Feature

Jin Sima | Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, Urbana, IL 61801 USA | E-mail: jsima@illinois.edu
Netanel Raviv, Senior Member, IEEE | Department of Computer Science and Engineering, McKelvey School of Engineering, Washington University in Saint Louis, St. Louis, MO 63130 USA | E-mail: netanel.raviv@wustl.edu
Moshe Schwartz, Senior Member, IEEE | Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON, L8S 4L8, Canada | E-mail: schwartz.moshe@mcmaster.ca
Jehoshua Bruck, Life Fellow, IEEE | Electrical Engineering Department, California Institute of Technology, Pasadena, CA 91125 USA | E-mail: bruck@caltech.edu

Error Correction for DNA Storage

Abstract—DNA-based storage is an emerging technology that provides high information density and longevity. Noise and errors are present in almost every stage of the process: writing, storing, and reading. Thus, efficient error correction is crucial to guarantee high reliability and low cost of storing data in DNA. Due to technological constraints and biological limitations, error correction in DNA-based storage poses several coding-theoretic challenges, some of which are new. In this paper, we briefly introduce some of these challenges, including deletion/insertion correcting codes, codes over sliced channels, and duplication-correcting codes. We describe some of the major concepts and solutions in the respective topics, and provide bibliographic notes that briefly review the related literature.

Introduction

Information theory is considered by many as the mathematical theory of *communication*. Normally, the word "communication" describes a scenario involving two physically distant parties that exchange information, but may equally involve two *temporally* distant parties that do so. The latter gives rise to communication across time, rather than across space, and is commonly referred to as *information storage* [8], i.e., the process of encoding information into a physical device in order to retrieve it at a later point in time, efficiently and accurately.

In his groundbreaking 1948 paper, Claude Shannon (1916–2001) showed that *all* types of information (images, text, videos, etc.) can be communicated using bits, i.e., zeros and ones, and an identical statement holds in the case of storage.

2692-4080 © 2023 IEEE
Digital Object Identifier 10.1109/MBITS.2023.3318516
Date of publication 25 September 2023; date of current version 19 August 2024.

In order to store a piece of information, one has to encode it using bits, and place those bits on a reliable physical device, preferably a *nonvolatile* one, i.e., that does not require electric current to retain that information.

The earliest example of high-density nonvolatile storage device (beyond punch-cards and written media which existed for millennia) is probably that of *magnetic storage*. In this 1950s technology, bits were organized on a magnetizable tape using different magnetization patterns. Over the following decades, increased demand for higher storage volumes pushed this technology forward to become the *hard-disk drives*, which in recent years cleared the way to *solid-state drives*.

Albeit over ten orders of magnitude increase in volume since they were introduced, digital storage devices struggle to keep up with increasing storage demands. The immense volume of data generated today, especially since the emergence of information sharing platforms such as YouTube and social networks, is projected to pass the rate in which digital storage devices improve. Especially prominent is the growing requirement for "cold" storage, i.e., one which is seldom accessed, such as old family photos or historical records. One of the most promising and most radical new technologies to resolve the cold storage problem, is *DNA storage*, i.e., storing information in DNA molecules.

In a way, DNA molecules are the storage device of nature. These long molecules, which contain sequences of either one of four basic molecules called nucleotides A, C, G, T, are used by all living organisms to communicate across time. The DNA molecules contain "recipes" for producing proteins, which are the building blocks of living organisms. By communicating DNA across generations, these recipes are literally transmitted from parents to offsprings, and enable life to continue.

