mathbf{y} \in \mathcal{R}$ is then performed as

$$\hat{i}(\mathbf{y}) = \begin{cases} i & \text{if } d_{\mathrm{P}}(\mathbf{w}^{(i)}, \mathbf{y}) < \delta_{\mathrm{C}} \text{ for any } i \in \mathcal{T}_{\mathbf{y}} \text{ and } \mathcal{S}_i \neq \emptyset, \\ \text{None} & \text{otherwise}\,. \end{cases}$$

If several candidates fulfill the distance requirement, we select $\hat{i}(\mathbf{y})$ with the highest $\mathbb{Q}\left(\hat{i}|\mathbf{y}\right)$.

For a specific block index i, let $\mathcal{W}_i$ be the union of the pre-assigned reads $\mathcal{S}_i$ and the assigned reads in the post-assignment step. We compute the final mismatched APPs for the block $\mathbf{w}^{(i)}$ as

$$\mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right) \propto \left[\prod_{\mathbf{y}_j \in \mathcal{W}_i} \mathbb{P}\left(w_t^{(i)}|\mathbf{y}_j\right) \right]^{s(|\mathcal{W}_i|)} .$$

Note that by this method, solely pre-assigned groups get additional reads assigned. Hence, the primary benefit of this method stems from assigning reads to groups with a small number of pre-assigned reads. On the other hand, already incorrectly pre-assigned groups will be

pushed further in the wrong direction. During optimization of the parameter δ_{I} (see Section 7.6.3), we have observed that the choice tends to thresholds close to 1. This directly helps minimize the number of incorrect assignments, hence also attempting to minimize the aforementioned 'avalanche' effect.

Further, the cardinality of the top candidate list $\mathcal{T}_{\mathbf{y}}$ is a sensitive design parameter that not only controls the complexity of the post-processing but also influences the performance of the method, again in an ambiguous way. Increasing the cardinality does increase the probability that the correct index candidate is in the list, but it also increases the probability of misassignments. We also remark that we have attempted to redo the index decision based on the newly assigned reads, but it does not seem to noticeably affect the system performance. Ultimately, some reads remain unassigned since no suitable representative might be found, e.g., due to empty clusters/no pre-assigned reads or structural errors in the index region. One might try to perform a version of iterative decoding between the outer code and this post-processing method to create representative APP vectors of otherwise-absent blocks $\mathbf{w}^{(i)}$.

7.4.5 Outer Decoding

We consider an outer decoder that ignores possible correlations between the symbolwise estimates $\mathbb{Q}\left(w_t|\underline{\mathbf{Y}}\right)$ given by the joint-code MAP decoder. Hence, we apply the mismatched outer decoding metric

$$\mathbb{Q}\left(\mathbf{w}|\underline{\mathbf{Y}}\right) = \prod_{i=1}^{M}\prod_{t=1}^{L_{\circ}} \mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right)$$

to finally output an estimate $\hat{\mathbf{u}}$ of our input data. Note that the outer decoder is aware of the applied random interleaving at the encoder's stage; hence, the decoder can reverse the operation.

We have summarized the decoding flow in Algorithm 7.1 for the reader's convenience.

7.5 Experimental Setup

The stored dataset consists of three files labeled `file1`, `file2`, and `file3`, comprised each of ∼30 600 DNA strands. Each strand has a length of 150 nt, of which 110 are dedicated to carrying data, and 40 are reserved for two file-specific 20 nt primer adapter sequences at both ends of the strand. The primer sequences were chosen among those designed by Organick *et al.* in [OAC+18]. The 110 nt data sequences were drawn uniformly at random and checked for collisions with the primers following [OAC+18]. The resulting ∼92 000 strands were synthesized by *GenScript* and stored jointly in a DNA pool. The pool was delivered in a volume of 80 µL at 17.5 ng/µL. (The synthesized pool contains a large number of physical copies of each strand.)

To retrieve a file, an aliquot of the pool is amplified via primer-targeted PCR following the amplification protocol in [OAC+18] and sequenced on a MinION Mk1C nanopore sequencing device produced by *Oxford Nanopore Technologies* with Kit 14 chemistry and the

Algorithm 7.1: Decoding of DNA reads

Input : reads $\underline{\mathbf{Y}} = \left[\underline{\mathbf{Y}}^{\mathrm{f}}, \underline{\mathbf{Y}}^{\mathrm{b}}\right]$
Output: $\hat{\mathbf{u}}$

/* Single strand decoding (see Sec. 7.4.1) */
for $\mathbf{y}_j \in \left[\underline{\mathbf{Y}}^{\mathrm{f}}, \underline{\mathbf{Y}}^{\mathrm{b}}\right]$ **do**
  if *forward* **then**
    $\mathbb{P}(\mathbf{v}|\mathbf{y}_j) \leftarrow$ BCJR($\mathbf{y}_j$,$KMER_k^{\mathrm{f}}$);
  else
    $\mathbb{P}(\mathbf{v}|\mathbf{y}_j) \leftarrow$ BCJR($\mathbf{y}_j$,$KMER_k^{\mathrm{b}}$);
/* Optional clustering of reads (see Sec. 7.4.2) */
$\mathbf{C}_1, \ldots, \mathbf{C}_{M'} \leftarrow$ clusteringAPP($\mathbb{P}(\mathbf{v}|\mathbf{y}_1), \ldots, \mathbb{P}(\mathbf{v}|\mathbf{y}_N)$);
/* Index decision (see Sec. 7.4.3) */
for $r \leftarrow 1$ **to** M' **do**
  $\hat{i} \leftarrow$ extractInd($\mathbb{Q}(\mathbf{v}|\mathbf{C}_r)$);
  /* Optional index threshold decoding */
  if $\max_{1 \le i \le M} \mathbb{Q}(i|\mathbf{C}_r) \ge \delta_{\mathrm{I}}$ **then**
    /* Assign data to outer block */
    $\mathbb{Q}\left(\mathbf{w}^{(\hat{i})}|\mathbf{Y}\right) \leftarrow$ multiplyData($\mathbb{Q}(\mathbf{v}|\mathbf{C}_r)$);
  else
    /* Save for post-assignment */
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathbf{y} : \mathbf{y} \in \mathbf{C}_r\}$;
/* Optional post-assignment (see Sec. 7.4.4) */
for $\mathbf{y} \in \mathcal{R}$ **do**
  for $i \in \mathcal{T}_{\mathbf{y}}$ **do**
    /* Compare payload dist. */
    if $d_{\mathrm{P}}(\mathbf{w}^{(i)}, \mathbf{y}) < \delta_{\mathrm{C}}$ **then**
      /* Assign data to outer block */
      $\mathbb{Q}\left(\mathbf{w}^{(\hat{i})}|\mathbf{Y}\right) \leftarrow$ multiplyData($\mathbb{P}(\mathbf{v}|\mathbf{y})$);
    break;
/* Outer decoding (see Sec. 7.4.5) */
$\hat{\mathbf{u}} \leftarrow$ OuterDecoder($\mathbb{Q}\left(\mathbf{w}^{(1)}|\mathbf{Y}\right), \ldots, \mathbb{Q}\left(\mathbf{w}^{(M)}|\mathbf{Y}\right)$);

Table 7.1: Experimental Dataset Specifications[a]

File	$M^{\mathcal{D}}$	Accurate basecalling (acc-BC)				Fast basecalling (fast-BC)			
		passed Q-score		failed Q-score		passed Q-score		failed Q-score	
		forw.	backw.	forw.	backw.	forw.	backw.	forw.	backw.
`file1`	30 589	244 510	193 079	44 411	32 826	166 713	96 385	63 160	39 254
`file2`[b]	30 589	494 026	602 463	19 658	25 987	242 106	391 019	113 530	148 695
`file3`[c]	30 588	432 501	470 052	73 106	51 346	641 832	613 096	114 602	105 714

[a]Each stored strand consists of a 110 nt random data part and on each side file-specific 20 nt primers.
[b]We used a smaller and cheaper flongle flow cell for nanopore sequencing; a larger number (24 versus 18) of PCR cycles was performed.
[c]We obtained a total 42 million reads for `file3` but only clustered a subsample of it.

corresponding standard Ligation Sequencing protocol. The resulting raw current signal readouts are basecalled (i.e., converted to sequences of nucleotides with associated quality scores) with both a fast and a high-accuracy basecaller available on the MinION device based on the *Guppy* basecaller. We refer to the two different processes of basecalling as *fast basecalling* (fast-BC) and *accurate basecalling* (acc-BC). The main difference is that fast-BC can be performed online (i.e., concurrently and keeping pace with the sequencing process) at the cost of higher error rates in the basecalled sequences, whereas acc-BC is only available offline. In both cases, a quality score threshold parameter of 8 (the default value for fast basecalling) is specified to split the reads into two groups—those that pass the quality score threshold and those that do not. To extract strand-specific parts of the basecalled reads, we align the reads against the six primer sequences used in our dataset (two primer sequences per file) using BLAST [AGM$^+$90] and crop the segments that correspond to adjacent alignments of correct primer pairs separated by 150 ± 15 nt. It is important to note that PCR amplification yields double-stranded DNA, and subsequent nanopore sequencing produces readouts of a strand in both its forward and reverse-complemented (or backward) form. We separate the two during segmentation by keeping track of the orientation of BLAST-generated alignments. In summary, for each file, we split the reads according to acc-BC versus fast-BC, passed versus failed quality score, and forward versus backward orientation, resulting in eight groups of reads per file. The number of segments in each group is shown in Table 7.1.

We remark that merging forward and backward reads by reverse-complementing the backward reads is a mistake if the channel model is not invariant under this operation; in particular, doing so for the $\texttt{KMER}_k$ channel model we describe in Section 7.2.1 would erase the asymmetries in the substitution matrix and possible directional memory effects, whereas the i.i.d. IDS channel model is invariant under such transformation.

To estimate the parameters of the channel model, we need a set of input-output pairs, which in turn means that we need to assign one of the originally stored strands as channel input to each read. To do so, we calculated the edit distance between each read and each

stored strand and assigned to each read the input strand that is closest to it.

In particular, a DNA storage dataset $\mathcal{D}$ contains $M^{\mathcal{D}}$ clusters, where each cluster consists of an input strand $\mathbf{x}^{(i)}$ of length L and its corresponding, possibly empty, set of output strands $\mathbf{y}_1^{(i)}, \ldots, \mathbf{y}_{M_i^{\mathcal{D}}}^{(i)}$.

In the following, we show how to use the dataset for the channel estimation on which our scheme relies (Section 7.5.1). Moreover, we show how we use the dataset to obtain desired DNA storage channel parameters (Section 7.5.2). In Section 7.5.2, we present a method to conduct Monte-Carlo simulations on experimental data for coding schemes that use pseudo randomization of strands, e.g., as [SGP+21].

7.5.1 Channel Estimation

In this subsection, we describe the estimation of the data-dependent parameters for our approach and analysis.

KMER Channel Transition Probability Estimation

The transition probabilities of the channel model are estimated using a DNA storage dataset containing labeled input (before synthesis) strands and the corresponding collection of output (after nanopore sequencing) strands split into forward and backward reads. We estimate a forward model $\texttt{KMER}_k^{\mathrm{f}}$ from the set of mapped input and forward read pairs $\mathcal{D}^{\mathrm{f}}$ and a backward model $\texttt{KMER}_k^{\mathrm{b}}$ from the set of mapped input and backward read pairs $\mathcal{D}^{\mathrm{b}}$. In the following, we describe the estimation of the forward model since the backward model follows similarly, taking the reverse-complement effect into account.

For each pair $(\mathbf{x}^{(i)}, \mathbf{y}_j^{(i)})$, we compute the edit distance matrix via the Wagner-Fischer algorithm [WF74]. By backtracking from the end and always choosing the minimum value (for ties deciding randomly), we can determine a chain of events $\mathbf{e}^{(i,j)}$ of length $L_e \geq L$, $e_l^{(i,j)} \in \mathcal{E} = \{\texttt{Ins}, \texttt{Del}, \texttt{Sub}, \texttt{Tra}\}$, that describes a possible outcome of how the strand $\mathbf{x}^{(i)}$ is edited to form $\mathbf{y}_j^{(i)}$. To each event, we add the information of the corresponding $k$mer and the preceding event $e' \in \mathcal{E}$ to form a chain of 3-tuples with entries $(e_l^{(i,j)}, k\mathrm{mer}_l^{(i,j)}, e_{l-1}^{(i,j)})$ and $e_0^{(i,j)} = \texttt{Beg}$. Note that by slight abuse of notation, we call here a $k$mer also a substring of $\mathbf{x}^{(i)}$ shorter than k, since for $k > 1$, the actual $k$mer$s$ at the beginning and end of $\mathbf{x}^{(i)}$ are fitted.

Subsequently, we can estimate the channel transition probabilities $\mathbb{P}\left(\mathsf{E}'|\mathsf{K}^k, \mathsf{E}\right)$ by a simple counting argument, were $\mathsf{E}' \in \{\texttt{Ins}, \texttt{Del}, \texttt{Sub}, \texttt{Tra}\}$ denotes the RV for the upcoming event, and recall E as the RV for the previous event and K^k to be RV for the current $k$mer. For a pair $(\mathbf{x}^{(i)}, \mathbf{y}_j^{(i)})$, we define

$$P_{e',e,k\mathrm{mer}}^{(i,j)} = \left|\left\{l : (e', k\mathrm{mer}, e) = (e_l^{(i,j)}, k\mathrm{mer}_l^{(i,j)}, e_{l-1}^{(i,j)})\right\}\right|.$$

Table 7.2: Average Error Rates for `File3` (Similar Across All Files)

Q-score	acc-BC[a]			fast-BC[a]		
	p_{I}	p_{D}	p_{S}	p_{I}	p_{D}	p_{S}
passed	0.009	0.014	0.020	0.014	0.038	0.043
failed	0.037	0.052	0.094	0.023	0.062	0.080

[a]We denote the average insertion, deletion, and substitution probability as p_{I}, p_{D}, and p_{S}, respectively.

Then, $\mathbb{P}\left(\mathsf{E}' = e' | \mathsf{K}^k = k\mathrm{mer}, \mathsf{E} = e\right)$ is estimated by

$$\frac{\sum_{i=1}^{|\mathcal{D}^{\mathrm{f}}|} \sum_{j=1}^{M_i^{\mathcal{D}^{\mathrm{f}}}} P_{e',e,k\mathrm{mer}}^{(i,j)}}{\sum_{\epsilon \in \{\texttt{Ins},\texttt{Del},\texttt{Sub},\texttt{Tra}\}} \sum_{i=1}^{|\mathcal{D}^{\mathrm{f}}|} \sum_{j=1}^{M_i^{\mathcal{D}^{\mathrm{f}}}} P_{\epsilon,e,k\mathrm{mer}}^{(i,j)}} .$$

We remark that instead of the counting-based parameter estimation described above we have also tried using the Baum-Welch algorithm [BPS+70] for HMMs using our single-strand BCJR decoder (presented in Section 7.4.1) on a slightly different trellis allowing state transition at the same time t to model insertions (similar to the expansion of the trellis in [SGP+21]). However, we have not observed any performance gain compared to the above counting method. Nonetheless, the Baum-Welch algorithm might become useful if the error rates are roughly known and the transition probabilities need to be estimated using only a small dataset (e.g., stored pilot strands). This is an interesting problem that would need further investigation and falls outside the scope of this work.

We list the average IDS error rates of our dataset in Table 7.2.

Draw Distribution Estimation

For each input strand $\mathbf{x}^{(i)}$, we calculate its drawing probability $\mathbb{P}(\mathsf{I} = i)$ by simply dividing the number of corresponding output strands by the total number of strands, i.e.,

$$\mathbb{P}(\mathsf{I} = i) = \frac{M_i^{\mathcal{D}}}{|\mathcal{D}|} .$$

Note that we can generate a drawing distribution for any smaller number of input DNA strands $M \leq M^{\mathcal{D}}$ by uniformly at random subsampling the clusters from the dataset. The probability of reverse-complementing, i.e., observing a backward read, is simply

$$p^{\mathrm{rc}} = \frac{|\mathcal{D}^{\mathrm{b}}|}{|\mathcal{D}|} .$$

7.5.2 Using Datasets for Simulation

Since our coding scheme uses a random offset, i.e., a scrambling sequence, we can execute performance evaluations on real data as described in [SGP+21]. In the following, we revisit

their method shortly. Let us describe a dataset as pairs of stored uniformly at random strands and corresponding reads. Let $\mathbf{x}'$ be the vector encoded by the index/inner code, Enc the encoding function, and $\mathbf{z}$ the random offset. Then, a stored sequence $\mathbf{x}$ can be described by $\mathbf{x} = \text{Enc}(\mathbf{x}') + \mathbf{z}$, where the symbols of $\mathbf{x}$ are thus uniformly distributed. Hence, we can use a random dataset for evaluation by fitting a $\mathbf{z} = \mathbf{x} - \text{Enc}(\mathbf{x}')$ for every strand of data. Since we conduct Monte-Carlo analyses, averaging over multiple simulations, we conclude that a single general fixed $\mathbf{z}$ exists that must perform as well or better by the pigeon-hole principle. Hence, the Monte-Carlo results on a random dataset are ensemble averages of the proposed system. We remark that the random offset is not applied to the primer region; hence, we are restricted in the choice of primers by the dataset.

7.6 Optimizing the Component Codes

The design of our coding scheme setup is flexible for any channel parameter β, i.e., any M and L, and $\texttt{KMER}_k$ channel noise. Nonetheless, its performance depends on how well the system parameters are optimized for the specific channel. In Section 7.5.1, we have already discussed how to infer the $\texttt{KMER}_k$ transition probabilities (we demonstrate in Section 7.7 that these parameters generalize over multiple experimental runs). This section discusses the criteria for optimizing the specific codes according to given system parameters and presents performance metrics for our coding scheme overall as well as for its components.

We give a brief overview of the optimization procedure in the following, where we always optimize based on experimental data (see Section 7.5). First, we compute AIRs for multiple inner code rates R_i on the extended trace reconstruction channel (see Section 7.2.3) for different numbers of traces T for each input strand (see Section 7.6.2). Within this computation, we directly optimize the scaling parameter $s(T)$, which mitigates the overconfidence/underconfidence bias of the mismatched decoding approach. Given the estimated drawing distribution and the information rates, we calculate an artificial read/write cost trade-off for each inner code rate R_i, assuming no permutation effect of the channel and a rate-achieving outer code. We pick the inner code rate R_i that shows the best artificial read/write cost trade-off across multiple coverage levels that we target. Then, we switch to the sampling channel, focusing on conducting the optimization based on average MIRs for the outer code, disregarding the finite-length phenomena due to computational complexity constraints. We choose the index code that maximizes the MIRs for the targeted coverage levels. Note that we do not consider clustering of reads here due to complexity in the considered parameter regime (see Section 7.4.2). Subsequently, we directly optimize the index threshold δ_I and the number of top candidates considered in post-processing $|\mathcal{T}_\mathbf{y}|$ using the same method (see Sections 7.6.3 and 7.6.4). For the best setup, we verify the MIRs with an outage analysis. The outage probability is a prominent performance metric in similar communication scenarios, e.g., the block-fading channel, and serves as a lower bound on the system's FER. Based on the outage analysis results, we construct an optimized SC-LDPC code such that its rate R_o is close to the outage bound for a targeted FER (see Section 7.6.5).

For the sake of presentation, we do not distinguish between RVs and their realizations in the following.