In the past several decades, scientists have had remarkable success in creating artificial DNA molecules in the lab, and in keeping those molecules stable in a vial (i.e., not inside any

living cell). The A, C, G, T nucleotides in these artificial molecules can be chosen freely, and can therefore store bits just like any other storage device. For instance, one can decide that

$$\mathsf{A} = 00, \mathsf{C} = 01, \mathsf{G} = 10, \mathsf{T} = 11$$

and then store the sequence 00101001 by creating the DNA molecule AGGC in the lab. DNA storage has fantastic and far-reaching advantages over existing technology:

- *DNA storage is ultradense:* Current data centers are the size of buildings; storing similar amounts of data in DNA would require the size of a refrigerator.
- *DNA storage is stable:* DNA molecules can last tens of thousands of years without any energy investment; some off-the-shelf hard drives will not be usable in as little as 20 years.
- *DNA storage is future-proof:* As long as there are humans, DNA reading technology will be of interest, and DNA reading devices will exist. This can hardly be said about, say, floppy disks, whose reading nowadays requires a trip to a museum. In other words, while humans already almost forgot how to read some fairly recent storage devices, humans will never forget how to read DNA.

A typical process to store information in DNA is as follows: First, the data, represented by 0s and 1s, are encoded into sequences of nucleotides. Then, the DNA molecules containing these sequences of nucleotides are synthesized and stored either in vials or in living organisms, which is the data writing process. To read the data, the polymerase chain reaction (PCR) technology is used to access the part of the data to be retrieved. Then, the DNA molecules obtained after the PCR process are read using DNA sequencing techniques, thereby recovering the sequences of nucleotides. Finally, the sequences of nucleotides are decoded back to the data. A more detailed description of a DNA storage workflow is given in Section III.

Whether old or new, all storage devices are prone to *errors*. Due to imperfect hardware, physical damage, or deterioration of materials, some bits in any storage device might be read in error. Without proper preparation, losing even a single bit might render the respective piece of information unreadable, and therefore lost. Coding-theorists and engineers have been combating this phenomenon ever since storage devices were invented, and various *error-correction* mechanisms, known as *codes*, were developed.

All these mechanisms require adding *redundancy* to the data, i.e., to store more bits than the actual size of the data. These redundant bits are then used in the reading process in cases, where some bits are read in error. The simplest form of redundancy is *replication:* instead of storing every bit of the data once, store it three times. For example,

Data:	01001
Store:	000111000000111

Then, while reading a possibly error-filled sequence from the device, the most frequent bit among every consecutive triple is most likely the correct one:

Store:	000111000000111
Errors:	100110000001111
Correction:	0 1 0 0 1

But how much redundancy is enough? The example above shows a three-times increase in the amount of storage, a high price to pay. The minimal amount of redundancy required to correct errors is an ongoing and difficult research area in coding theory, and depends on the type of storage medium. Coding theorists have worked relentlessly over the past 80 years in order to come up with algorithms that guarantee error-free information storage for the existing storage technology. However, as we shall see throughout this article, DNA storage devices have a very unique structure and constraints, which give rise to new and interesting types of errors which have never been studied.

The error shown above is called *substitution:* A "1" bit is replaced by a "0" bit, or vice versa. This is a common and well-understood error in traditional storage media, which seldom appears in DNA storage devices. However, most errors in DNA storage devices are new, i.e., have never appeared in traditional devices. These new types of errors depend on the type of DNA storage device at hand.

DNA storage devices are partitioned to two families: The most common family is called *in vitro*, which includes devices that contain a vial with short, unordered sequences of DNA that float in a solution inside that vial. The other less common family is called *in vivo*, where artificially synthesized DNA molecules are planted inside a primitive life form, such as bacteria, for better data longevity and stability. Better longevity is guaranteed by the self-sustaining property of primitive life-forms; with minimal energy investment a bacteria colony could last millions of years. Better stability is guaranteed by reproduction across generations; redundancy in the data will be introduced naturally, and will be stabilized via natural selection.

In-vitro DNA storage devices pose several interesting coding-theoretic challenges. First, they are prone to known but understudied errors called *deletions*. In a deletion, a bit completely disappears from a sequence without leaving a trace, and the read sequence is shorter than the one which was initially stored. Deletions occur in DNA storage mainly due to imperfections in the synthesis reaction, which

commonly skip one or more of the nucleotides that need to be written due to weak chemical bonds. A closely related type of errors, where redundant bits sneak into the sequence unexpectedly, are called *insertions*.

Even though mechanisms for deletion correction, known as deletion codes, were studied to some extent since the 1960s, much was left unknown. Driven by the advent in DNA storage, the interest in deletion errors increased recently, and optimal solutions were found only in the past few years. Deletion errors will be described in Section II.

An even more substantial challenge in in-vitro DNA storage systems is the fact that only *short and unordered* DNA sequences can be stored together in a vial. Current limitation in DNA-synthesis technology can only generate sequences that are a few thousand nucleotides long, and placing those sequences together in a vial makes it impossible to know which one comes after or before any other. This is in sharp contrast to traditional storage media, where data are partitioned to *pages*, which always appear in memory in the same order they were written.

A simple solution to the ordering problem comes in the form of *indexing:* begin each short DNA sequence with several nucleotides that determine its correct position relative to other sequences in the same vial. Surprisingly, this is *not* the best solution in terms of the amount of redundancy. Due to errors, the indexing nucleotides might get scrambled and interfere with the correct order. An optimal solution for the order problem was also found very recently, and it is described in Section III.

For in-vivo storage, however, the picture is remarkably different. While placing the synthetic DNA inside bacteria improves its longevity and stability, it exposes the stored data to the natural biochemical and evolutionary processes inside the bacteria. As the reader might already know, cells reproduce by a process called *mitosis*, where one cell splits into two. During mitosis, DNA molecules replicate themselves in each one of the offspring cells, a process that is not perfect, and some errors might occur. In nature, these errors are the basis of Darwin's natural selection theory: arbitrary errors cause arbitrary mutations, and only the mutations which improve the organism's ability to survive persist among generations. In the context of information storage, however, these errors must be understood, and corrected; an *evolution-correcting code* must be developed. A common error in this setting is *duplication*, where a piece of DNA material is replicated, and attaches itself at a different location. Duplication errors in in-vivo DNA storage are described in Section IV.

Deletion Codes

Though deletion errors were studied from the 1960s, motivated by synchronization errors in traditional media, the interest in correcting deletion errors increased recently due to their prevalence in DNA storage. As mentioned in Section I, deletions, insertions, and substitutions are the notable three types of errors that occur in the reading, writing, and storing processes in DNA storage. Hence, codes correcting these three types of errors are necessary for reliably storing information in DNA. Beyond the applications in DNA storage and communication, the study of deletions, insertions, and substitutions, is also connected to edit distance and sequence alignment, etc., which have applications in natural language processing and other applications involving DNA sequence analysis.

What are Deletion, Insertion, and Substitution Errors?

We now describe the three types of errors in greater detail. A deletion removes a symbol from a sequence; in the context of DNA storage, it removes a nucleotide from the sequence, e.g., turning TGGA into TGA. In the context of natural language, it removes a letter from a word or a sentence, e.g., turning the word "cat" into "at." An insertion adds an extra symbol to the sequence, e.g., turning ACTG into ACCTG or turning "eat" into "heat." A substitution replaces a symbol in the sequence, e.g., changing TGGA to TGGG or "for" to "far."

Among those three types of errors, substitution errors are better understood compared to deletions and insertions, as there are many classic code constructions for correcting substitution errors such as Hamming codes, polar codes, LDPC codes, Reed–Muller codes, Reed–Solomon codes, and BCH codes, etc. Many of these codes were proved to be optimal in terms of redundancy in some settings. However, less is known about deletion and insertion errors, which are commonly referred to as *synchronization* errors. Nevertheless, the information-theorist Vladimir Levenshtein (1935–2017) proved an interesting fact about deletion and insertion errors already in the 1960s: if a code corrects deletion errors, it can also correct an equal number of combination of deletions and insertions (However, an efficient encoding/decoding algorithm for correcting deletions does not necessarily imply an efficient algorithm for correcting deletion and insertion errors). Moreover, a substitution error can be regarded as a deletion error followed by an insertion. Therefore, it is reasonable to focus on deletion errors, as correcting deletion errors implies correcting a combination of the three types of errors.

There are two scenarios for correcting deletion errors:

1) The *probabilistic* scenario, where a fraction of deletions occur randomly. The goal is to find the optimal information rate (i.e., the data size relative to the code length), known as the *channel capacity*, such that the information could be recovered with high probability.