7.6.1 Performance Metrics

This subsection presents the main performance metrics applied in our analysis: the *read/write cost trade-off* and *outage probability*.

Read/Write Cost Trade-Off

For an overall performance measure of a DNA storage coding scheme, one can consider the read/write cost trade-off introduced in [CTL+19].[6]

Definition 7.1 (Read/Write Cost Trade-Off)**.** *We define the writing cost c_{w} as the fraction of the number of synthesized nucleotides and the number of information bits stored. Similarly, we define the reading cost c_{r} as the fraction of the number of sequenced nucleotides by the number of information bits stored. Assuming a coverage depth $c \geq 1$, this implies the following two formulations given the overall rate R:*

$$\begin{aligned} c_{\mathrm{w}} &= R^{-1} && \geq 0.5 \quad [nt/bit] \ , \\ c_{\mathrm{r}} &= cR^{-1} && \geq 0.5 \quad [nt/bit] \ . \end{aligned}$$

Outage Probability

Due to its drawing effect, the DNA storage channel seen by the outer code resembles a block-fading channel when considering a finite number of strands M and finite strand length L [WM22]. Hence, it also shares its afflictions, most notably its non-ergodic property. In particular, an *outage* event occurs when an insufficient number of strands are drawn for a specific outer code block. Formally, an outage event occurs when the instantaneous mutual information between the input and output of the channel is lower than the transmission rate R. We consider the *BCJR-once* (see Section 6.1) version of the information-outage probability adapted from [BV08], to which we refer as the *mismatched information-outage probability* q_{out}. It incorporates our mismatched decoding approach and the dispersion due to the finite blocklength phenomena.

For that, we first introduce the *instantaneous BCJR-once rate* $r_{\mathbf{d}}$, which is defined as the instantaneous symbolwise mutual information between the input and the MAP decoder output for a fixed drawing realization $\mathbf{d}$ and with uniform input distribution. This definition aligns with our applied outer decoding metric in Section 7.4.5. Fixing the input distribution, multiplying symbolwise posteriors for multiple strands, impairments introduced in the reordering, and neglecting output correlations of the APPs introduce mismatches that only decrease the metric $\mathbb{Q}(\mathbf{w}|\underline{\mathbf{Y}})$ compared to the true—but infeasible to compute—metric $\mathbb{P}(\mathbf{w}|\underline{\mathbf{Y}})$ (see Section 6.1). In detail, the instantaneous BCJR-once rate $r_{\mathbf{d}}$ for the finite-length regime and fixed drawing realization ($\mathbf{D} = \mathbf{d}$) is computed as

$$r_{\mathbf{d}} = \frac{1}{ML} \sum_{i=1}^{M} \sum_{t=1}^{L_{\mathrm{o}}} \left(-\log_2 \mathbb{P}\left(w_t^{(i)}\right) + \log_2 \mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right) \right) \ ,$$

[6] This definition is similar (inverse) to the *storage-sequencing trade-off* definition described in [SH22]. Additionally, [SH22] also considers the difference in the synthesis and sequencing cost per base. We choose [CTL+19] due to its better graphical visualization.

where $\mathbb{P}\left(w_t^{(i)}\right) = \frac{1}{4}$ is the a priori probability and $\mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}\right)$ is the decoder's mismatched metric for the symbols of the corresponding block $\mathbf{w}^{(i)}$ as computed in Section 7.4.

Definition 7.2 (Outage Probability)**.** *The mismatched information-outage probability* q_{out} *is formally defined as*

$$q_{\text{out}} = \mathbb{P}\left(r_{\mathbf{d}} < R\right) \geq p_{\text{out}}\,,$$

where R *is the overall system rate measured in bit/nt and* p_{out} *is the true outage probability.*

The value q_{out} gives a lower bound on the FER for a given encoder and mismatched decoder pair for a fixed finite number of strands, fixed finite blocklength, and fixed channel parameters. We approximate q_{out} using the Monte-Carlo method by generating drawing realizations $\mathbf{d}$ for a given R.

7.6.2 Inner Code

As mentioned in Section 7.3, we consider MR codes as our inner codes due to their good performance in the high-rate/low-error probability regime in the case of i.i.d. errors and mismatched decoding, as shown in [SGP+21]. Nevertheless, the rate $R_{\text{i}} = \frac{k_{\text{i}}}{n_{\text{i}}} = \frac{k_{\text{i}}}{k_{\text{i}}+1}$ still needs to be optimized based on experimental data.

For the extended trace reconstruction channel with a fixed number of draws T per block $\mathbf{w}^{(i)}$ (see Section 7.2.3), we can compute actual AIRs for a fixed joint-code mismatched MAP decoder on a dataset. This is due to the reintroduced ergodicity by fixing the drawing realization, fixing L, and $M \to \infty$. Specifically, we compute *BCJR-once* rates R_{trace} (see Definition 6.2), measured in bit/nt. We also follow a mismatched decoding approach for the extended trace reconstruction channel due to the separate strand decoding technique and symbolwise multiplication combining, and neglecting symbolwise APP correlations at the MAP decoder output. Let the input to the inner encoder be $\mathbf{w} = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)})$ and the output be defined as $\underline{\mathbf{Y}}^{\text{t}} = (\underline{\mathbf{Y}}_1^{\text{t}}, \ldots, \underline{\mathbf{Y}}_M^{\text{t}})$, where $\underline{\mathbf{Y}}_i^{\text{t}} = (\mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,T})$. In detail, for a fixed number of traces T and an inner code rate R_{i}, we define the achievable BCJR-once information rate as

$$R_{\text{trace}}(T, R_{\text{i}}) \triangleq \max_{s>0} \lim_{M\to\infty} \frac{1}{ML} \sum_{i=1}^{M} \sum_{t=1}^{L_{\circ}} \Big(-\log_2 \mathbb{P}\left(w_t^{(i)}\right) + \log_2 \mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}_i^{\text{t}}\right)^s \Big),$$

using for multi-read combining the decoder's mismatched metric

$$\mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}^{\text{t}}\right)^{s(T)} = \left[\prod_{j=1}^{T} \mathbb{P}\left(w_t^{(i)}|\mathbf{y}_{i,j}\right)\right]^{s(T)}.$$

We approximate the achievable BCJR-once information rate using experimental data by averaging over a large number of data blocks ($M \approx 30\,000$) as

$$R_{\text{trace}}(T, R_{\text{i}}) \approx \max_{s(T)>0} \frac{1}{ML} \sum_{i=1}^{M} \sum_{t=1}^{L_{\circ}} \left(-\log_2 \mathbb{P}\left(w_t^{(i)}\right) + \log_2 \mathbb{Q}\left(w_t^{(i)}|\underline{\mathbf{Y}}_i^{\text{t}}\right)^{s(T)} \right).$$

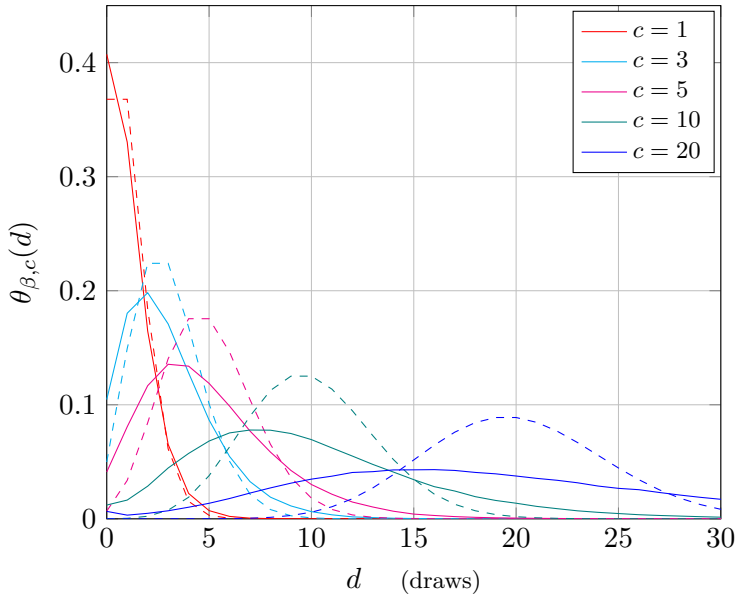


Figure 7.8: The fraction of reads $\theta_{\beta,c}(d)$ exactly drawn d times for a fixed $\beta = 0.13546$ and various coverages c obtained via subsampling from dataset `file3` using acc-BC. The dashed lines show the Poisson distribution for the respective coverage, which is obtained for uniform drawing distributions $\mathbb{P}(i) = 1/M$. Noet that the empirical drawing distributions do not differ for acc-BC and fast-BC.

Note that due to the maximization over $s(T)$, we directly obtain the values for the scaling parameter for the decoding on the sampling channel.

Furthermore, let $\theta_{\beta,c}(d)$ denote the fraction of input reads drawn d times, i.e., $\frac{\mathsf{Q}_d}{M}$, for a sampling channel with parameter β, associated drawing distribution $\mathbb{P}(\mathsf{I})$, and a certain coverage depth c. In Figure 7.8, we depict the empirical drawing distribution $\theta_{\beta,c}(d)$ of `file3` and acc-BC for various coverages c obtained via subsampling in comparison to the value obtained by uniform drawing distributions $\mathbb{P}(\mathsf{I} = i) = \frac{1}{M}$. Note that for uniform drawing distributions, $\theta_{\beta,c}(d)$ asymptotically tends to a Poission distribution, $\theta_{\beta,c}(d) = \frac{c^d}{d!}\mathrm{e}^{-c}$ [WM22; LSWZ$^+$22]. The skew in comparison to the Poisson curve indicates a drawing bias due to synthesis, PCR, or sequencing effects.

For an i.i.d. uniform data input distribution, we can compute the achievable information rate R_{th} in alignment with [SH22; LSWZ$^+$22; WM22] as

$$R_{\mathrm{th}} \triangleq \sum_{d=1}^{\infty} \theta_{\beta,c}(d) R_{\mathrm{trace}}(d, R_{\mathrm{i}}) - \beta(1 - \theta_{\beta,c}(0)) \,.$$

The ergodic rate R_{th} represents the theoretical rate for a channel with no errors in indexing and an optimal rate-achieving outer code and, thus, large enough number of input strands M. We can draw a connection between R_{th} and the instantaneous BCJR-once rate $r_{\mathbf{d}}$. In

Table 7.3: Exhaustive Search Index Codes

Code	Codebook
Code I	$(0,0,3,3)$, $(0,1,0,1)$, $(0,2,2,0)$, $(0,3,1,2)$, $(1,1,2,2)$, $(1,2,0,3)$, $(1,3,3,1)$, $(2,0,0,2)$, $(2,1,3,0)$, $(2,2,1,1)$, $(3,0,2,1)$, $(3,1,1,3)$, $(3,2,3,2)$, $(3,3,0,0)$
Code II	$(0,0,2,0)$, $(0,2,3,2)$, $(0,3,0,1)$, $(1,1,3,1)$, $(1,2,1,0)$, $(1,3,2,3)$, $(2,0,1,1)$, $(2,1,2,2)$, $(2,2,0,3)$, $(2,3,3,0)$, $(3,0,3,3)$, $(3,1,0,0)$, $(3,2,2,1)$, $(3,3,1,2)$

the limit $ML \to \infty$, we have

$$R_{\text{th}} \geq \mathbb{E}_{\mathbf{D}}\left[r_{\mathbf{d}}\right] ,$$

where the inequality solely comes from the fact that $r_{\mathbf{d}}$ incorporates the rate loss of the index code (still assuming perfect permutation recovery here).

As a final step, we choose the inner code rate R_{i} which minimizes the theoretical read/write cost trade-off, $(c_{\text{w}}, c_{\text{r}})_{\text{th}} \triangleq (R_{\text{th}}^{-1}, cR_{\text{th}}^{-1})$, for the targeted coverage levels.

7.6.3 Index Code

In general, using indices for regrouping reads reduces the overall rate of the system directly by β. Since coding the indices further reduces the overall rate but increases the correct decoding probability, one must carefully design the index rate R_{ix} and, thus, the index code itself. We target index codes with a good edit distance spectrum.[7]

The cardinality of the index code needs to be at least M. Unfortunately, finding good codes which fit into our coding system becomes more difficult for larger M since the theory of error correction in the Levenshtein/edit metric is not as advanced as, e.g., in the Hamming metric. As explained in Section 7.3, to circumvent this challenge, we build the index code by concatenating shorter component codes of smaller cardinality. For very small cardinality, we have obtained codes via an exhaustive graph search algorithm optimizing the code's edit distance spectrum [Sew98; MLW+22]. In Table 7.3, we give two codes of length 4, cardinality 14, and minimum edit distance 3 which we alternate to build an index code. For larger cardinalities, we investigated the non-binary Varshamov-Tenegolts codes (see Construction 3.2, [Ten84]) and 2-fold repetition codes (every input symbol gets repeated once). While the latter two codes can correct a deletion or an insertion, they cannot uniquely correct a substitution.

We evaluate the index codes by simple average mismatched MIRs $\overline{R}_{\text{MI}}$. Formally, for a fixed coding and decoding scheme, for a feasible number of drawing realizations $\mathbf{d}_1, \ldots, \mathbf{d}_\Phi$, we compute

$$\overline{R}_{\text{MI}} \triangleq \frac{1}{\Phi} \sum_{\phi=1}^{\Phi} r_{\mathbf{d}_\phi} .$$

[7] Even though the random offset is also applied in the index region, which destroys the designed edit distance properties, we have observed better performance when using codes optimized for the edit distance instead of other metrics.

We emphasize that the computed value $\overline{R}_{\mathrm{MI}}$ does not imply any achievability guarantee due to the non-ergodic property of the sampling channel for finite M and L (in contrast to the infinite assumption with R_{th}). However, we have observed that $\overline{R}_{\mathrm{MI}}$ correlates well in our parameter regime with more elaborate performance metrics, such as outage probability (which also requires more computations for evaluation analysis).

For each index code $\mathcal{C}_{\mathrm{ix}}$, we search for the decoding threshold δ_{I} by maximizing the average MIR, i.e., we choose

$$\delta_{\mathrm{I}}^{\mathrm{opt}} = \underset{0 \leq \delta_{\mathrm{I}} \leq 1}{\arg\max}\, \overline{R}_{\mathrm{MI}} \,.$$

We remark that optimizing the index threshold δ_{I} does not depend on the coverage when applying no reads clustering before index decoding. Finally, we choose the index code that maximizes $\overline{R}_{\mathrm{MI}}$ with the optimized δ_{I} for the targeted coverage levels.

7.6.4 Post-Assignment Optimization

In the post-assignment, we can control the cardinality of the top candidate list $\mathcal{T}_{\mathbf{y}}$. This parameter controls the trade-off between complexity and performance, as discussed in Section 7.4.4. We limit $|\mathcal{T}_{\mathbf{y}}|$ such that it is constant with respect to N and choose the cardinality where the MIR $\overline{R}_{\mathrm{MI}}$ of the scheme empirically saturates for the targeted coverage levels. As seen in Section 7.7, we choose $|\mathcal{T}_{\mathbf{y}}| = 5$ independently of the scheme. However, we do not claim that this performance saturation holds in general.

7.6.5 Outer Code

We employ protograph-based non-binary SC-LDPC codes as outer codes in our concatenated coding scheme due to their excellent performance in standard transmission models under belief-propagation (BP) window decoding, e.g., over erasure channels [PIS+10], block-fading channels [HLA+14], etc. [FBG+15]. We mitigate the block-fading effects of the channel due to the drawing effect through the interleaving operation since we scramble possible block errors and erasures randomly over the entire outer codeword.

Formally, a protograph is a small multi-edge-type Tanner graph with n_{p} variable-node (VN) types and r_{p} check-node (CN) types. A protograph can be represented by a base matrix

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} & \dots & \mathbf{B}_{1,n_{\mathsf{p}}} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} & \dots & \mathbf{B}_{2,n_{\mathsf{p}}} \\ \vdots & \vdots & \dots & \vdots \\ \mathbf{B}_{r_{\mathsf{p}},1} & \mathbf{B}_{r_{\mathsf{p}},2} & \dots & \mathbf{B}_{r_{\mathsf{p}},n_{\mathsf{p}}} \end{pmatrix},$$

where entry $\mathbf{B}_{i,j}$ is an integer representing the number of edge connections from a type-i CN to a type-j VN. A parity-check matrix $\mathbf{H}$ of an LDPC code can then be constructed by lifting the base matrix $\mathbf{B}$ by replacing each non-zero (zero) $\mathbf{B}_{i,j}$ with a $Q_{\mathsf{p}} \times Q_{\mathsf{p}}$ circulant (zero) matrix with row and column weight equal to $\mathbf{B}_{i,j}$. The resulting lifted parity-check matrix, of dimensions $Q_{\mathsf{p}} r_{\mathsf{p}} \times Q_{\mathsf{p}} n_{\mathsf{p}}$, defines an LDPC code of length $Q_{\mathsf{p}} n_{\mathsf{p}}$ and dimension

at least $Q_{\mathsf{p}}(n_{\mathsf{p}} - r_{\mathsf{p}})$. To construct a non-binary code from the lifted matrix, we randomly assign non-zero entries from $\mathbb{F}_4$ to the edges of the corresponding Tanner graph.

An SC-LDPC code is constructed from a series of o_{sc} (coupling length) Tanner graphs by interconnecting the neighboring m_{sc} (coupling memory) graphs according to a predefined pattern. Constructing an SC-LDPC protograph from a corresponding LDPC protograph $\mathbf{B}$ is done as follows. The protograph $\mathbf{B}$ of the LDPC code is copied o_{sc} times. Then, the edges of each protograph are interconnected to the m_{sc} neighbors. The protograph matrix of an SC-LDPC code then has the form

$$\mathbf{B}_{\text{sc}} = \begin{pmatrix} \mathbf{B}_1 & & & \\ \mathbf{B}_2 & \mathbf{B}_1 & & \\ \vdots & \vdots & \ddots & \\ \mathbf{B}_{m_{\text{sc}}+1} & \mathbf{B}_{m_{\text{sc}}} & \dots & \mathbf{B}_1 \\ & \mathbf{B}_{m_{\text{sc}}+1} & \dots & \mathbf{B}_2 \\ & & \ddots & \vdots \\ & & & \mathbf{B}_{m_{\text{sc}}+1} \end{pmatrix}_{(o_{\text{sc}}+m_{\text{sc}})r_{\mathsf{p}} \times o_{\text{sc}} n_{\mathsf{p}}},$$

where $\sum_{i=1}^{m_{\text{sc}}+1} \mathbf{B}_i = \mathbf{B}$ to maintain the same VN and CN degrees of the underlying block protograph $\mathbf{B}$. The resulting SC-LDPC code is then constructed by lifting and assigning edge weights in the same manner as mentioned earlier.

In detail, we build our coupled codes from regular LDPC protographs with fixed VN degree d_{VN} and CN degree d_{CN} as the underlying base protograph $\mathbf{B}$ and fix the coupling memory $m_{\text{sc}} = d_{\text{VN}} - 1 = 2$ (i.e., implying $d_{\text{VN}} = 3$) such that each memory block $\mathbf{B}_i$ is connected to each of its neighbors exactly once, i.e., $\mathbf{B} = \sum_{i=1}^{d_{\text{VN}}} (1, \dots, 1)$, where $(1, \dots, 1)$ is of length $\frac{d_{\text{CN}}}{d_{\text{VN}}}$. Moreover, we always choose Q_{p} such that one spatial position covers approximately $Q_{\mathsf{p}} \frac{d_{\text{CN}}}{d_{\text{VN}}} \approx 10\,000$ VN positions and, when needed, shorten positions uniformly at random to fit the required blocklength n_{o}. We remark that we limit the rate optimization significantly to the possible coding rates $R_{\text{o}} \approx \frac{1}{2}, \frac{2}{3}, \dots, \frac{8}{9}$. Higher rates and/or better performance could possibly be achieved by considering different ways of coupling or using irregular protographs. We choose to couple regular protographs due to their simple design and promising decoding performance under BP decoding on classical transmission channels. For decoding, we consider sliding window decoding from the front and back simultaneously. We fix the window size to nine spatial positions and limit the BP iterations to five per window position, with ten iterations at the boundaries of the coupled chain when the window is not yet slid fully into the Tanner graph to kick-off decoding [SAB23].

7.7 Simulation Results

We design our coding scheme to be viable for different modes of operation, with a specific focus on three primary coverages: $c = 5$, $c = 8$, and $c = 10$ for both acc-BC and fast-BC (refer to Table 7.2 for IDS error estimates obtained for the two basecalling methods) and fixed $\beta = \frac{\log_2(M)}{L} = 0.135$ ($M = 30\,588$, $L = 110$).

We perform code optimization on `file1` using MIRs following the steps and criteria dis-

cussed in Section 7.6. Then, we analyze our proposed coding scheme on the experimental data for `file3` by means of the information-outage probability for the outer code and FER simulations for the above channel setups. Moreover, the channel estimation for $\texttt{KMER}_k$ is performed using `file1`. Recall that the stored DNA strands in our dataset have a total length of 110 nt plus additionally on each side a 20 nt file-specific primer, and we stored 30 589 (`file3` 30 588) strands per file (more details in Table 7.1). We subsample reads uniformly at random when targeting a specific c such that we mimic the underlying sampling distribution due to possible synthesis, PCR, and/or sequencing bias.

We focus only on using the passed Q-score reads. However, we believe cleverly utilizing the failed Q-score reads could increase the system's performance at no additional cost.

7.7.1 Read/Write Cost Trade-Off

In Figure 7.9, we show the read/write cost trade-off of our designed coding systems (optimized for different coverages c) and compare them to existing experiments (see Section 2.1.2 for a DNA experiment review). Note that we focus on the detailed construction and performance evaluation in the subsequent sections. With the setup for acc-BC (green squared markers), we outperform other nanopore-based and even Illumina-based experiments while ensuring a failure rate of less than one percent (see Figure 7.12). We attribute our gain mostly to the clever combining/clustering techniques of the strands and the overall soft-decision-based decoding algorithms since our channel error rates, IDS errors combined roughly at 4%, are similar (or even worse) to some of the other experiments. Since it is expected that by optimizing the outer code better we could gain in rate, we have included the theoretical read/write cost trade-off of our inner system with an outage probability $q_{\text{out}} = 10^{-3}$ (empty triangle-shaped markers). We believe one could approach these points closer while ensuring a FER of approximately 10^{-3}. Additionally, we have included the read/write cost trade-off for our fast-BC setups. Despite significantly higher IDS error rates (of roughly 10%), we remain competitive—but not better—than most other nanopore experiments.

In our view, the read/write cost trade-off is the best available metric for experimental comparisons because it naturally incorporates the fundamental tension between synthesis and sequencing. However, the values reported in Figure 7.9 should be interpreted with caution for two reasons: First, they do not reflect the differences in FER and the encoding/decoding complexity. Second, the read and write costs are affected by the details of the channel setup, which vary significantly between different approaches. For example, the choice of β, the IDS error rates, and different drawing distributions all naturally require different amounts of redundancy and thus massively influence the read-write cost performance of a system. For example, *Lau-2023* [LCR+23] evaluates only with low $\beta = 0.0088$ ($M = 792$, $L = 114$) or *Erlich-2017* [EZ17] deals with IDS error rates of less than 1%. Additionally, the same can be seen by the different read/write cost trade-offs for our proposed system obtained for acc-BC and fast-BC.

7.7.2 Extended Trace Reconstruction Channel: KMER Decoder

We analyze the proposed $\texttt{KMER}_k$ model for different values of k. In Figure 7.10, we illustrate the achievable rate R_{trace} for the extended trace reconstruction channel (see Section 7.2.3)

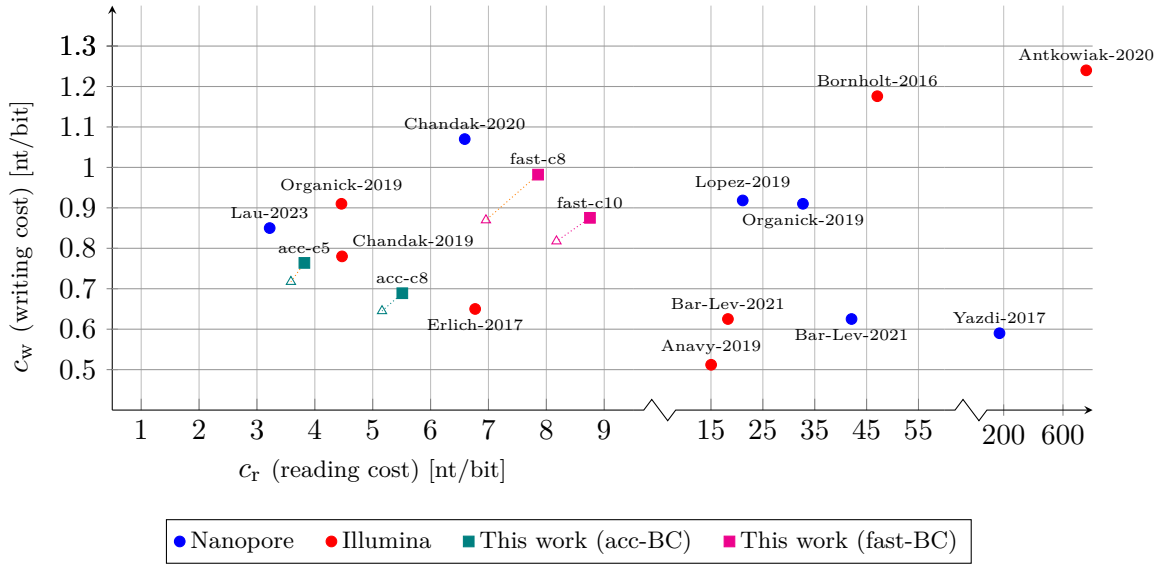


Figure 7.9: Read/write cost trade-off for our setup and other experiments. Some experiments conducted storage trials with various parameters. We have selected the one that appears to be the best based on our knowledge.
Label references: *Lau-2023* [LCR+23], *Bar-Lev-2021* [BLOS+21], *Antkowiak-2020* [ALD+20], *Chandak-2020* [CNT+20], *Anavy-2019* [AVA+19], *Chandak-2019* [CTL+19], *Lopez-2019* [LCA+19], *Organick-2019* [OAC+18], *Erlich-2017* [EZ17], *Yazdi-2017* [YGM17], and *Bornholt-2016* [BLC+16].

as a function of the number of traces T. We perform our analysis using `file3`, where in our computation, we average over all available clusters with at least size T for acc-BC and fast-BC separately (for cluster with larger size we pick randomly T reads). A higher R_{trace} for a fixed setup corresponds to a reduced mismatch with the data's true underlying strand error channel. We observe an increase in R_{trace} once we introduce the $\texttt{KMER}_k$ model for both setups, acc-BC and fast-BC. As expected, this gain diminishes slowly when increasing the number of traces T since the gain in extra physical redundancy dominates. Further, we observe that for the fast basecaller, increasing k yields higher rates; however, this gain is marginal and is associated with a substantial increase in decoding complexity—e.g., changing from $k = 1$ to $k = 3$ increases decoding complexity 16-fold. For the accurate basecaller, we can see no proper performance gain when increasing $k > 1$. In this case, we conjecture that the machine learning-based basecaller, which converts the electric current at the nanopore to nucleotides, already reduces the memory between the neighboring nucleotides. For completeness, we include the curves using the model from [HD23b] to show the performance gain of the slight adaption of the channel model at no additional cost of decoding complexity. Moreover, we have included the curves for i.i.d. decoding with fixed scaling parameter $s = 1.0$ from [MLW+22]. The obtained scaling parameters of this work can be found in Table B.1 in Appendix B.1. Furthermore, we present results for the uncoded traces in Appendix B.2 in Figure B.1. Here, the main takeaway is that introducing the scaling parameter $s(\cdot)$

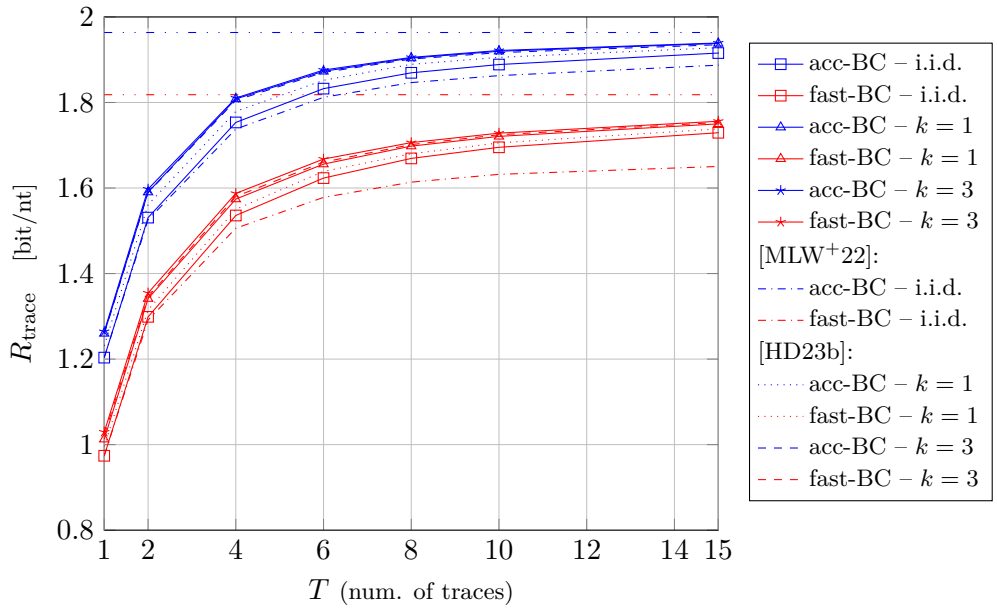


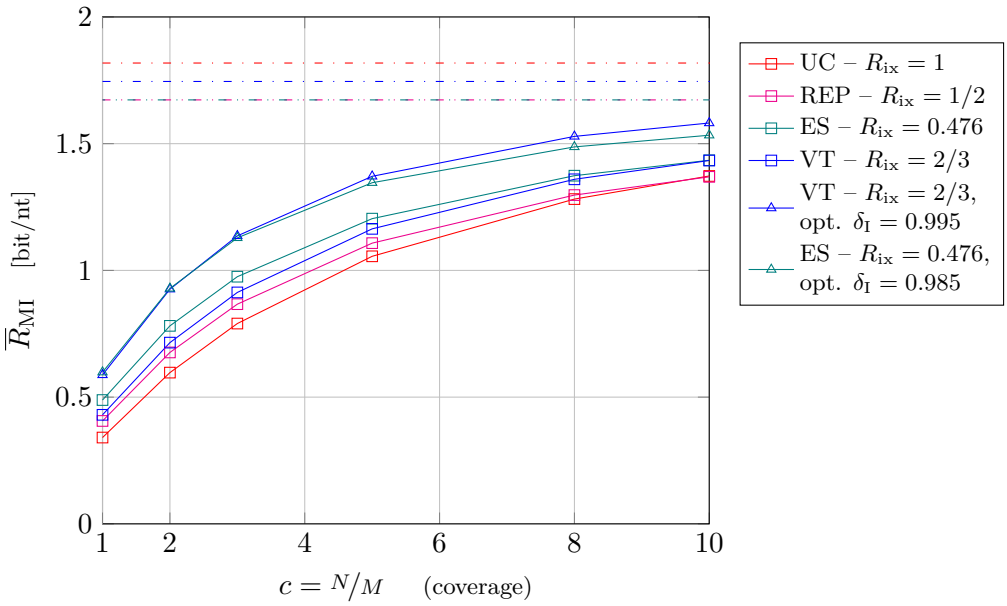
Figure 7.10: BCJR-once rates R_{trace} versus number of traces T for the extended trace reconstruction channel with coded strands obtained on data of `file3`. We compare different single-strand decoders, i.e., different memory k in the $\texttt{KMER}_k$ modeling in the decoder. For acc-BC we use $R_{\text{i}} = {}^{216}/_{110}$ and for fast-BC we use $R_{\text{i}} = {}^{200}/_{110}$. Note that no index part is included here. The horizontal dash-dotted lines represent the rate limit imposed by the inner code rates.

mitigates the APP mismatch significantly for increasing number of traces T.

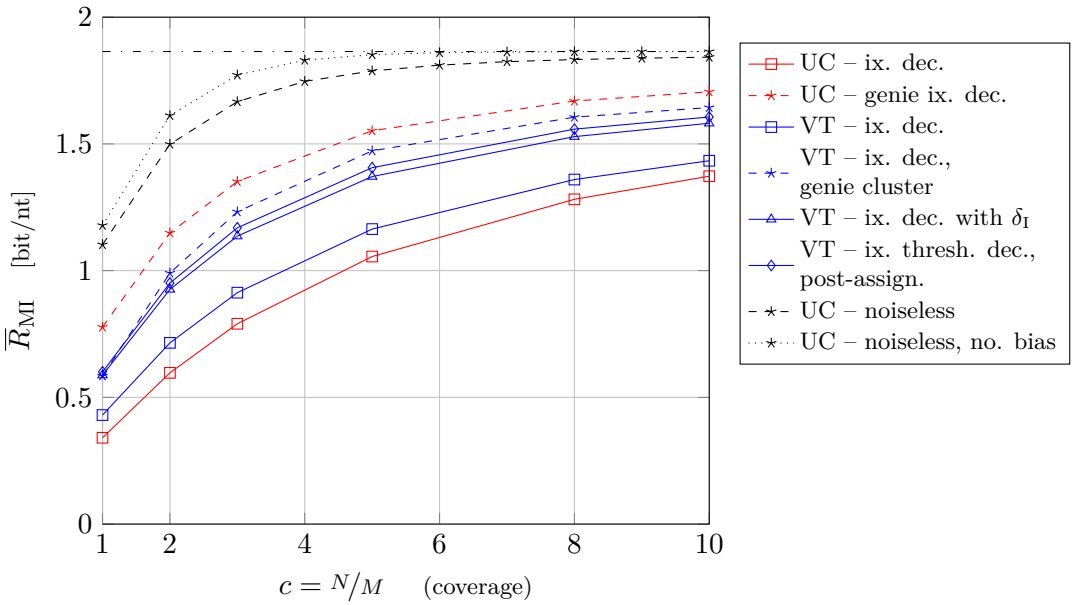
In our optimization procedure, we compute $R_{\text{trace}}(T)$ for various inner rates R_{i} and select the rate that yields the best theoretical read/write cost trade-off $(c_{\text{w}}, c_{\text{r}})_{\text{th}}$ as discussed in Section 7.6.2. We have observed the best performance for $R_{\text{i}} = \frac{216}{110}$ for acc-BC and $R_{\text{i}} = \frac{200}{110}$ for fast-BC. Due to the necessity of the coded address part in the strand of total length 110, the data part is shorter than 110. Thus, we fix the number of redundancy symbols for data protection: two symbols for acc-BC and ten symbols for fast-BC. This effectively introduces a slight discrepancy in the inner code rates for the final setup relative to the values of R_{i} above.

7.7.3 MIR Results: Optimization of the Joint-Code and Decoding Strategy

For the choice of inner code and decoding optimization, we analyze MIRs $\overline{R}_{\text{MI}}$ versus coverage c for different setups using `file1`. As a side-effect, the MIRs highlight the effects of the sampling channel, which we will include in the discussion. In this section, we focus on the results for acc-BC depicted in Figure 7.11; nonetheless, we have included the results for fast-BC in Figure B.2 in Section B.3, where similar conclusions (albeit different outcomes) can be obtained based on the same logic. Throughout this subsection, we fix $k = 1$ in the single-strand joint-code decoder.



(a) We compare different index codes plus the index thresholding strategy. The horizontal dash-dotted lines represent the rate limit imposed by the combined respective index code rates and inner code rates (line colors match the index codes indicated in the legend). We use two redundancy symbols for the data part coding.



(b) We compare different decoding strategies and provide genie-based benchmarks for different scenarios. For the index reliability threshold, we use $\delta_{\mathrm{I}} = 0.995$, and for post-assignment, we use $\delta_{\mathrm{C}} = 44.0$ and $|\mathcal{T}| = 5$. For clarity, the horizontal black dash-dotted line represents the rate limit $(2 - \beta)$ imposed by the index for error-free sampling channel transmission.

Figure 7.11: MIRs $\overline{R}_{\mathrm{MI}}$ versus coverage c for the sampling channel on experimental data of `file1` ($M = 30\,589$, $L = 110$) for acc-BC using $k = 1$ in the single-strand decoder.

Table 7.4: List of Compared Index Codes

Name	R_{ix} [bit/nt]	Component code[a]	a
Uncoded (UC)	2	$[4, 4]_4$	2
Repetition (REP)	1	$[8, 4]_4$	2
Varshamov-Tenengolts (VT)	1.25	$[6, \log_4(178)]_4$	2
Exhaustive-search (ES)[b]	0.952	$[4, \log_4(14)]_4$	4

[a]For an component code $\mathcal{C}'$, we use the notation $[n', k']_{q'}$, where n' is the length, the dimension $k' = \log_4(|\mathcal{C}'|)$, and q' the alphabet size.
[b]For the ES index code, we alternate Code I and Code II from Table 7.3 to increase the inter minimum edit distance between two neighboring codebooks.

First, we compare the different index codes in Figure 7.11(a). Recall that the overall index code is formed from concatenating an even number a of component codes. Moreover, we split the index part equally to be at the strand's front and back. In detail, we compare four index coding strategies: uncoded (UC), repetition (REP) coding, Varshamov-Tenengolts (VT) coded, and exhaustive-search (ES) coded (for the code details see Table 7.4 and Section 7.6.3). For all setups, we use two redundancy symbols for the inner data part coding; hence, we briefly discuss the inner code rate implications using the UC index coding strategy as an example. For UC we need $\lceil \log_4(30\,589) \rceil = 8\,\text{nt}$ of the total 110 nt for the index. Hence, we have an inner data rate of $R_{\text{i}} = \frac{200}{102}$; consequently, imposing a natural limit at $\frac{200}{110}$ (rate limits for each setup are indicated by the dash-dotted lines). Similarly, a VT-coded index needs 12 nt of indexing; thus, combined with having two redundancy symbols for the data part, it imposes a natural limit of $\frac{192}{110}$.