2) The *adversarial* scenario, where at most a certain number of deletions are caused by an adversary that wishes the reading process to fail. The goal is to find the minimal amount of added redundancy that guarantees successful reading in all cases.

DNA storage devices are commonly considered as an adversarial scenario, since the number of deletions is usually quite small, and the respective amount of redundancy can be optimized directly. Therefore, in this section, we focus on *adversarial deletion correcting codes*. Moreover, for simplicity we consider bits rather than nucleotides, however, similar statements can be made for nucleotides as well.

In contrast to multiple results about correcting substitution errors, there are not many efficient and well structured codes correcting deletion errors, and even the deletion channel capacity is still unknown except for cases where the deletion probability is small. One of the reasons for deletions and insertions being more difficult is that channels with substitution channels are *memoryless*, i.e., different output bits are independent given the input, while deletion/insertion channels are not; one deletion affects all subsequent bits by shifting them one position to the left. Moreover, there is a symmetry in substitution errors that is not present in deletion or insertion errors.

To see this symmetry, we introduce the notion of an *error ball*, which is common in the analysis of error correcting codes. An error ball is the set of all possible sequences one can get after at most some number of errors occur in a given input sequence. If the type of error is substitution or deletion, we call the corresponding error ball a substitution ball or a deletion ball, respectively.

For example, the substitution ball and the deletion ball of the input sequence 1001 with at most 1 error are given by $\{1001, 0001, 1101, 1011, 1000\}$ and $\{1001, 001, 101, 100\}$, respectively. The symmetry in substitution errors reflects the fact that the size of the substitution ball is independent of the input sequence; any other sequence of length 4 will have a substitution ball with 5 sequences. Moreover, the erroneous sequence is uniformly distributed over the substitution ball if the substitution indices are uniformly and randomly selected. These two properties do not hold for deletion balls, since two deletion balls (for different input sequences) can have different sizes. In addition, the probabilities of getting different erroneous sequences in the deletion ball are different when the deletion indices are uniformly and randomly selected. The following example illustrates this crucial difference in symmetry.

Example 1. Consider the sequences 0000 and 1010. The substitution balls of 0000 and 1010, with at most a single substitution, are $\{0000, 1000, 0100, 0010, 0001\}$ and $\{1010, 0010, 1110, 1000, 1011\}$, respectively. Each set has five elements and each element appears once under all possible error patterns. The deletion balls of 0000 and 1011 with at most a single deletion are given by $\{0000, 000\}$ and $\{1011, 011, 111, 101\}$, respectively. The numbers of elements in the two sets are different. In addition, after a single deletion in 1011, the erroneous sequence becomes 011 or 111 if the first bit or the second bit is deleted, respectively. The erroneous sequence becomes 101 if either the third or the fourth bit is deleted. Hence, it is more probable to obtain 101 than 011 or 111 after a single deletion uniformly occurs in 1011.

How to Efficiently Correct Deletions?

One of the natural approaches to correct deletion errors is to use a *repetition code*, which was presented in the introduction for substitution errors but also works well for deletion errors. This is undesirable due to high redundancy. Another potential approach is to borrow results from substitution correcting codes. The difficulty with this approach is that even two binary sequences with large *Hamming distance* (i.e., have multiple different bits), which are resilient to substitution errors, can be ambiguous even under a single deletion. As an example, one can consider two sequences 1010101 and 0101010 that have Hamming distance 7 (i.e., all bits are different), and therefore one can identify the correct sequence between them under any three substitutions. However, these two sequences become indistinguishable in the case of even a single deletion, since deleting the first bit in one produces the same string 010101 as deleting the last bit in the other.

How to correct a single deletion with low redundancy? One classic construction is the Varshamov–Tenengolt (VT) codes, defined by

$$\mathcal{C}_0 = \{(c_1, \ldots, c_n) : \textstyle\sum_{i=1}^{n} i c_i \equiv 0 \bmod n+1\} \tag{1}$$

which corrects a single deletion. In words, VT codes show that one can protect any sequence from a single deletion by summing up the indices of all the 1 entries in the sequence and taking a modulo of $n+1$. The following example demonstrates how this modulo summation scheme works.