We begin by comparing the index codes using the decoding strategy where the decoder bases its regrouping decisions solely on the soft information of the decoded index (see Section 7.4.3), i.e., no clustering or post-assignment is applied (squared markers). In a general view, we can conclude that although coding induces a rate loss, we obtain higher $\overline{R}_{\text{MI}}$ for the coded setups REP, ES, and VT than for simply UC. Further, for low coverages $c \leq 5$, we obtain a clear ranking in the performance of the studied index coding approaches. We conjecture that the respective edit distance spectrums of the index codes directly impact decoding performance. Codes with the spectrum shifted toward higher edit distances achieve higher $\overline{R}_{\text{MI}}$ even with a higher coding rate R_{ix}. For higher coverages, there is a slight change in the behavior of the codes, e.g., for $c = 10$, UC overtakes REP. This can be explained by the fact that we operate closer to the respective rate limits of each system, e.g., REP's limit is 1.68 bit/nt, while UC's limit is 1.82 bit/nt. Consequently, the choice of the index code depends on the targeted coverage c.

Additionally, for VT and ES index codes, we included in Figure 7.11(a) the curves for an alternative decoding strategy where we assign an index to a specific read if the decoded soft information exceeds a reliability threshold δ_{I} (triangle markers). After optimization of δ_{I}, which turns out to be a convex optimization problem, we acquired for the ES index code a reliability threshold of $\delta_{\text{I}} = 0.985$ and for the VT index code a threshold of $\delta_{\text{I}} = 0.995$. This corresponds to discarding 26% of the reads for the ES strategy and 33.5% for the VT strat-

egy, independent of the coverage c. Due to the performance gain compared to the previous method, we conjecture that, most often, the discarded reads would have been grouped incorrectly since discarding is less detrimental than incorrect index assignments. Surprisingly, the benefit of this method, which comes at no additional decoding cost, remains relatively constant over different coverages (except $c = 1$). We had anticipated that discarding incorrectly assigned reads at low coverage would be significantly more beneficial than at higher coverage. Moreover, we observe that the index code that performs best in the simple index grouping strategy is not the best choice when introducing the reliability threshold δ_{I}, i.e., the VT index code performs better for higher coverages than the ES index code. Be that as it may, the VT index code has a higher rate limit than the ES index code, which might be responsible for the performance difference. In a final conclusion, we have picked the VT index code for the acc-BC setup since it had the best performance for coverages $c = 5$ until $c = 10$ using the reliability threshold $\delta_{\mathrm{I}} = 0.995$.

Next, we compare the decoding strategies by plotting $\overline{R}_{\mathrm{MI}}$ versus coverage c in Figure 7.11(b) using data simulations on `file1`. This plot highlights the different effects of the sampling $\texttt{KMER}_k$ channel—i.e., the full DNA storage channel—nicely. The horizontal dash-dotted line marks the rate limit imposed by indexing and highlights the unavoidable rate loss due to the *permutation* effect. The next two curves (highest to lowest) correspond to the idealized case without strand-level noise (i.e., for which the rate loss is only due to the *permutation* and *sampling* effects). We observe a significant sampling-related rate loss at low coverage (caused primarily by the non-drawn strands); for increasing coverage c, this loss diminishes thanks to the multi-copy gain introduced by sampling. More specifically, the two curves show the $\overline{R}_{\mathrm{MI}}$ of an optimal index-based coding scheme over the sampling channel without strand noise for two drawing distributions: one curve (black, dashed, star markers) uses the drawing distribution based on our dataset, while the other (black, dotted, star markers) uses a uniform distribution $\frac{1}{M}$ with replacement, highlighting the rate loss due to the synthesis/PCR/sequencing bias in terms of drawing distribution (see Figure 7.8 for a drawing distribution comparison). Comparing the two noiseless benchmarks, we conclude that without knowledge about the strand bias at the transmitter, we suffer a direct rate loss due to the sampling bias in any setup. With strand-dependent IDS noise, we include another benchmark for an index-based coding approach given an index genie in the decoding process, i.e., we artificially exclude the permutation loss of the channel (red, dashed, star markers). The rate gap to the noiseless scenario can be explained by the IDS noise and is also due to our chosen sub-optimal coding and mismatched decoding approach, i.e., simply multiplying the symbol APPs to combine reads in the decoding and memoryless outer decoding metric. This curve also functions as a true upper bound for our coding approach. The last benchmark we present uses the VT-coded index and has a clustering genie in the decoder, which groups reads that originate from the identical stored strands together before making index decisions (blue, dashed, star markers). This benchmark provides insight into the performance of our approach when leveraging the multi-copy gain, i.e., APP combining, for the index decisions as well. During optimization, we desire to approach this benchmark as closely as possible. In theory, we could surpass this benchmark, e.g., by trying to discard clusters that would group to the wrong index, which is the dominant effect responsible for the rate loss to the benchmark of the genie index decoding curve.

We can now move on to specific decoding approaches for the empirical channel (Figure 7.11(b), solid curves) instead of the benchmark genie-aided or noise-free setups covered earlier. As discussed before, protecting the index via error-correction mechanisms produces a performance gain (VT, blue, solid, square markers versus UC, red, solid, square markers), whereas disregarding reads with unreliable index information (here, 35% of all reads) boosts the possible system rate even higher (blue, solid, triangle markers). Using the post-assignment strategy of the discarded reads, i.e., assigning them to indices by comparing payload distances (see Section 7.4.4) of discarded reads to the already-grouped clusters based on index decisions, provides an additional performance gain (blue, solid, diamond markers). While the fraction of discarded reads for the simple index threshold strategy is basically constant for all coverages, after the post-assignment step, we discard only 24% of the reads for $c = 1$ and even only 16% for $c = 10$. This could stem from the effect that there are fewer empty pre-grouped clusters for higher coverages, and empty pre-grouped clusters cannot benefit from the post-processing strategy. We have not observed valuable performance gains for top candidate lists of size $|\mathcal{T}| > 5$ for our parameter regimes. Overall, we find a coding/decoding system tackling and leveraging the sampling effect of the channel for the experimental data with acc-BC, which approaches the genie clustering benchmark. We remark that we have deliberately not discussed any clustering approaches before index decoding since we could not find a clustering algorithm with a good complexity/performance trade-off for our number of reads. In particular, we have not found a clustering approach of DNA reads or APPs that did not dominate the system's decoding complexity while providing a performance gain for our coding/decoding setup. One possible approach that could be investigated is clustering exclusively based on the index APP distances.

For fast-BC, we observe the same effects of the channel for different coding/decoding strategies (see Figures B.2(a) and B.2(b) in Section B.3). However, there are stronger changes in performance measured in $\overline{R}_{\mathrm{MI}}$ for different setups. Moreover, there is still a bigger gap between the benchmark of the clustering genie curve and our best setup. The best setup uses an ES-coded index, an inner code rate of $\frac{168}{110}$, an index reliability threshold of $\delta_{\mathrm{I}} = 0.975$, a clustering threshold of $\delta_{\mathrm{C}} = 42.5$, and $|\mathcal{T}_{\mathbf{y}}| = 5$ during post-assignment.

7.7.4 Outage and FER Results

Figure 7.12 shows the information-outage probability q_{out} versus the overall system rate R on our four channel setups using the best respective index/inner coding and decoding scheme as described in the previous subsections simulated on data of `file3` using $\sim$16 000 Monte-Carlo simulations. Recall that the mismatched information-outage probability q_{out} represents a lower bound on the FER for the whole system at the given overall system rate R for any outer code using the decoding metric from Section 7.4.5. We included the value $\overline{R}_{\mathrm{MI}}$ for each channel setup as a vertical line. We can see that for all setups, the waterfall of the outage is very steep. We attribute this to the long blocklengths of $n_{\mathsf{o}} = 2\,569\,392$ nt for fast-BC and $n_{\mathsf{o}} = 2\,936\,448$ nt for acc-BC, the comparatively large numbers of blocks $M = 30\,588$, and the fact that the sampling drawing distribution mimics very closely the population of the drawing distribution for our choices of c and fixed β. Moreover, we infer that using $\overline{R}_{\mathrm{MI}}$ as the code optimization criterion is reasonable for our parameters. For a fixed q_{out}, we achieve the highest possible system rate for the setup with $c = 8$ and acc-BC,

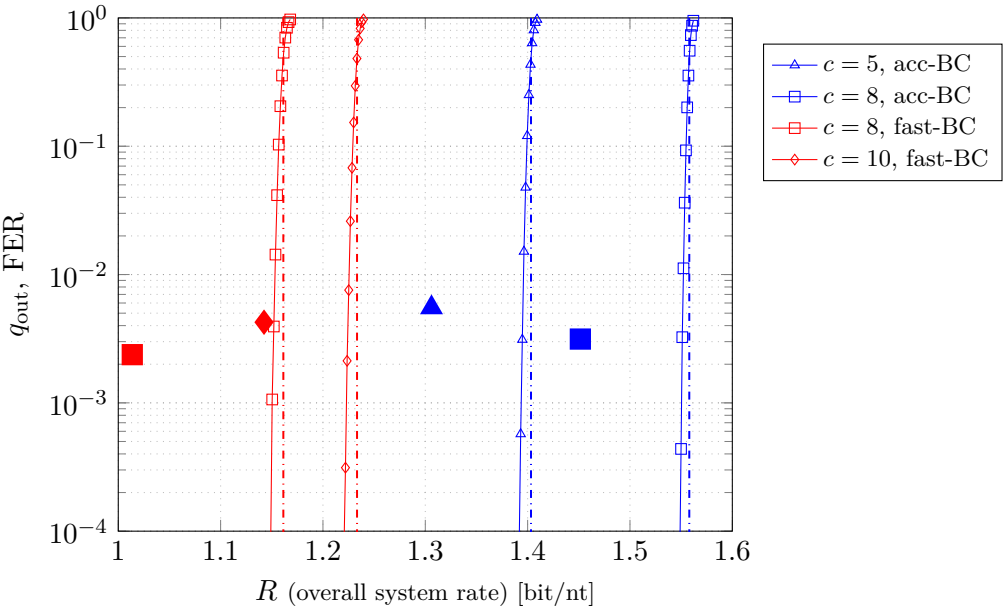


Figure 7.12: Information-outage probability q_{out} and FER versus the overall system rate R for acc-BC and fast-BC and various coverages c on experimental data of `file3` ($M = 30\,588$, $L = 110$). We apply the decoding strategy by pre-assigning reads with the index threshold and applying post-assignment. Parameters for acc-BC: VT-coded index, an inner code with $R_{\text{i}} = {}^{192}/_{110}$, $\delta_{\text{I}} = 0.995$, $\delta_{\text{C}} = 44$, $|\mathcal{T}| = 5$, and $k = 3$. Parameters for fast-BC: ES-coded index, an inner code with $R_{\text{i}} = {}^{168}/_{110}$, $\delta_{\text{I}} = 0.975$, $\delta_{\text{C}} = 42.5$, $|\mathcal{T}| = 5$, and $k = 3$. The vertical lines correspond to the respective $\overline{R}_{\text{MI}}$ for the same setup. The stand-alone markers correspond to the FER performance using an SC-LDPC code ensemble (see Table 7.5 for parameters).

since the IDS error rates are low and the coverage is relatively high.

For the above channel setups, we also include FER results (solid shaped markers, Figure 7.12) for the protograph-based SC-LDPC code ensembles with parameters depicted in Table 7.5. We can ensure a FER around 10^{-3} while the constructed outer codes have an outer code rate R_{o} of 5–10% far from the value given by the outage bound for $q_{\text{out}} \approx 10^{-3}$. (Note that in Figure 7.12, the overall system rate R is used and not the outer code rate R_{o}.) Formally, the results are ensemble averages of the outer code due to the random lifting and edge weight assignment, the random interleaver, and the offset of the joint code (see Section 7.5.2).

Finally, the stored information amount for `file3` would evaluate for acc-BC with $c = 5$ to 549.3 kByte and with $c = 8$ to 610.5 kByte, and for fast-BC with $c = 8$ to 426.5 kByte and with $c = 10$ to 480.5 kByte.

Table 7.5: SC-LDPC Code Parameters[a]

Basecalling	c	R_{o} [bit/nt]	$\mathbf{B}_i$	m_{sc}	o_{sc}	Q_{p}
acc-BC	5	1.497	$(1,1,1,1)$	2	288	2549
acc-BC	8	1.663	$(1,1,1,1,1,1)$	2	192	2549
fast-BC	8	1.327	$(1,1,1)$	2	227	3773
fast-BC	10	1.496	$(1,1,1,1)$	2	252	2549

[a]For all setups, sliding window decoding from front and end simultaneously with 5 BP iterations per window position is performed. Window size is fixed to 9 spatial positions. Double the BP iterations are performed at the front and back until the window is slid fully into the Tanner graph.

7.8 Conclusion

We proposed a competitive end-to-end coding system that can be optimized given the specific DNA storage channel requirements. Besides code design, we presented fitting low-complexity (only linear in the total number of reads N) decoding techniques that push the performance of the proposed system toward its fundamental limits. Further, we have shown that using strand-dependent error modeling in the decoder improves the performance of the system. Finally, we have obtained an explicit construction that outperforms state-of-the-art techniques in terms of read/write cost trade-off with FER of magnitude 10^{-3}, where the results are obtained by simulations on true DNA storage data with a nanopore sequencer.

There are several future interesting research directions. To name a few, we could try to use neural network decoders (similar as in [LCR+23] directly on the signal) or try to reduce the model mismatch at the decoder's side to the true DNA storage error channel by replacing the BCJR decoder with a clever neural network architecture (similar as in [BLOS+21]). A more comprehensive future step would be to design the system for the idea of composite DNA [AVA+19] since this approach seems promising to reduce the writing cost even further by a (possibly) slight increase of the reading cost.

Chapter 8

Concluding Remarks

The scope of the dissertation treats problems related to DNA storage. This molecular storage media promises to solve the world's growing storage demands due to its theoretical opportunities with respect to storage density, long-durability, and energy-efficiency during the storage process.

We began by analyzing DNA storage's main principles and components, highlighting the challenges and opportunities, and reviewing the history of DNA storage experiments with a coding-theoretic focus. After introducing essential coding theory definitions and discussing error types, the dissertation was split into two parts. In the first part, we treated (sub)channels related to DNA storage with a special focus on insertion and deletion errors in the adversarial setting. We analyzed and constructed block codes for the combination of insertion/deletion and substitution errors. Afterward, we generalized the insertion/deletion error-correction problem to the two-dimensional space, considering row and column insertions and deletions in arrays. In the second part, our objective was to construct a practical end-to-end coding system using a probabilistic decoder. We modeled the overall DNA storage process as a sampling noisy channel, where we explicitly consider stochastic strand-dependent noise, which is motivated by nanopore sequencing. Then, we analyzed various aspects of the coding scheme using experimental data to deepen the understanding of the true underlying DNA storage model and to show its excellent performance in comparison to state-of-the-art schemes.

We remark that we discussed interesting future research directions at the conclusions of the respective chapters.

Appendix

Appendix A

Simplified Criss-Cross Model

A.1 Redundancy of the $(1,1)$-Criss-Cross Construction

We will prove in the following the redundancy of the $(1,1)$-criss-cross code $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ construction given in Theorem 5.3 from Section 5.5. Hence, this section will prove Corollary 5.3.

Recall that $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ is given in detail by the intersection of the following sets:

$$\mathcal{U}(a,b) \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad j \in [n-1] \\ \mathbf{X}_{[4],n-1} = (0,0,0,0)^T, \\ \mathbf{X}_{[\ell+1],n} = (0,1,0,1,\ldots)^T, \\ \mathbf{X}_{[\ell],[n]} \in \mathcal{VT}_{a,b}(n,2^\ell) \end{array} \right\},$$

$$\mathcal{V}(c,d) \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{X}_{i,[n-\ell+1,n]} \neq \mathbf{X}_{i+1,[n-\ell+1,n]}, \quad i \in [n-1] \\ \mathbf{X}_{[\ell+1],n} = (0,0,0,\ldots)^T, \\ \mathbf{X}^T_{[n-\ell+1,n],n-1} \in \mathcal{VT}_{c,d}(n-1,2^\ell) \end{array} \right\},$$

$$\mathcal{V}'(c,d) \triangleq \left\{ \mathbf{Y} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{Y}_{[\ell+1],n} = (0,1,0,1,\ldots)^T, \\ \mathbf{Y} \oplus \mathbf{W} \in \mathcal{V}(c,d) \end{array} \right\},$$

$$\mathcal{P}_c \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \mathbf{X}_{i,1} = \sum_{j=2}^{n} \mathbf{X}_{i,j}, \quad i = \ell+1,\ldots,n-1 \right\},$$

$$\mathcal{P}_r \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \mathbf{X}_{n,j} = \sum_{i=1}^{n-1} \mathbf{X}_{i,j}, \quad j \in [n] \right\}.$$

The redundancy $r^{\mathrm{cc}}_{(1,1)}(n)$ of $\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$ is given by

$$\begin{aligned} r^{\mathrm{cc}}_{(1,1)}(n) &\triangleq \log_2(2^{n^2}) - \log_2 \left|\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)\right| \\ &= n^2 - \log_2 \left|\mathcal{U}(a,b) \cap \mathcal{V}'(c,d) \cap \mathcal{P}_c \cap \mathcal{P}_r\right| . \end{aligned}$$

Hence, we show that there exist a, b, c, d for which

$$r^{\mathrm{cc}}_{(1,1)}(n) \leq 2n + 4\log_2(n) + 7 + 2\log_2(e).$$

We do so by computing a lower bound on $\log_2 \left|\mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)\right|$. To that end we count the number of $n \times n$ binary arrays that satisfy all the requirements imposed on the codewords $\mathbf{C} \in \mathcal{C}^{\mathrm{cc}}_{(1,1),n}(a,b,c,d)$.

Since the constraints imposed on the codes $\mathcal{U}(a,b) \cap \mathcal{V}'(c,d)$, $\mathcal{P}_c$, and $\mathcal{P}_r$ are disjoint, we have that

$$\begin{aligned} r^{\mathrm{cc}}_{(1,1)}(n) &= r_{\mathcal{U}(a,b)\cap\mathcal{V}'(c,d)} + r_{\mathcal{P}_c} + r_{\mathcal{P}_r} \\ &= r_{\mathcal{U}(a,b)\cap\mathcal{V}'(c,d)} + 2n - \log_2(n) - 1\,. \end{aligned} \tag{A.1}$$

The Equation (A.1) follows from the fact that the $n - \log_2(n) - 1$ bits of $\mathbf{p}_c$ and the n bits of $\mathbf{p}_r$ are fixed to predetermined values (parities).

We now compute an upper bound on the redundancy of the set $\mathcal{U}(a,b) \cap \mathcal{V}'(c,d)$.

Proposition A.1. *There exists four values $a^\star, b^\star, c^\star$ and $d^\star$ for which the redundancy of $\mathcal{U}(a^\star, b^\star) \cap \mathcal{V}'(c^\star, d^\star)$ is bounded from above by*

$$r_{\mathcal{U}(a^\star,b^\star)\cap\mathcal{V}'(c^\star,d^\star)} < (2n - 2\log_2(n) - 3)\log_2\left(\frac{n}{n-1}\right) + 5\log_2(n) + 6\,.$$

Hence, by combining the results from Equation (A.1) and Proposition A.1 we obtain,

$$\begin{aligned} r^{\mathrm{cc}}_{(1,1)}(n) &< (2n - 2\log_2(n) - 3)\log_2\left(\frac{n}{n-1}\right) + 2n + 4\log_2(n) + 5 \\ &< 2n\log_2\left(\frac{n}{n-1}\right) + 2n + 4\log_2(n) + 5 \\ &\overset{\text{(a)}}{<} 2n + 4\log_2(n) + 5 + 2\log_2 2e \\ &= 2n + 4\log_2(n) + 7 + 2\log_2 e\,. \end{aligned}$$

In Step (a) we use the inequality $2n\log_2\left(\frac{n}{n-1}\right) \leq 2\log_2 2e$. This inequality follows from noting that $\left(1 - \frac{1}{n}\right)^n$ is an increasing function of n that converges to $\frac{1}{e}$ and is always greater than or equal to $0.25 > \frac{1}{2e} = 0.1852$ for $n \geq 2$. The proof of Corollary 5.3 is now complete.

We continue by presenting the proof of Proposition A.1.

Proof of Proposition A.1. We start with counting the number of arrays that satisfy all the imposed constraints except for the VT constraints in the codes $\mathcal{U}(a^\star, b^\star)$ and $\mathcal{V}'(c^\star, d^\star)$. To

that end, we define the following three sets over $\Sigma_2^{n\times n}$:

$$\mathcal{U}_\perp \triangleq \left\{\mathbf{X}\in\Sigma_2^{n\times n} : \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad j\in[n-\ell-1]\right\},$$

$$\mathcal{V}_\perp \triangleq \left\{\mathbf{X}\in\Sigma_2^{n\times n} : \begin{array}{l}\mathbf{X}_{i,[n-\ell+1,n]} \neq \mathbf{X}_{i+1,[n-\ell+1,n]}, \quad \ell < i < n-1\\ \mathbf{X}_{\ell+1,n} \equiv \ell \mod 2\end{array}\right\},$$

$$\mathcal{S}_\cap \triangleq \left\{\mathbf{X}\in\Sigma_2^{n\times n} : \begin{array}{l}\mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad n-\ell \leq j < n,\\ \mathbf{X}_{i,[n-\ell+1,n]} \neq \mathbf{X}_{i+1,[n-\ell+1,n]}, i\in[\ell],\\ \mathbf{X}_{[4],n-1} = (0,0,0,0)^T,\\ \mathbf{X}_{[\ell],n} = (0,1,0,1,0,1,\ldots)^T\end{array}\right\}.$$

$\mathcal{U}_\perp$ is the set of all $n\times n$ arrays in which any two consecutive columns, from column 1 to $n-\ell$, are different when restricted to the first ℓ entries; $\mathcal{V}_\perp$ is the the set of all $n\times n$ arrays in which the entry $X_{\ell+1,n}$ is fixed to a predetermined value and any two consecutive rows, from row $\ell+1$ to $n-1$, are different when restricted to the last ℓ entries; and $\mathcal{S}_\cap$ is the set of $n\times n$ arrays in which the $\ell\times\ell$ subarray ending at the last bit of the first row of the original array has distinct consecutive columns, distinct consecutive rows, the last row fixed to a predetermined value and the first 4 bits of the second to last column are also predetermined. The set $\mathcal{S}_\cap$ is also defined to guarantee that the first column of the $\ell\times\ell$ subarray is different from the ℓ entries of column $n-\ell$ and similarly to the last row.

Claim A.1. *The redundancies of $\mathcal{U}_\perp$ and $\mathcal{V}_\perp$ are respectively given by*

$$r_{\mathcal{U}_\perp} = (n-\log_2(n)-1)\log_2\left(\frac{n}{n-1}\right),$$
$$r_{\mathcal{V}_\perp} = (n-\log_2(n)-2)\log_2\left(\frac{n}{n-1}\right)+1.$$

The intuition behind Claim A.1 is that the first $\log_2(n)$ bits of any two consecutive columns of $\mathbf{U}$ (last $\log_2(n)$ bits of any two consecutive rows of $\mathbf{V}$) must be different.

Claim A.2. *The redundancy of $\mathcal{S}_\cap$ is upper bounded by*

$$r_{\mathcal{S}_\cap} < \log_2(n)+5.$$

The intuition behind Claim A.2 is that with at most one bit of redundancy we can guarantee that every two consecutive rows and every to consecutive columns of the $\log_2(n)\times\log_2(n)$ square are different. The remaining $\log_2(n)+4$ bits are due to the use of the alternating sequence and fixing four bits of the second to last column of the square. The proofs of Claims A.1 and A.2 will be given after the end of this proof.

The remaining part of the proof is to count the number of arrays that satisfy the above requirements and have $\mathbf{U}\in\mathcal{VT}_{a,b}(n,n)$ and $\mathbf{V}\in\mathcal{VT}_{c,d}(n-1,n)$. The VT constraints partition the set $\mathcal{U}_\perp\cap\mathcal{V}_\perp\cap\mathcal{S}_\cap$ into $(n^3)(n-1)$ disjoint cosets. Therefore, there exist $a^\star,d^\star,c^\star,d^\star$

for which

$$|\mathcal{U}(a^\star, b^\star) \cap \mathcal{V}(c^\star, d^\star)| \geq \frac{|\mathcal{U}_\perp \cap \mathcal{V}_\perp \cap \mathcal{S}_\cap|}{(n^3)(n-1)} .$$

In other words, the redundancy $r_{\mathcal{U}(a^\star,b^\star)\cap\mathcal{V}'(c^\star,d^\star)}$ is bounded from above by

$$r_{\mathcal{U}(a^\star,b^\star)\cap\mathcal{V}'(c^\star,d^\star)} \leq r_{\mathcal{U}_\perp\cap\mathcal{V}_\perp\cap\mathcal{S}_\cap} + \log_2\Big((n^3)(n-1)\Big) .$$

Since all the constraints in $\mathcal{U}_\perp$, $\mathcal{V}_\perp$, $\mathcal{S}_\cap$ are disjoint by construction, we can rewrite the previous equation as

$$\begin{aligned} r_{\mathcal{U}(a^\star,b^\star)\cap\mathcal{V}'(c^\star,d^\star)} &\leq r_{\mathcal{U}_\perp} + r_{\mathcal{V}_\perp} + r_{\mathcal{S}_\cap} + \log_2\Big(n^3(n-1)\Big) \\ &< r_{\mathcal{U}_\perp} + r_{\mathcal{V}_\perp} + r_{\mathcal{S}_\cap} + 4\log_2(n) \\ &\overset{\text{(b)}}{\leq} (2n - 2\log_2(n) - 3)\log_2\left(\frac{n}{n-1}\right) + 5\log_2(n) + 6 . \end{aligned}$$

In Step (b) we substituted the results from Claims A.1 and A.2. □

It remains to show the proofs of Claims A.1 and A.2.

Proof of Claim A.1. Remember that we have defined $\ell = \log_2(n)$. We first show that the redundancy of $\mathcal{U}_\perp$ is given by

$$r_{\mathcal{U}_\perp} = (n - \log_2(n) - 1)\log_2\left(\frac{n}{n-1}\right) .$$

Recall that $\mathcal{U}_\perp$ is defined as the set of all $n \times n$ arrays in which any two consecutive columns, from column 1 to $n - \ell$, are different when restricted to the first ℓ entries, i.e.,

$$\mathcal{U}_\perp \triangleq \Big\{\mathbf{X} \in \Sigma_2^{n\times n} : \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad j \in [n-\ell-1]\Big\} .$$

We count the number of arrays that satisfy those constraints. The first ℓ entries of the first column can take 2^ℓ different values. For every other column from 2 to $n - \ell$, the first ℓ entries can take $2^\ell - 1$ different values because they have to be different from the entries of the column before. All other entries have no constraints and can take $2^{n^2-\ell(n-\ell)}$ values. We can then write,

$$\begin{aligned} |\mathcal{U}_\perp| &= 2^\ell(2^\ell - 1)^{n-\ell-1}2^{n^2-\ell(n-\ell)} \\ &= 2^{n^2}2^{-(n-\ell-1)\ell}(2^\ell - 1)^{n-\ell-1} \\ &= 2^{n^2}(1 - 2^{-\ell})^{n-\ell-1} . \end{aligned}$$

Thus, the redundancy can be computed as

$$\begin{aligned} r_{\mathcal{U}_\perp} &= n^2 - \log_2 |\mathcal{U}_\perp| \\ &= -(n - \log_2(n) - 1) \log_2 \left(1 - \frac{1}{n}\right) \\ &= (n - \log_2(n) - 1) \log_2 \left(\frac{n}{n-1}\right) . \end{aligned}$$

To complete the proof we need to show that

$$r_{\mathcal{V}_\perp} = (n - \log_2(n) - 2) \log_2 \left(\frac{n}{n-1}\right) + 1 .$$

Recall that $\mathcal{V}_\perp$ is defined as the set of all $n \times n$ arrays in which the entry $\mathbf{X}_{\ell+1,n}$ is fixed to a predetermined value and any two consecutive rows from row $\ell + 1$ to $n - 1$ are different when restricted to the last ℓ entries, i.e.,

$$\mathcal{V}_\perp \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n \times n} : \begin{array}{ll} \mathbf{X}_{i,[n-\ell+1,n]} \neq \mathbf{X}_{i+1,[n-\ell+1,n]}, & \ell < i < n-1 \\ \mathbf{X}_{\ell+1,n} \equiv \ell \mod 2 & \end{array} \right\} .$$

We count the number of arrays that satisfy those constraints. The last ℓ entries of row $\ell+1$ can take $2^{\ell-1}$ different values, because $\mathbf{X}_{\ell+1,n}$ is predetermined. For every other row from $\ell + 2$ to $n - 1$, the last ℓ entries can take $2^\ell - 1$ different values because they have to be different from the entries of the row before. All other bits have no constraints and can take $2^{n^2-\ell(n-\ell-1)}$ values. We can then write,

$$\begin{aligned} |\mathcal{V}_\perp| &= 2^{\ell-1}(2^\ell - 1)^{n-\ell-2} 2^{n^2-\ell(n-\ell-1)} \\ &= 2^{n^2} 2^{-(n-\ell-2)\ell} (2^\ell - 1)^{n-\ell-2} 2^{-1} \\ &= 2^{n^2} (1 - 2^{-\ell})^{n-\ell-2} 2^{-1} . \end{aligned}$$

The redundancy can then be computed as

$$\begin{aligned} r_{\mathcal{V}_\perp} &= n^2 - \log_2 |\mathcal{V}_\perp| \\ &= -(n - \log_2(n) - 2) \log_2 \left(1 - \frac{1}{n}\right) + 1 \\ &= (n - \log_2(n) - 2) \log_2 \left(\frac{n}{n-1}\right) + 1 . \end{aligned}$$

□

Next we prove Claim A.2, i.e. we show that the redundancy of $\mathcal{S}_\cap$ is upper bounded by

$$r_{\mathcal{S}_\cap} < \log_2(n) + 5.$$

Proof of Claim A.2. Recall that $\mathcal{S}_\cap$ is defined as the set of $n \times n$ arrays in which the $\ell \times \ell$ subarray ending at the last bit of the first row of the original array has distinct consecutive

columns, distinct consecutive rows, the last row fixed to a predetermined value and the first 4 bits of the second to last column are also predetermined. $\mathcal{S}_\cap$ also guarantees that the first column of the $\ell \times \ell$ subarray is different from the ℓ entries of column $n-\ell$ and similarly to the last row., i.e.,

$$\mathcal{S}_\cap \triangleq \left\{ \mathbf{X} \in \Sigma_2^{n\times n} : \begin{array}{l} \mathbf{X}_{[\ell],j} \neq \mathbf{X}_{[\ell],j+1}, \quad n-\ell \leq j < n, \\ \mathbf{X}_{i,[n-\ell+1,n]} \neq \mathbf{X}_{i+1,[n-\ell+1,n]}, i \in [\ell], \\ \mathbf{X}_{[4],n-1} = (0,0,0,0)^T, \\ \mathbf{X}_{[\ell],n} = (0,1,0,1,0,1,\ldots)^T \end{array} \right\}.$$

Let $\mathcal{S}_{c,r}$ be the set of arrays that have different consecutive columns and different consecutive rows. $\mathcal{S}_\cap$ is the intersection between $\mathcal{S}_{c,r}$ and the set of all arrays that have the first ℓ entries of the last column for an alternating sequence and the first four entries of the second to last columns fixed to 0. We shall prove in the sequel that $|\mathcal{S}_{c,r}| > 2^{\ell^2-1}$. Once we have this bound, we can write

$$|\mathcal{S}_\cap| \geq \frac{|\mathcal{S}_{c,r}|}{2^\ell 2^4} > \frac{2^{\ell^2}}{2^\ell 2^5}. \tag{A.2}$$

The first inequality follows from the fact that fixing the last column to a predetermined value reduces the number of arrays in $\mathcal{S}_\cap$ by at most 2^ℓ arrays and fixing 4 bits of the second to last column reduces the number of arrays by at most 2^4. Therefore, using Equation (A.2) we have

$$r_{\mathcal{S}_\cap} = \ell^2 - |\mathcal{S}_\cap| \leq \ell + 5 < \log_2(n) + 5.$$

The remainder of the proof is to show that $|\mathcal{S}_{c,r}| \geq 2^{\ell^2-1}$. We start by showing that the number of $\ell\times\ell$ arrays is lower bounded by 2^{ℓ^2-1}. This means that with one bit of redundancy we can guarantee the constraints on the rows and columns.

To that end, we count the number of arrays that have at least two identical consecutive columns. Let j and $j+1$, $j = n-\ell,\ldots,n-1$, be the indices of two identical consecutive columns. Column j can take $2^\ell - 1$ possible values and column $j+1$ can only take one value. Not imposing any constraints on the other $(\ell-2)$ columns, each column can have 2^ℓ values and we have $(\ell-1)$ possible values for j. Therefore, the number of arrays having at least two identical consecutive columns is $(\ell-1)(2^\ell-1)(2^\ell)^{\ell-2}$. Following the same counting argument, the number of $\ell\times\ell$ arrays that have at least two identical consecutive rows is $(\ell-1)2^\ell(2^\ell)^{\ell-2}$.

The number of arrays in $\mathcal{S}_{c,r}$ is lower bounded by the total number of $\ell\times\ell$ arrays minus the number of arrays that have at least two identical consecutive columns and minus the number of arrays that have at least two identical consecutive rows. Thus, we can write

$$\begin{aligned} |\mathcal{S}_{c,r}| &\geq 2^{\ell^2} - 2(\ell-1)(2^\ell-1)(2^\ell)^{\ell-2} \\ &\overset{(c)}{>} 2^{\ell^2} - 2(\ell-1)2^\ell(2^\ell)^{\ell-2} \\ &\overset{(d)}{\geq} 2^{\ell^2-1}. \end{aligned}$$

The inequality in Step (c) follows from

$$2(\ell-1)(2^\ell-1)(2^\ell)^{\ell-2} < 2(\ell-1)2^\ell(2^\ell)^{\ell-2}\,,$$

which is true because $2^\ell - 1 < 2^\ell$. The inequality in Step (d) follows from

$$2(\ell-1)2^{-\ell} \leq \frac{1}{2}\,,$$

which is equivalent to $4(\ell-1) \leq 2^\ell$ and is true for all $\ell \geq 3$. □

Appendix B

Supplementary Information for the End-To-End Coding Scheme

In this appendix chapter, we provide supplementary information about optimized hyperparameters and further numerical results for the end-to-end coding scheme discussed in Chapter 7. First, in Section B.1, we provide the scaling parameters $s(T)$ for the mutli-trace combining discussed in Section 7.4. Then, we present numerical results for uncoded traces on the extended trace reconstruction channel evaluated on experimental data in Section B.2.

B.1 Scaling Parameter $s(T)$

We obtain the hyperparameters $s(T)$ by maximizing the BCJR-once rate on the extended trace reconstruction channel (see Section 7.2.3) using experimental data as discussed in Section 7.6.2. We obtain the values depicted in Table B.1.

Table B.1: Scaling parameter $s(T)$ for acc-BC with $R_\mathrm{i} = {}^{216}/_{110}$ and fast-BC with $R_\mathrm{i} = {}^{200}/_{110}$ when combining T traces. Intermediate (non-depicted) values of $s(T)$ are obtained via interpolation.

T	1	2	4	6	8	10	15	20	30	50
$s(T)$ for acc-BC	1.00	0.90	0.75	0.65	0.57	0.52	0.42	0.35	0.26	0.18
$s(T)$ for fast-BC	1.01	0.88	0.69	0.57	0.49	0.43	0.33	0.27	0.20	0.13

B.2 Numerical Results for Uncoded Traces Given a Non-I.I.D. Decoder

We analyze our $\texttt{KMER}_k$ decoding model by evaluting it on the extended trace reconstruction model (see Section 7.2.3) using experimental data. In particular, we conisder uncoded traces ($R_\mathrm{i} = 1$) in Figure B.1 and plot the BCJR-once rates R_trace versus number of traces T. We see that introducing the scaling parameters $s(T)$ significantly improves the performance, already

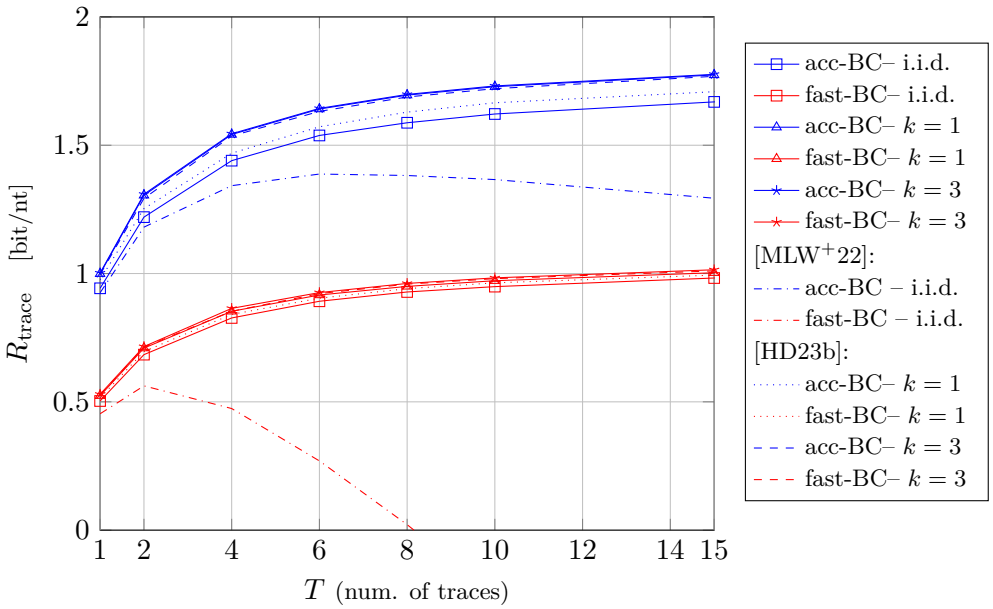
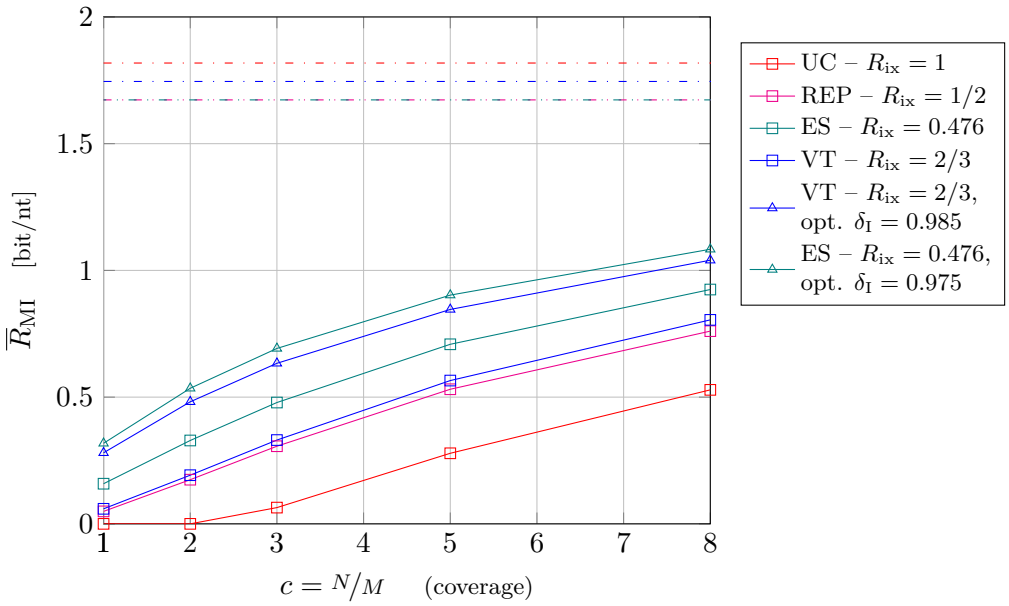


Figure B.1: BCJR-once rates R_{trace} versus number of traces T with for the extended trace reconstruction channel with uncoded strands obtained on data of `file3`.