Example 2. Suppose the erroneous sequence 00110 is obtained from some unknown sequence $c_1c_2c_3c_4c_5c_6$ after a single deletion. To see what $c_1c_2c_3c_4c_5c_6$ is, enumerate all possible sequences of length 6 that might become 00110 after a single deletion, given by 000110, 001010, 001100, 100110, 010110, 001110, and 001101. The respective modulo summations, i.e., the expressions $\sum_{i=1}^{6} i c_i \bmod (n+1)$, are given by 2, 1, 0, 3, 4, and 5, respectively, all of which are different. Therefore, the only sequence with modulo summation 0 is 001100, and hence it must be the correct answer.

The VT code has redundancy at most $\log_2(n+1)$ bits, which is asymptotically optimal for a single deletion.

Can we generalize the VT code to correct more than a single deletion with asymptotically optimal redundancy? This is a key question toward a good understanding of how to correct deletion errors, and turns out to be highly nontrivial. A counting argument by Vladimir Levenshtein showed that the optimal redundancy of codes of length n correcting t deletions lies between $t\log_2 n + o(\log_2 n)$ and $2t\log_2 n + o(\log_2 n)$ for constant t, where the notation $o(\log_2 n)$ means that $(o(\log_2 n))/(\log_2 n)$ approaches 0 as n goes to infinity. For larger t, the optimal redundancy is linear in $t\log_2(\frac{n}{t})$. These arguments only prove that a code exists, without finding it explicitly. Finding the respective codes explicitly with comparable redundancy has been puzzling for decades even for the case $t = 2$.

Inspired by the modulo summation in VT codes, researchers wondered if *higher power* summation might be useful for correcting multiple deletions with optimal redundancy. Specifically, given the modulo sums $\sum_{i=1}^{n} i^p c_i$ for p from 0 up to some positive integer, is it possible to recover $(c_1, \ldots, c_n)$ from multiple deletions? This is a challenging question even for two deletions, and unfortunately, counterexamples were found showing that knowing the sums $\sum_{i=1}^{n} i^p c_i$ for ps from 0 up to $p = 4$ does not guarantee successful correction.

To generalize the idea of using weighted modulo sum for correcting multiple deletions, one can use weights that are exponential in indices in the weighted modulo sum. However, due to the exponential weights, such generalization requires redundancy that is linear in the code length to correct even two deletions, in contrast to the optimal redundancy which is a logarithm in the code length for correcting a constant number of deletions.

Another brilliant idea for correcting multiple deletions is to use a concatenated code, that has a two-level structure of an inner code and an outer code. Specifically, the codewords are separated into blocks. The inner code protects each block from deletions and is constructed by using exhaustive search (i.e., finding the best code by traversing all codes using a computer). The outer code treats each block as a symbol and uses a substitution correcting code to correct blocks in case the inner code fails in some blocks. Note that the brute force search to construct the inner code is tractable when the block size is small (specifically, a logarithm of the code length). The concatenated code approach reduces the problem of correcting deletions in a long sequence to that of correcting deletions in short sequences, by using the well-constructed substitution correcting codes. Using concatenated codes, it is possible to correct a number of deletions which is linear in the code length, with redundancy that is also linear in the code length. This is asymptotically optimal based on the bounds that were mentioned above.

What about correcting a small number, say a constant, of deletions, a regime that is of interest in DNA storage since the number of deletions is small and the code length is moderately sized? As discussed previously, the optimal redundancy should be asymptotically between $t\log_2 n$ and $2t\log_2 n$ where t is the number of deletions. In the following, we discuss codes with close to optimal redundancy, using a fundamentally different idea from the concatenated code construction. Specifically, we discuss a generalization of the VT codes using an algebraic approach. Recall that the higher order weighted sum $\sum_{i=1}^{n} i^p c_i$, a natural generalization of the VT codes, is not guaranteed to provide a code correcting even two deletions. However, a similar higher order weighted sum is capable of correcting deletions for constrained sequences.