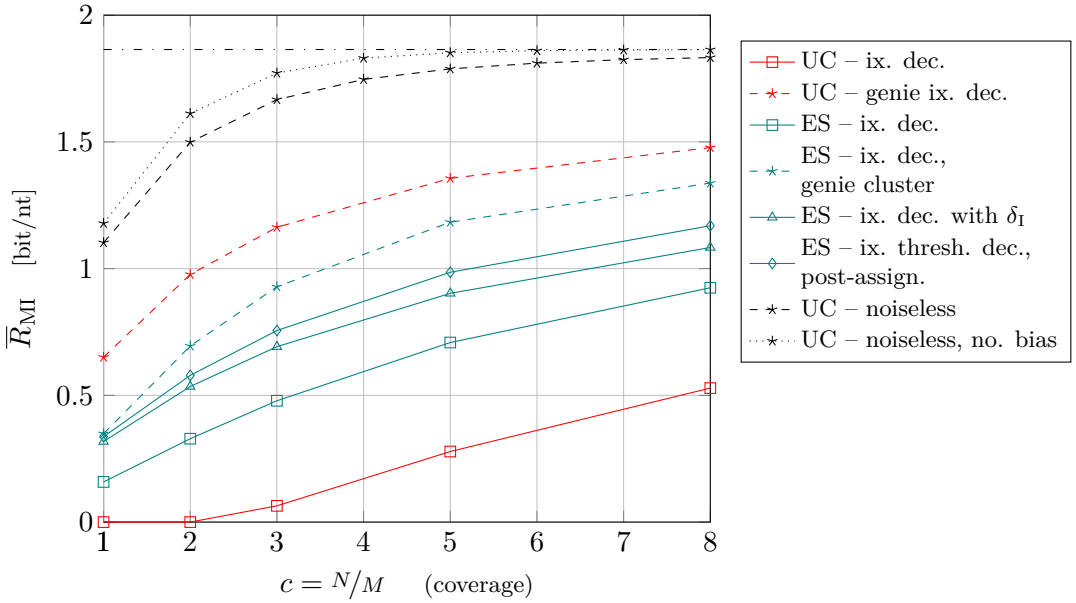
in the decoding with the i.i.d. model in the decoder (see Section 6.2.2). However, with the $\texttt{KMER}_k$ model, we are able to improve R_{trace} even further, reducing the model mismatch to the unknown underlying data model. Note that we provide a detailed discussion about all effects in Section 7.7.2 with coded input data.

B.3 MIR Results on the Sampling Model on Experimental Data Obtained by Fast Basecalling

In Figure B.2, we compare different choices of inner code and decoding strategies by analyzing MIRs $\overline{R}_{\text{MI}}$ versus coverage c on the sampling channel using experimental data obtained via fast-BC. We deduce that in presence of higher error rates, the $\overline{R}_{\text{MI}}$ is lower than for acc-BC for all coverages. It seems crucial to use a coded index as can be seen by the curves with the uncoded index (UC), especially for low coverages. The best choice of the inner code is the index code obtained by exhaustive-search (ES) (see Table 7.4). For a detailed discussion on the effects due to the sampling nature of DNA storage, we refer to the discussion in Section 7.7.3 for acc-BC.



(a) We compare different index codes plus the index thresholding strategy. The horizontal dash-dotted lines represent the rate limit imposed by the combined respective index code rates and inner code rates (line colors match the index codes indicated in the legend). We use ten redundancy symbols for the data part coding.



(b) We compare different decoding strategies and provide genie-based benchmarks for different scenarios. For the index reliability threshold, we use $\delta_{\mathrm{I}} = 0.975$, and for post-assignment, we use $\delta_{\mathrm{C}} = 42.5$ and $|\mathcal{T}| = 5$. For clarity, the horizontal black dash-dotted line represents the rate limit $(2 - \beta)$ imposed by the index for error-free sampling channel transmission.

Figure B.2: MIRs $\overline{R}_{\mathrm{MI}}$ versus coverage c for the sampling channel on experimental data of `file1` ($M = 30\,589$, $L = 110$) for fast-BC using $k = 1$ in the single-strand decoder.

List of Illustrations

List of Algorithms

List of Figures

List of Tables

Nomenclature and Abbreviations

Basic Sets

$\mathbb{Z}$	Set of all integers
$\mathbb{N}, \mathbb{N}_0$	Set of all positive integers, with $\mathbb{N}_0$ including zero
$\mathbb{R}, \mathbb{R}^+$	Set of all real numbers, with $\mathbb{R}^+$ for positive real numbers
Σ_q	Alphabet of size q with letters $\Sigma_q = \{0, 1, \ldots, q-1\}$
$\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$	DNA nucleotides: Adenine (A), Cytosine (C), Guanine (G), Thymine (T)
Σ_{DNA}	DNA alphabet with $\Sigma_{\text{DNA}} = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$
$\mathbb{F}_q$	Finite field of size q
$[a, b]$	Set of integers $\{i \mid a \leq i \leq b\}$
$[b]$	Set of integers $\{i \mid 1 \leq i \leq b\}$
$\mathcal{A}$	Generic set
$\lvert\mathcal{A}\rvert$	Cardinality of a set $\mathcal{A}$

Vectors and Arrays

$\mathbf{x}$	Vectors
x_i	i^{th} element of a vector $\mathbf{x}$
$(\mathbf{x} \parallel \mathbf{y})$	Concatenation of vector $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{A}$	Matrix/Array
$\mathbf{A}_{i,j}$	Element in i^{th} row and j^{th} column of $\mathbf{A}$
$\mathbf{A}^T$	Transpose of an array
$\det(\mathbf{A})$	Determinant of $\mathbf{A}$
Σ_q^n	Set of vectors of length n with elements in Σ_q
$\Sigma_q^\star$	Set of vectors of arbitrary length in Σ_q
$\mathbb{F}_q^n$	Vector space over $\mathbb{F}_q$ of length n
$\Sigma_q^{n \times n}$	Set of arrays with elements of Σ_q with dimension $n \times n$.
$\mathbb{F}_q^{n \times n}$	Matrix space over $\mathbb{F}_q$ of dimension $n \times n$

Distance Metrics

$\mathrm{d}_{\mathsf{H}}(\mathbf{x}, \mathbf{y})$	Hamming distance
$\mathrm{d}_{\mathsf{L}}(\mathbf{x}, \mathbf{y})$	Levenshtein distance
$\mathrm{d}_{\mathsf{E}}(\mathbf{x}, \mathbf{y})$	Edit distance

Codes

$\mathcal{C}$	Code over some space
$[n,k]_q$	Code of length n and dimension k over the alphabet q
r	redundancy of a code
R	rate of a code
$\mathcal{VT}_{a,b}(n,q)$	Non-binary Varshamov-Tenengolts codes (binary: $\mathcal{VT}_a(n)$)

Random processes

$\mathsf{X}, \mathbf{X}$	Random variable/vector
$\mathbb{P}(\mathsf{X})$	Probability mass function of discrete random variable X
$\mathbb{P}(x)$	Probability of the event x, i.e, short for $\mathbb{P}(\mathsf{X} = x)$
$\mathbb{E}_{\mathsf{X}}[f(\mathsf{X})]$	Expectation of function $f(\mathsf{X})$ with respect to $\mathbb{P}(\mathsf{X})$
$\mathbb{V}_{\mathsf{X}}[f(\mathsf{X})]$	Variance of function $f(\mathsf{X})$ with respect to $\mathbb{P}(\mathsf{X})$
$\mathrm{H}(\mathsf{X})$	Entropy of random variable X
$\mathrm{I}(\mathsf{X};\mathsf{Y})$	Mutual information of random variables X, Y
$\mathsf{Unif}(\Sigma_q)$	Uniform distribution over the set Σ_q with probability $1/\lvert\Sigma_q\rvert$

Abbreviations

DNA	deoxyribonucleic acid
RV	random variable/vector
insdel	insertion and/or deletion
edit	insertion, deletion, or substitution
VT	Varshamov-Tenengolts
APP	a posteriori probability
ML	maximum likelihood
MAP	maximum a posteriori
s-MAP	symbolwise maximum a posteriori
MIR	mutual information rate
IDS	insertion/deletion/substitution
nt	nucleotide
i.i.d.	independent and identically distributed
PCR	polymerase chain reaction
FER	frame-error-rate
HMM	hidden Markov model
AIR	achievable information rate
LDPC	low-density parity-check code
SC-LDPC	spatially-coupled low-density parity-check code

Related Publications by the Author

[BWS+21] R. Bitar, L. Welter, I. Smagloy, A. Wachter-Zeh, and E. Yaakobi. "Criss-Cross Insertion and Deletion Correcting Codes". In: *IEEE Transactions on Information Theory* 67.12 (2021), pp. 7999–8015.

[SWB+22] E. Stylianou, L. Welter, R. Bitar, A. Wachter-Zeh, and E. Yaakobi. "Equivalence of Insertion/Deletion Correcting Codes for d-Dimensional Arrays". In: *Proceedings of 2022 IEEE International Symposium on Information Theory*. IEEE. 2022, pp. 814–819.

[SWWZ+23] I. Smagloy, L. Welter, A. Wachter-Zeh, and E. Yaakobi. "Single-Deletion Single-Substitution Correcting Codes". In: *IEEE Transactions on Information Theory* (2023).

[WBWZ+22] L. Welter, R. Bitar, A. Wachter-Zeh, and E. Yaakobi. "Multiple Criss-Cross Insertion and Deletion Correcting Codes". In: *IEEE Transactions on Information Theory* 68.6 (2022), pp. 3767–3779.

[WML+23] L. Welter, I. Maarouf, A. Lenz, A. Wachter-Zeh, E. Rosnes, and A. Graell i Amat. "Index-Based Concatenated Codes for the Multi-Draw DNA Storage Channel". In: *Proceedings of 2023 IEEE Information Theory Workshop (ITW)*. IEEE. 2023, pp. 383–388.

Related Preprints by the Author

[SW24a] R. Sokolovskii and L. Welter. *Experimental data for "An End-to-End Coding Scheme for DNA-Based Data Storage With Nanopore Sequenced Reads"*. 2024. DOI: 10.5281/ZENODO.10943281. URL: https://zenodo.org/doi/10.5281/zenodo.10943281.

[WSH+24] L. Welter, R. Sokolovskii, T. Heinis, A. Wachter-Zeh, E. Rosnes, and A. Graell i Amat. "An End-to-End Coding Scheme for DNA-Based Data Storage With Nanopore-Sequenced Reads". In: *arXiv preprint arXiv:2406.12955* (2024).

Bibliography

[AGF98] K. A. S. Abdel-Ghaffar and H. C. Ferreira. “Systematic Encoding of the Varshamov-Tenengol’ts Codes and the Constantin-Rao Codes”. In: *IEEE Transactions on Information Theory* 44.1 (1998), pp. 340–345.

[AGM+90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. “Basic Local Alignment Search Tool”. In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410.

[AGPF+11] K. A. S. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke. “On Helberg’s Generalization of the Levenshtein Code for Multiple Deletion/Insertion Error Correction”. In: *IEEE Transactions on Information Theory* 58.3 (2011), pp. 1804–1808.

[ALD+20] P. L. Antkowiak, J. Lietard, M. Z. Darestani, M. M. Somoza, W. J. Stark, R. Heckel, and R. N. Grass. “Low Cost DNA Data Storage Using Photolithographic Synthesis and Advanced Information Reconstruction and Error Correction”. In: *Nature Communications* 11.1 (2020), p. 5345.

[ALV+06] D.-M. Arnold, H.-A. Loeliger, P. O. Vontobel, A. Kavcic, and W. Zeng. “Simulation-Based Computation of Information Rates for Channels with Memory”. In: *IEEE Transactions on Information Theory* 52.8 (2006), pp. 3498–3508.

[AO04] M. Arita and Y. Ohashi. “Secret Signatures Inside Genomic DNA”. In: *Biotechnology Progress* 20.5 (2004), pp. 1605–1607.

[ASY21] M. A. Abu-Sini and E. Yaakobi. “On Levenshtein’s Reconstruction Problem under Insertions, Deletions, and Substitutions”. In: *IEEE Transactions on Information Theory* 67.11 (2021), pp. 7132–7158.

[AVA+19] L. Anavy, I. Vaknin, O. Atar, R. Amit, and Z. Yakhini. “Data Storage in DNA with Fewer Synthesis Cycles Using Composite DNA Letters”. In: *Nature Biotechnology* 37.10 (2019), pp. 1229–1236.

[AVF18] M. Abroshan, R. Venkataramanan, and A. G. i Fàbregas. “Efficient Systematic Encoding of Non-Binary VT Codes”. In: *Proceedings of 2018 IEEE International Symposium on Information Theory*. IEEE. 2018, pp. 91–95.

[AW71] R. Ahlswede and J. Wolfowitz. “Channels Without Synchronization”. In: *Advances in Applied Probability* 3.2 (1971), pp. 383–403.

[Bau95] E. B. Baum. “Building an Associative Memory Vastly Larger Than the Brain”. In: *Science* 268.5210 (1995), pp. 583–585.

[BB11] V. Buttigieg and J. A. Briffa. “Codebook and Marker Sequence Design for Synchronization-Correcting Codes”. In: *Proceedings of 2011 IEEE International Symposium on Information Theory*. IEEE. 2011, pp. 1579–1583.

[BB97] M. Blaum and J. Bruck. “Array Codes for Correction of Criss-Cross Errors”. In: *Proceedings of 1997 IEEE International Symposium on Information Theory*. IEEE. 1997, p. 412.

[BBB+01] C. Bancroft, T. Bowler, B. Bloom, and C. T. Clelland. “Long-Term Storage of Information in DNA”. In: *Science* 293.5536 (2001), pp. 1763–1765.

[BBW14] J. A. Briffa, V. Buttigieg, and S. Wesemeyer. “Time-Varying Block Codes for Synchronization Errors: Maximum a Posteriori Decoder and Practical Issues”. In: *The Journal of Engineering* 2014.6 (2014), pp. 340–351.

[BCF+19] F. Ban, X. Chen, A. Freilich, R. A. Servedio, and S. Sinha. “Beyond Trace Reconstruction: Population Recovery from the Deletion Channel”. In: *Proceedings of 2019 IEEE 60th Annual Symposium on Foundations of Computer Science*. IEEE. 2019, pp. 745–768.

[BCJ+74] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. “Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate (Corresp.)” In: *IEEE Transactions on Information Theory* 20.2 (1974), pp. 284–287.

[BCS+19] F. Ban, X. Chen, R. A. Servedio, and S. Sinha. “Efficient Average-Case Population Recovery in the Presence of Insertions and Deletions”. In: *arXiv preprint arXiv:1907.05964* (2019).

[BE24] S. Bakirtas and E. Erkip. “Database Matching Under Noisy Synchronization Errors”. In: *IEEE Transactions on Information Theory* (2024).

[BF15] V. Buttigieg and N. Farrugia. “Improved Bit Error Rate Performance of Convolutional Codes with Synchronization Errors”. In: *Proceedings of 2015 IEEE International Conference on Communications*. IEEE. 2015, pp. 4077–4082.

[BFC00] B. Brink, H. C. Ferreira, and W. A. Clarke. “Pruned Convolutional Codes for Flexible Unequal Error Protection Against Insertion/Deletion/Reversal Errors”. In: *Proceedings of 2000 IEEE International Symposium on Information Theory*. IEEE. 2000, p. 260.

[BGH+16] M. Blawat, K. Gaedke, I. Huetter, X.-M. Chen, B. Turczyk, S. Inverso, B. W. Pruitt, and G. M. Church. “Forward Error Correction for DNA Data Storage”. In: *Procedia Computer Science* 80 (2016), pp. 1011–1022.

[BGZ17] J. Brakensiek, V. Guruswami, and S. Zbarsky. “Efficient Low-Redundancy Codes for Correcting Multiple Deletions”. In: *IEEE Transactions on Information Theory* 64.5 (2017), pp. 3403–3410.

[BJ75] L. Bahl and F. Jelinek. “Decoding for Channels with Insertions, Deletions, and Substitutions with Applications to Speech Recognition”. In: *IEEE Transactions on Information Theory* 21.4 (1975), pp. 404–411.

[BKK+04] T. Batu, S. Kannan, S. Khanna, and A. McGregor. “Reconstructing Strings from Random Traces”. In: *SODA*. Vol. 4. 2004, pp. 910–918.

[BLC+16] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss. "A DNA-Based Archival Storage System". In: *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems*. 2016, pp. 637–649.

[BLOS+21] D. Bar-Lev, I. Orr, O. Sabary, T. Etzion, and E. Yaakobi. "Deep DNA Storage: Scalable and Robust DNA Storage via Coding Theory and Deep Learning". In: *arXiv preprint arXiv:2109.00031* (2021).

[BLWZ24] A. Banerjee, A. Lenz, and A. Wachter-Zeh. "Sequential Decoding of Multiple Sequences for Synchronization Errors". In: *IEEE Transactions on Communications* (2024).

[BPR+20] V. Bhardwaj, P. A. Pevzner, C. Rashtchian, and Y. Safonova. "Trace Reconstruction Problems in Computational Biology". In: *IEEE Transactions on Information Theory* 67.6 (2020), pp. 3295–3314.

[BPS+70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". In: *The Annals of Mathematical Statistics* 41.1 (1970), pp. 164–171.

[BSB+21] J. L. Banal, T. R. Shepherd, J. Berleant, H. Huang, M. Reyes, C. M. Ackerman, P. C. Blainey, and M. Bathe. "Random Access DNA Memory Using Boolean Search in an Archival File Storage System". In: *Nature Materials* 20.9 (2021), pp. 1272–1280.

[BSW10] J. A. Briffa, H. G. Schaathun, and S. Wesemeyer. "An Improved Decoding Algorithm for the Davey-MacKay Construction". In: *Proceedings of 2010 IEEE International Conference on Communications*. IEEE. 2010, pp. 1–5.

[BV08] D. Buckingham and M. C. Valenti. "The Information-Outage Probability of Finite-Length Codes Over AWGN Channels". In: *Proceedings of 2008 42nd Annual Conference on Information Sciences and Systems*. IEEE. 2008, pp. 390–395.

[BWA+24] A. Banerjee, L. Welter, A. Graell i Amat, A. Wachter-Zeh, and E. Rosnes. "Sequential Decoding of Multiple Traces Over the Syndrome Trellis for Synchronization Errors". In: *arXiv preprint arXiv:2410.07120* (2024).

[BYWZ+24] A. Banerjee, Y. Yehezkeally, A. Wachter-Zeh, and E. Yaakobi. "Error-Correcting Codes for Nanopore Sequencing". In: *IEEE Transactions on Information Theory* (2024).

[CFS06] L. Cheng, H. C. Ferreira, and T. G. Swart. "Bidirectional Viterbi Decoding Using the Levenshtein Distance Metric for Deletion Channels". In: *2006 IEEE Information Theory Workshop*. IEEE. 2006, pp. 254–258.

[CG17] B. Chowdhury and G. Garai. "A Review on Multiple Sequence Alignment from the Perspective of Genetic Algorithm". In: *Genomics* 109.5-6 (2017), pp. 419–431.

[CGK12] G. M. Church, Y. Gao, and S. Kosuri. "Next-Generation Digital Information Storage in DNA". In: *Science* 337.6102 (2012), pp. 1628–1628.

[CH69] L. Calabi and W. E. Hartnett. “Some General Results of Coding Theory with Applications to the Study of Codes for the Correction of Synchronization Errors”. In: *Information and Control* 15.3 (1969), pp. 235–249.

[CHK21] Y. M. Chee, M. Hagiwara, and V. Van Khu. “Two Dimensional Deletion Correcting Codes and Their Applications”. In: *Proceedings of 2021 IEEE International Symposium on Information Theory*. IEEE. 2021, pp. 2792–2797.

[CJL+18] K. Cheng, Z. Jin, X. Li, and K. Wu. “Deterministic Document Exchange Protocols and Almost Optimal Binary Codes for Edit Errors”. In: *Proceedings of 2018 IEEE 59th Annual Symposium on Foundations of Computer Science*. IEEE. 2018, pp. 200–211.

[CK14] D. Cullina and N. Kiyavash. “An Improvement to Levenshtein’s Upper Bound on the Cardinality of Deletion Correcting Codes”. In: *IEEE Transactions on Information Theory* 60.7 (2014), pp. 3862–3870.

[CKN19] Y. M. Chee, H. M. Kiah, and T. T. Nguyen. “Linear-Time Encoders for Codes Correcting a Single Edit for DNA-Based Data Storage”. In: *Proceedings of 2019 IEEE International Symposium on Information Theory*. IEEE. 2019, pp. 772–776.

[CKN+21] K. Cai, H. M. Kiah, T. T. Nguyen, and E. Yaakobi. “Coding for Sequence Reconstruction for Single Edits”. In: *IEEE Transactions on Information Theory* 68.1 (2021), pp. 66–79.

[CL88] H. Carrillo and D. Lipman. “The Multiple Sequence Alignment Problem in Biology”. In: *SIAM Journal on Applied Mathematics* 48.5 (1988), pp. 1073–1082.

[CMC+03] J. Chen, M. Mitzenmacher, N. Chaki, and N. Varnica. “Concatenated Codes for Deletion Channels”. In: *Proceedings of 2003 IEEE International Symposium on Information Theory*. 2003, pp. 218–218.

[CNS19] L. Ceze, J. Nivala, and K. Strauss. “Molecular Digital Data Storage Using DNA”. In: *Nature Reviews Genetics* 20.8 (2019), pp. 456–466.

[CNT+20] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman, et al. “Overcoming High Nanopore Basecaller Error Rates for DNA Storage via Basecaller-Decoder Integration and Convolutional Codes”. In: *Proceedings of 2020 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2020, pp. 8822–8826.

[CR20] M. Cheraghchi and J. Ribeiro. “An Overview of Capacity Results for Synchronization Channels”. In: *IEEE Transactions on Information Theory* 67.6 (2020), pp. 3207–3232.

[CSF+14] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. S. Abdel-Ghaffar. “Codes for Correcting Three or More Adjacent Deletions or Insertions”. In: *Proceedings of 2014 IEEE International Symposium on Information Theory*. IEEE. 2014, pp. 1246–1250.

[CT06] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

[CTL+19] S. Chandak, K. Tatwawadi, B. Lau, J. Mardia, M. Kubit, J. Neu, P. Griffin, M. Wootters, T. Weissman, and H. Ji. "Improved Read/Write Cost Tradeoff in DNA-Based Data Storage Using LDPC Codes". In: *Proceedings of 2019 57th Annual Allerton Conference on Communication, Control, and Computing*. IEEE. 2019, pp. 147–156.

[Dav96] J. Davis. "Microvenus". In: *Art J.* 55.1 (1996), pp. 70–74.

[DM01] M. C. Davey and D. J. C. MacKay. "Reliable Communication Over Channels with Insertions, Deletions, and Substitutions". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 687–698.

[DN21] C. Delahaye and J. Nicolas. "Sequencing DNA with Nanopores: Troubles and Biases". In: *PLOS One* 16.10 (2021), e0257521.

[Dob67] R. L. Dobrushin. "Shannon's Theorems for Channels with Synchronization Errors". In: *Problemy Peredachi Informatsii* 3.4 (1967), pp. 18–36.

[DPG+22] A. Doricchi, C. M. Platnich, A. Gimpel, F. Horn, M. Earle, G. Lanzavecchia, A. L. Cortajarena, L. M. Liz-Marzán, N. Liu, R. Heckel, et al. "Emerging Approaches to DNA Data Storage: Challenges and Prospects". In: *ACS Nano* 16.11 (2022), pp. 17552–17571.

[DSP+20] Y. Dong, F. Sun, Z. Ping, Q. Ouyang, and L. Qian. "DNA Storage: Research Landscape and Future Prospects". In: *National Science Review* 7.6 (2020), pp. 1092–1107.

[EZ17] Y. Erlich and D. Zielinski. "DNA Fountain Enables a Robust and Efficient Storage Architecture". In: *Science* 355.6328 (2017), pp. 950–954.

[FBG+15] Y. Fang, G. Bi, Y. L. Guan, and F. C. M. Lau. "A Survey on Protograph LDPC Codes and Their Applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 1989–2016.

[Fey59] R. P. Feynman. "There's Plenty of Room at the Bottom". In: *Talk at the Meeting of the American Physical Society at the California Institute of Technology* (1959).

[For65] G. D. Forney. *Concatenated Codes*. 1965.

[FVY15] A. Fazeli, A. Vardy, and E. Yaakobi. "Generalized Sphere Packing Bound". In: *IEEE Transactions on Information Theory* 61.5 (2015), pp. 2313–2334.

[FZ99] A. Felström and K. Sh. Zigangirov. "Time-Varying Periodic Convolutional Codes with Low-Density Parity-Check Matrix". In: *IEEE Transactions on Information Theory* 45.6 (1999), pp. 2181–2191.

[Gab85] E. M. Gabidulin. "Theory of Codes with Maximum Rank Distance". In: *Problemy Peredachi Informatsii* 21.1 (1985), pp. 3–16.

[Gal61] R. G. Gallager. *Sequential Decoding for Binary Channel with Noise and Synchronization Errors*. Tech. rep. Arlington, VA, USA, 1961.

[Gal62] R. Gallager. "Low-Density Parity-Check Codes". In: *IRE Transactions on Information Theory* 8.1 (1962), pp. 21–28.

[GBC+13] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney. "Towards Practical, High-Capacity, Low-Maintenance Information Storage in Synthesized DNA". In: *Nature* 494.7435 (2013), pp. 77–80.

[GGL+10] D. G. Gibson, J. I. Glass, C. Lartigue, V. N. Noskov, R.-Y. Chuang, M. A. Algire, G. A. Benders, M. G. Montague, L. Ma, M. M. Moodie, et al. "Creation of a Bacterial Cell Controlled by a Chemically Synthesized Genome". In: *Science* 329.5987 (2010), pp. 52–56.

[GGR+22] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu. "Beyond Single-Deletion Correcting Codes: Substitutions and Transpositions". In: *IEEE Transactions on Information Theory* 69.1 (2022), pp. 169–186.

[GH21] V. Guruswami and J. Håstad. "Explicit Two-Deletion Codes with Redundancy Matching the Existential Bound". In: *IEEE Transactions on Information Theory* 67.10 (2021), pp. 6384–6394.

[GHP+15] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark. "Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes". In: *Angewandte Chemie International Edition* 54.8 (2015), pp. 2552–2555.

[Gil60] E. Gilbert. "Synchronization of Binary Messages". In: *IRE Transactions on Information Theory* 6.4 (1960), pp. 470–477.

[GK18] R. Goto and K. Kasai. "Sparse Graph Codes for Channels with Synchronous Errors". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 101.12 (2018), pp. 2064–2071.

[GMM16] S. Goodwin, J. D. McPherson, and W. R. McCombie. "Coming of Age: Ten Years of Next-Generation Sequencing Technologies". In: *Nature Reviews Genetics* 17.6 (2016), pp. 333–351.

[GP08] E. M. Gabidulin and N. I. Pilipchuk. "Error and Erasure Correcting Algorithms for Rank Codes". In: *Designs, Codes and Cryptography* 49 (2008), pp. 105–122.

[GS18] R. Gabrys and F. Sala. "Codes Correcting Two Deletions". In: *IEEE Transactions on Information Theory* 65.2 (2018), pp. 965–974.

[GSG+24] A. L. Gimpel, W. J. Stark, R. N. Grass, and R. Heckel. "Challenges for Error-Correction Coding in DNA Data Storage: Photolithographic Synthesis and DNA Decay". In: *Digital Discovery* (2024).

[Gus09] C. Gustafsson. "For Anyone Who Ever Said There's No Such Thing as a Poetic Gene". In: *Nature* 458.7239 (2009), pp. 703–703.

[GW17] V. Guruswami and C. Wang. "Deletion Codes in the High-Noise and High-Rate Regimes". In: *IEEE Transactions on Information Theory* 63.4 (2017), pp. 1961–1970.

[GYD15] R. Gabrys, E. Yaakobi, and L. Dolecek. "Correcting Grain-Errors in Magnetic Media". In: *IEEE Transactions on Information Theory* 61.5 (2015), pp. 2256–2272.

[Hae19] B. Haeupler. "Optimal Document Exchange and New Codes for Insertions and Deletions". In: *Proceedings of 2019 IEEE 60th Annual Symposium on Foundations of Computer Science*. IEEE. 2019, pp. 334–347.

[Hag20] M. Hagiwara. "Conversion Method from Erasure Codes to Multi-Deletion Error-Correcting Codes for Information in Array Design". In: *Proceedings of 2020 International Symposium on Information Theory and Its Applications*. IEEE. 2020, pp. 274–278.

[Hag23] M. Hagiwara. "Multi Deletion/Substitution/Erasure Error-Correcting Codes for Information in Array Design". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 106.3 (2023), pp. 368–374.

[Ham50] R. W. Hamming. "Error Detecting and Error Correcting Codes". In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160.

[HCW21] R. Hulett, S. Chandak, and M. Wootters. "On Coding for an Abstracted Nanopore Channel for DNA Storage". In: *Proceedings of 2021 IEEE International Symposium on Information Theory*. IEEE. 2021, pp. 2465–2470.

[HD23a] J. Haghighat and T. M. Duman. "A Practical Concatenated Coding Scheme for Noisy Shuffling Channels with Coset-Based Indexing". In: *Proceedings of 2023 IEEE Global Communications Conference*. IEEE. 2023, pp. 1842–1847.

[HD23b] B. Hamoum and E. Dupraz. "Channel Model and Decoder with Memory for DNA Data Storage with Nanopore Sequencing". In: *IEEE Access* 11 (2023), pp. 52075–52087.

[HF02] A. S. J. Helberg and H. C. Ferreira. "On Multiple Insertion/Deletion Correcting Codes". In: *IEEE Transactions on Information Theory* 48.1 (2002), pp. 305–308.

[HLA+14] N. ul Hassan, M. Lentmaier, I. Andriyanova, and G. P. Fettweis. "Improving Code Diversity on Block-Fading Channels by Spatial Coupling". In: *Proceedings of 2014 IEEE International Symposium on Information Theory*. IEEE. 2014, pp. 2311–2315.

[HMG19] R. Heckel, G. Mikutis, and R. N. Grass. "A Characterization of the DNA Data Storage Channel". In: *Scientific Reports* 9.1 (2019), p. 9663.

[HOP96] J. Hagenauer, E. Offer, and L. Papke. "Iterative Decoding of Binary Block and Convolutional Codes". In: *IEEE Transactions on Information Theory* 42.2 (1996), pp. 429–445.

[HS21] B. Haeupler and A. Shahrasbi. "Synchronization Strings and Codes for Insertions and Deletions—A Survey". In: *IEEE Transactions on Information Theory* 67.6 (2021), pp. 3190–3206.

[HSA23] T. Heinis, R. Sokolovskii, and J. J. Alnasir. “Survey of Information Encoding Techniques for DNA”. In: *ACM Computing Surveys* 56.4 (2023), pp. 1–30.

[HSR+17] R. Heckel, I. Shomorony, K. Ramchandran, and N. C. David. “Fundamental Limits of DNA Storage Systems”. In: *Proceedings of 2017 IEEE International Symposium on Information Theory*. IEEE. 2017, pp. 3130–3134.

[HSS+12] E. Hof, I. Sason, S. Shamai, and C. Tian. “Capacity-achieving polar codes for arbitrarily permuted parallel channels”. In: *IEEE Transactions on Information Theory* 59.3 (2012), pp. 1505–1516.

[IK12] M. Inoue and H. Kaneko. “Adaptive Synchronization Marker for Insertion/Deletion/Substitution Error Correction”. In: *Proceedings of 2012 IEEE International Symposium on Information Theory*. IEEE. 2012, pp. 508–512.

[JA11] X. Jiao and M. A. Armand. “Interleaved LDPC Codes, Reduced-Complexity Inner Decoder and an Iterative Decoder for the Davey-MacKay Construction”. In: *Proceedings of 2011 IEEE International Symposium on Information Theory*. IEEE. 2011, pp. 742–746.

[Kac00] E. Kac. “Genesis: A Transgenic Artwork”. In: *Art, Technology, Consciousness* (2000), pp. 17–19.

[KFL01] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. “Factor Graphs and the Sum-Product Algorithm”. In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519.

[KK13] A. A. Kulkarni and N. Kiyavash. “Nonasymptotic Upper Bounds for Deletion Correcting Codes”. In: *IEEE Transactions on Information Theory* 59.8 (2013), pp. 5115–5130.

[KK20] H. Koremura and H. Kaneko. “Insertion/Deletion/Substitution Error Correction by a Modified Successive Cancellation Decoding of Polar Code”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 103.4 (2020), pp. 695–703.

[KMM03] A. Kavcic, X. Ma, and M. Mitzenmacher. “Binary Intersymbol Interference Channels: Gallager Codes, Density Evolution, and Code Performance Bounds”. In: *IEEE Transactions on Information Theory* 49.7 (2003), pp. 1636–1652.

[KMM+21] A. Krishnamurthy, A. Mazumdar, A. McGregor, and S. Pal. “Trace Reconstruction: Generalized and Parameterized”. In: *IEEE Transactions on Information Theory* 67.6 (2021), pp. 3233–3250.

[Koz71] O. K. Kozlov. “A Strong Converse of Shannon’s Theorem for Memoryless Channels with Synchronization Errors”. In: *Problemy Peredachi Informatsii* 7.1 (1971), pp. 102–105.

[KS93] G. Kaplan and S. Shamai. “Information Rates and Error Exponents of Compound Channels with Application to Antipodal Signaling in a Fading Environment”. In: *AEU. Archiv für Elektronik und Übertragungstechnik* 47.4 (1993), pp. 228–239.

[LCA+19] R. Lopez, Y.-J. Chen, S. Dumas Ang, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Seelig, K. Strauss, and L. Ceze. "DNA Assembly for Nanopore Data Storage Readout". In: *Nature Communications* 10.1 (2019), p. 2933.

[LCR+23] B. Lau, S. Chandak, S. Roy, K. Tatwawadi, M. Wootters, T. Weissman, and H. P. Ji. "Magnetic DNA Random Access Memory with Nanopore Readouts and Exponentially-Scaled Combinatorial Addressing". In: *Scientific Reports* 13.1 (2023), p. 8514.

[Len22] A. P. Lenz. "Channel Codes for Reliable and Efficient Data Storage in Modern Memories". PhD thesis. Technische Universität München, 2022.

[Lev01] V. I. Levenshtein. "Efficient Reconstruction of Sequences". In: *IEEE Transactions on Information Theory* 47.1 (2001), pp. 2–22.

[Lev02] V. I. Levenshtein. "Bounds for Deletion/Insertion Correcting Codes". In: *IEEE International Symposium on Information Theory*. IEEE. 2002, pp. 370–.

[Lev66a] V. I. Levenshtein. "Asymptotically Optimum Binary Codes with Correction for Losses of One or Two Adjacent Bits". In: *Problemy Kibernetiki*. Vol. 19. 1966, pp. 293–298.

[Lev66b] V. I. Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals". In: *Soviet Physics Doklady*. Vol. 10. 8. 1966, pp. 707–710.

[LGH00] D. Lund, E. M. Gabidulin, and B. Honary. "A New Family of Optimal Codes Correcting Term Rank Errors". In: *Proceedings of 2000 IEEE International Symposium on Information Theory*. IEEE. 2000, p. 115.

[LHS22] K. Levick, R. Heckel, and I. Shomorony. "Achieving the Capacity of a DNA Storage Channel with Linear Coding Schemes". In: *Proceedings of 2022 56th Annual Conference on Information Sciences and Systems*. IEEE. 2022, pp. 218–223.

[LSC+10] M. Lentmaier, A. Sridharan, D. J. Costello, and K. Sh. Zigangirov. "Iterative Decoding Threshold Analysis for LDPC Convolutional Codes". In: *IEEE Transactions on Information Theory* 56.10 (2010), pp. 5274–5289.

[LSWZ+19] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi. "An Upper Bound on the Capacity of the DNA Storage Channel". In: *Proceedings of 2019 IEEE Information Theory Workshop*. IEEE. 2019, pp. 1–5.

[LSWZ+20] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakohi. "Achieving the Capacity of the DNA Storage Channel". In: *Proceedings of 2020 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2020, pp. 8846–8850.

[LSWZ+22] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi. "The Noisy Drawing Channel: Reliable Data Storage in DNA Sequences". In: *IEEE Transactions on Information Theory* 69.5 (2022), pp. 2757–2778.

[LWP20] A. Lenz, L. Welter, and S. Puchinger. "Achievable Rates of Concatenated Codes in DNA Storage under Substitution Errors". In: *Proceedings of 2020 International Symposium on Information Theory and Its Applications*. IEEE. 2020, pp. 269–273.

[Mac03] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[MBT10] H. Mercier, V. K. Bhargava, and V. Tarokh. "A Survey of Error-Correcting Codes for Channels with Symbol Synchronization Errors". In: *IEEE Communications Surveys & Tutorials* 12.1 (2010), pp. 87–96.

[MDK18] W. Mao, S. N. Diggavi, and S. Kannan. "Models and Information-Theoretic Bounds for Nanopore Sequencing". In: *IEEE Transactions on Information Theory* 64.4 (2018), pp. 3216–3236.

[MG04] R. R. Müller and W. H. Gerstacker. "On the Capacity Loss Due to Separation of Detection and Decoding". In: *IEEE Transactions on Information Theory* 50.8 (2004), pp. 1769–1778.

[MGK+18] O. Milenkovic, R. Gabrys, H. M. Kiah, and S. M. H. Tabatabaei Yazdi. "Exabytes in a Test Tube". In: *IEEE Spectrum* 55.5 (2018), pp. 40–45.

[Mit09] M. Mitzenmacher. "A Survey of Results for Deletion Channels and Related Synchronization Channels". In: (2009).

[MJM+24] G. Ma, X. Jiao, J. Mu, H. Han, and Y. Yang. "Deep Learning-Based Detection for Marker Codes over Insertion and Deletion Channels". In: *IEEE Transactions on Communications* (2024).

[MK05] O. Milenkovic and N. Kashyap. "DNA Codes that Avoid Secondary Structures". In: *Proceedings of International Symposium on Information Theory*. IEEE. 2005, pp. 288–292.

[MKB08] H. Mercier, M. Khabbazian, and V. K. Bhargava. "On the Number of Subsequences When Deleting Symbols from a String". In: *IEEE Transactions on Information Theory* 54.7 (2008), pp. 3279–3285.

[MKG03] S. Marillonnet, V. Klimyuk, and Y. Gleba. "Encoding Technical Information in GM Organisms". In: *Nature Biotechnology* 21.3 (2003), pp. 224–226.

[MKL+94] N. Merhav, G. Kaplan, A. Lapidoth, and S. Shamai Shitz. "On Information Rates for Mismatched Decoders". In: *IEEE Transactions on Information Theory* 40.6 (1994), pp. 1953–1967.

[MLW+22] I. Maarouf, A. Lenz, L. Welter, A. Wachter-Zeh, E. Rosnes, and A. Graell i Amat. "Concatenated Codes for Multiple Reads of a DNA Sequence". In: *IEEE Transactions on Information Theory* 69.2 (2022), pp. 910–927.

[MRA23] I. Maarouf, E. Rosnes, and A. Graell i Amat. "Achievable Information Rates and Concatenated Codes for the DNA Nanopore Sequencing Channel". In: *Proceedings of 2023 IEEE Information Theory Workshop*. IEEE. 2023, pp. 377–382.

[MT10] M. F. Mansour and A. H. Tewfik. “Convolutional Decoding in the Presence of Synchronization Errors”. In: *IEEE Journal on Selected Areas in Communications* 28.2 (2010), pp. 218–227.

[MT12] M. F. Mansour and A. H. Tewfik. “A Turbo Coding Scheme for Channels with Synchronization Errors”. In: *IEEE Transactions on Communications* 60.8 (2012), pp. 2091–2100.

[MVS24] B. McBain, E. Viterbo, and J. Saunderson. “Information Rates of the Noisy Nanopore Channel”. In: *IEEE Transactions on Information Theory* (2024).

[NCS23] T. T. Nguyen, K. Cai, and P. H. Siegel. “Every Bit Counts: A New Version of Non-Binary VT Codes with More Efficient Encoder”. In: *Proceedings of 2023 IEEE International Conference on Communications*. IEEE. 2023, pp. 5477–5482.

[NCS24] T. T. Nguyen, K. Cai, and P. H. Siegel. “A New Version of q-ary Varshamov-Tenengolts Codes with more Efficient Encoders: The Differential VT Codes and The Differential Shifted VT Codes”. In: *IEEE Transactions on Information Theory* (2024).

[Nei64] M. S. Neiman. “Some Fundamental Issues of Microminiaturization”. In: *Radiotekhnika* 1.1 (1964), pp. 3–12.

[NW70] S. B. Needleman and C. D. Wunsch. “A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins”. In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453.

[OAC+18] L. Organick, S. Dumas Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, et al. “Random Access in Large-Scale DNA Data Storage”. In: *Nature Biotechnology* 36.3 (2018), pp. 242–248.

[OCA+20] L. Organick, Y.-J. Chen, S. Dumas Ang, R. Lopez, X. Liu, K. Strauss, and L. Ceze. “Probing the Physical Limits of Reliable DNA Data Retrieval”. In: *Nature Communications* 11.1 (2020), p. 616.

[PIS+10] M. Papaleo, A. R. Iyengar, P. H. Siegel, J. K. Wolf, and G. E. Corazza. “Windowed Erasure Decoding of LDPC Convolutional Codes”. In: *Proceedings of 2010 IEEE Information Theory Workshop on Information Theory*. IEEE. 2010, pp. 1–5.

[PMB+18] D. Panda, K. A. Molla, M. J. Baig, A. Swain, D. Behera, and M. Dash. “DNA as a Digital Information Storage Device: Hope or Hype?” In: *3 Biotech* 8 (2018), pp. 1–9.

[PT21] H. D. Pfister and I. Tal. “Polar Codes for Channels with Insertions, Deletions, and Substitutions”. In: *Proceedings of 2021 IEEE International Symposium on Information Theory*. IEEE. 2021, pp. 2554–2559.

[PTY+22] C. Pan, S. Kasra Tabatabaei, S. M. H. Tabatabaei Yazdi, A. G. Hernandez, C. M. Schroeder, and O. Milenkovic. "Rewritable Two-Dimensional DNA-Based Data Storage with Machine Learning Reconstruction". In: *Nature Communications* 13.1 (2022), p. 2984.

[Rat05] E. A. Ratzer. "Marker Codes for Channels with Insertions and Deletions". In: *Annales des Télécommunications*. Vol. 60. 1-2. Paris, Societe de la Revue optique. 2005, pp. 29–44.

[RB99] C. T. Clelland V. Risca and C. Bancroft. "Hiding Messages in DNA Microdots". In: *Nature* 399.6736 (1999), pp. 533–534.

[RL09] W. Ryan and S. Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[RMR+17] C. Rashtchian, K. Makarychev, M. Racz, S. Ang, D. Jevdjic, S. Yekhanin, L. Ceze, and K. Strauss. "Clustering Billions of Reads for DNA Data Storage". In: *Advances in Neural Information Processing Systems* 30 (2017).

[Rot06] R. M. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006.

[Rot91] R. M. Roth. "Maximum-Rank Array Codes and Their Application to Crisscross Error Correction". In: *IEEE Transactions on Information Theory* 37.2 (1991), pp. 328–336.

[Rot97] R. M. Roth. "Probabilistic Crisscross Error Correction". In: *IEEE Transactions on Information Theory* 43.5 (1997), pp. 1425–1438.

[RS60] I. S. Reed and G. Solomon. "Polynomial Codes Over Certain Finite Fields". In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304.

[SAB23] R. Sokolovskii, A. Graell i Amat, and F. Brännström. "Finite-Length Scaling of SC-LDPC Codes With a Limited Number of Decoding Iterations". In: *IEEE Transactions on Information Theory* 69.8 (2023), pp. 4869–4888.

[SB20] J. Sima and J. Bruck. "On Optimal k-Deletion Correcting Codes". In: *IEEE Transactions on Information Theory* 67.6 (2020), pp. 3360–3375.

[SCN22] W. Song, K. Cai, and T. T. Nguyen. "List-Decodable Codes for Single-Deletion Single-Substitution with List-Size Two". In: *Proceedings of 2022 IEEE International Symposium on Information Theory*. IEEE. 2022, pp. 1004–1009.

[SDD+18] S. R. Srinivasavaradhan, M. Du, S. Diggavi, and C. Fragouli. "On Maximum Likelihood Reconstruction Over Multiple Deletion Channels". In: *Proceedings of 2018 IEEE International Symposium on Information Theory*. IEEE. 2018, pp. 436–440.

[Sel62] F. Sellers. "Bit Loss and Gain Correction Code". In: *IRE Transactions on Information Theory* 8.1 (1962), pp. 35–38.

[Sew98] E. C. Sewell. “A Branch and Bound Algorithm for the Stability Number of a Sparse Graph”. In: *INFORMS Journal on Computing* 10.4 (1998), pp. 438–447.

[SFH+03] G. C. Smith, C. C. Fiddes, J. P. Hawkins, and J. P. L. Cox. “Some Possible Codes for Encrypting Data in DNA”. In: *Biotechnology Letters* 25 (2003), pp. 1125–1130.

[SFSB+20] J. Scarlett, A. Guillén i Fàbregas, A. Somekh-Baruch, and A. Martinez. “Information-Theoretic Foundations of Mismatched Decoding”. In: *Foundations and Trends® in Communications and Information Theory* 17.2–3 (2020), pp. 149–401.

[SGB20a] J. Sima, R. Gabrys, and J. Bruck. “Optimal Codes for the q-ary Deletion Channel”. In: *Proceedings of 2020 IEEE International Symposium on Information Theory*. IEEE. 2020, pp. 740–745.

[SGB20b] J. Sima, R. Gabrys, and J. Bruck. “Optimal Systematic t-Deletion Correcting Codes”. In: *Proceedings of 2020 IEEE International Symposium on Information Theory*. IEEE. 2020, pp. 769–774.

[SGB20c] J. Sima, R. Gabrys, and J. Bruck. “Syndrome Compression for Optimal Redundancy Codes”. In: *Proceedings of 2020 IEEE International Symposium on Information Theory*. IEEE. 2020, pp. 751–756.

[SGD14] F. Sala, R. Gabrys, and L. Dolecek. “Gilbert-Varshamov-like Lower Bounds for Deletion-Correcting Codes”. In: *Proceedings of 2014 IEEE Information Theory Workshop*. IEEE. 2014, pp. 147–151.

[SGP+21] S. R. Srinivasavaradhan, S. Gopi, H. D. Pfister, and S. Yekhanin. “Trellis BMA: Coded Trace Reconstruction on IDS Channels for DNA Storage”. In: *Proceedings of 2021 IEEE International Symposium on Information Theory*. IEEE. 2021, pp. 2453–2458.

[SH22] I. Shomorony and R. Heckel. “Information-Theoretic Foundations of DNA Data Storage”. In: *Foundations and Trends® in Communications and Information Theory* 19.1 (2022), pp. 1–106.

[Sha48] C. E. Shannon. “A Mathematical Theory of Communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[SHY19] R. Shibata, G. Hosoya, and H. Yashima. “Design of Irregular LDPC Codes Without Markers for Insertion/Deletion Channels”. In: *Proceedings of 2019 IEEE Global Communications Conference*. IEEE. 2019, pp. 1–6.

[SHY20] R. Shibata, G. Hosoya, and H. Yashima. “Concatenated LDPC/Trellis Codes: Surpassing the Symmetric Information Rate of Channels with Synchronization Errors”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 103.11 (2020), pp. 1283–1291.

[Sid76] V. R. Sidorenko. “Class of Correcting Codes for Errors with a Lattice Configuration”. In: *Problemy Peredachi Informatsii* 12.3 (1976), pp. 3–12.

[Sim22] J. Sima. “Correcting Errors in DNA Storage”. PhD thesis. California Institute of Technology, 2022.

[SK20] R. Sakogawa and H. Kaneko. “Symbolwise MAP Estimation for Multiple-Trace Insertion/Deletion/Substitution Channels”. In: *Proceedings of 2020 IEEE International Symposium on Information Theory*. IEEE. 2020, pp. 781–785.

[SKF99] K. Saowapa, H. Kaneko, and E. Fujiwara. “Systematic Deletion/Insertion Error Correcting Codes with Random Error Correction Capability”. In: *Proceedings 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. IEEE. 1999, pp. 284–292.

[SKS+24] O. Sabary, H. M. Kiah, P. H. Siegel, and E. Yaakobi. “Survey for a Decade of Coding for DNA Storage”. In: *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications* (2024).

[Slo02] N. J. A. Sloane. “On Single-Deletion-Correcting Codes”. In: *Codes and Designs* 10 (2002), pp. 273–291.

[SPC+20] W. Song, N. Polyanskii, K. Cai, and X. He. “Systematic Single-Deletion Multiple-Substitution Correcting Codes”. In: *arXiv preprint arXiv:2006.11516* (2020).

[SPC+21] W. Song, N. Polyanskii, K. Cai, and X. He. “On Multiple-Deletion Multiple-Substitution Correcting Codes”. In: *Proceedings of 2021 IEEE International Symposium on Information Theory*. IEEE. 2021, pp. 2655–2660.

[SPC+22] W. Song, N. Polyanskii, K. Cai, and X. He. “Systematic Codes Correcting Multiple-Deletion and Multiple-Substitution Errors”. In: *IEEE Transactions on Information Theory* 68.10 (2022), pp. 6402–6416.

[SPS07] J. B. Soriaga, H. D. Pfister, and P. H. Siegel. “Determining and Approaching Achievable Rates of Binary Intersymbol Interference Channels Using Multistage Decoding”. In: *IEEE Transactions on Information Theory* 53.4 (2007), pp. 1416–1429.

[SRB19] J. Sima, N. Raviv, and J. Bruck. “Two Deletion Correcting Codes from Indicator Vectors”. In: *IEEE Transactions on Information Theory* 66.4 (2019), pp. 2375–2391.

[SSAG+17] D. Smith, T. G. Swart, K. A. S. Abdel-Ghaffar, H. C. Ferreira, and L. Cheng. “Interleaved Constrained Codes with Markers Correcting Bursts of Insertions or Deletions”. In: *IEEE Communications Letters* 21.4 (2017), pp. 702–705.

[SW24b] W. J. P. Spee and J. H. Weber. “Bounds On the Maximum Cardinality of Indel and Substitution Correcting Codes”. In: *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications* (2024).

[SWZG+17] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi. “Codes Correcting a Burst of Deletions or Insertions”. In: *IEEE Transactions on Information Theory* 63.4 (2017), pp. 1971–1985.

[SY19] M. A. Sini and E. Yaakobi. “Reconstruction of Sequences in DNA Storage”. In: *Proceedings of 2019 IEEE International Symposium on Information Theory*. IEEE. 2019, pp. 290–294.

[SYS+24] O. Sabary, A. Yucovich, G. Shapira, and E. Yaakobi. “Reconstruction Algorithms for DNA-Storage Systems”. In: *Scientific Reports* 14.1 (2024), p. 1951.

[SYY20] O. Sabary, E. Yaakobi, and A. Yucovich. “The Error Probability of Maximum-Likelihood Decoding Over Two Deletion/Insertion Channels”. In: *Proceedings of 2020 IEEE International Symposium on Information Theory*. IEEE. 2020, pp. 763–768.

[SZ99] L. J. Schulman and D. Zuckerman. “Asymptotically Good Codes Correcting Insertions, Deletions, and Transpositions”. In: *IEEE Transactions on Information Theory* 45.7 (1999), pp. 2552–2557.

[Ten84] G. M. Tenengolts. “Nonbinary Codes Correcting Single Deletion or Insertion (Corresp.)” In: *IEEE Transactions on Information Theory* 30.5 (1984), pp. 766–769.

[TFV21] K. Tian, A. Fazeli, and A. Vardy. “Polar Coding for Channels with Deletions”. In: *IEEE Transactions on Information Theory* 67.11 (2021), pp. 7081–7095.

[TPF+21] I. Tal, H. D. Pfister, A. Fazeli, and A. Vardy. “Polar Codes for the Deletion Channel: Weak and Strong Polarization”. In: *IEEE Transactions on Information Theory* 68.4 (2021), pp. 2239–2265.

[Ull66] J. Ullman. “Near-Optimal Single-Synchronization-Error-Correcting Code”. In: *IEEE Transactions on Information Theory* 12.4 (1966), pp. 418–424.

[Ull67] J. Ullman. “On the Capabilities of Codes to Correct Synchronization Errors”. In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 95–105.

[Vin68] T. K. Vintsyuk. “Speech Discrimination by Dynamic Programming”. In: *Cybernetics* 4.1 (1968), pp. 52–57.

[VT65] R. R. Varshamov and G. M. Tenengolts. “Codes Which Correct Single Asymmetric Errors (in Russian)”. In: *Automatika i Telemkhanika* 161.3 (1965), pp. 288–292.

[WC53] J. D. Watson and F. H. C. Crick. “The Structure of DNA”. In: 18 (1953), pp. 123–131.

[Wei22] N. Weinberger. “Error Probability Bounds for Coded-Index DNA Storage Systems”. In: *IEEE Transactions on Information Theory* 68.11 (2022), pp. 7005–7022.

[WF74] R. A. Wagner and M. J. Fischer. “The String-to-String Correction Problem”. In: *Journal of the ACM* 21.1 (1974), pp. 168–173.

[WFD10] F. Wang, D. Fertonani, and T. M. Duman. “Marker Code Optimization and Symbol-Level Synchronization for Insertion/Deletion Channels”. In: *Proceedings of 2010 IEEE International Symposium on Information Theory*. IEEE. 2010, pp. 1007–1011.

[WFD11] F. Wang, D. Fertonani, and T. M. Duman. “Symbol-Level Synchronization and LDPC Code Design for Insertion/Deletion Channels”. In: *IEEE Transactions on Communications* 59.5 (2011), pp. 1287–1297.

[WG08] F. M. J. Willems and A. Gorokhov. “Signaling over arbitrarily permuted parallel channels”. In: *IEEE Transactions on Information Theory* 54.3 (2008), pp. 1374–1382.

[Wie64] N. Wiener. “Interview: Machines Smarter Than Men”. In: *US News World Rep* 56 (1964), pp. 84–6.

[WM22] N. Weinberger and N. Merhav. “The DNA Storage Channel: Capacity and Error Probability Bounds”. In: *IEEE Transactions on Information Theory* 68.9 (2022), pp. 5657–5700.

[WWF03] P. C. Wong, K. k. Wong, and H. Foote. “Organic Data Memory Using the DNA Approach”. In: *Communications of the ACM* 46.1 (2003), pp. 95–98.

[WZ16] A. Wachter-Zeh. “List Decoding of Crisscross Errors”. In: *IEEE Transactions on Information Theory* 63.1 (2016), pp. 142–149.

[XLC+23] L. Xiang, Q. Liu, S. Chen, K. Yan, W. Wu, and K. Yang. “A Tutorial on Coding Methods for DNA-Based Molecular Communications and Storage”. In: *IEEE Internet of Things Journal* (2023).

[YGM17] S. M. H. Tabatabaei Yazdi, R. Gabrys, and O. Milenkovic. “Portable and Error-Free DNA-Based Data Storage”. In: *Scientific Reports* 7.1 (2017), p. 5011.

[YKGR+15] S. M. H. Tabatabaei Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic. “DNA-Based Storage: Trends and Methods”. In: *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 1.3 (2015), pp. 230–248.

[YSS+07] N. Yachie, K. Sekiyama, J. Sugahara, Y. Ohashi, and M. Tomita. “Alignment-Based Approach for Durable Data Storage into Living Organisms”. In: *Biotechnology Progress* 23.2 (2007), pp. 501–505.

[YYM+15] S. M. H. Tabatabaei Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic. “A Rewritable, Random-Access DNA-Based Storage System”. In: *Scientific Reports* 5.1 (2015), pp. 1–10.

[Zig69] K. S. Zigangirov. “Sequential Decoding for a Binary Channel with Drop-Outs and Insertions”. In: *Problemy Peredachi Informatsii* 5.2 (1969), pp. 23–30.

[ZZ24] W. Zhong and X. Zhang. “Trace Reconstruction of Matrices and Hypermatrices”. In: *arXiv preprint arXiv:2407.11795* (2024).

[ZZS+16] V. Zhirnov, R. M. Zadegan, G. S. Sandhu, G. M. Church, and W. L. Hughes. “Nucleic Acid Memory”. In: *Nature Materials* 15.4 (2016), pp. 366–370.