We illustrate the idea for the case of two deletions. In this case, an interesting observation is that if the codewords are sequences with at least a single 0 between any two 1s, then any codeword $(c_1, \ldots, c_n)$ can be protected from two deletions by providing the sums $\sum_{i=1}^{n}(\sum_{j=1}^{i} j^p)c_i \bmod 2n^{p+1}$ for $p = 0, 1, 2$.

Example 3. Consider the sequence 101001 of length 6, where any two 1s are separated by at least one 0. Its weighted modulo sums are

$$\sum_{i=1}^{6}(\sum_{j=1}^{i} j^0)c_i \bmod 2\cdot 6^{0+1} = 3$$
$$\sum_{i=1}^{6}(\sum_{j=1}^{i} j^1)c_i \bmod 2\cdot 6^{1+1} = 28$$
$$\sum_{i=1}^{6}(\sum_{j=1}^{i} j^2)c_i \bmod 2\cdot 6^{2+1} = 106.$$

Suppose two deletions occur in 101001, resulting in 1001. We list all possible sequences that satisfy: 1) can result in 1001 after two deletions; and 2) there is at least one 0 bit between any two 1 bits. These sequences are given by 001001,010001,010010,100001,100010,100100, 10100, 1,010101, and 100101. It can be verified that only 101001 has the weighted modulo sums 3 28 106 as above.

But how do we guarantee at least one 0 between any two 1s? To this end, let us define an *indicator vector* $\mathbb{1}_{10}(c_1, \ldots, c_n)$ of length n for a sequence $(c_1, \ldots, c_n)$ as follows. The ith bit of $\mathbb{1}_{10}(c_1, \ldots, c_n)$ is

$$\mathbb{1}_{10}(c_1, \ldots, c_n)_i = \begin{cases} 1 & \text{if } c_i = 1, \text{ and } c_{i+1} = 0 \\ 0 & \text{otherwise} \end{cases}$$

for $i \in \{1, \ldots, n\}$, where it is assumed that $c_{n+1} = 1$. Note that by definition, for any binary sequence $(c_1, \ldots, c_n)$, there is at least one 0 between any two 1s in $\mathbb{1}_{10}(c_1, \ldots, c_n)$: it is impossible to have to consecutive 1s, since it would imply both that $c_i = 1$ and $c_{i+1} = 0$, and that $c_{i+1} = 1$ and $c_{i+2} = 0$, and clearly c_{i+1} cannot simultaneously be a 0 and a 1.

Therefore, for any sequence $(c_1, \ldots, c_n)$, the indicator vector $\mathbb{1}_{10}(c_1, \ldots, c_n)$ can be protected from two deletions given the weighted modulo sums $\sum_{i=1}^{n}(\sum_{j=1}^{i} j^p)\mathbb{1}_{10}(c_1, \ldots, c_n)_i \mod 2n^{p+1}$ for $p = 0, 1, 2$. Then, it suffices to protect the modulo sums $\sum_{i=1}^{n}(\sum_{j=1}^{i} j^p)\mathbb{1}_{10}(c_1, \ldots, c_n)_i \bmod 2n^{p+1}$, $p = 0, 1, 2$ by using a very short repetition code, in order to recover the vector $\mathbb{1}_{10}(c_1, \ldots, c_n)$. After recovering the vector $\mathbb{1}_{10}(c_1, \ldots, c_n)$, one can protect a similarly defined indicator vector $\mathbb{1}_{01}(c_1, \ldots, c_n)$. Finally, the sequence $(c_1, \ldots, c_n)$ can be recovered from indicator vectors $\mathbb{1}_{10}(c_1, \ldots, c_n)$ and $\mathbb{1}_{01}(c_1, \ldots, c_n)$.

This approach extends to any constant number t of deletions by generalizing the observation: For any t, if the codewords are sequences that have at least $t-1$ 0s between any two 1s, then the codewords can be protected from t deletions by using the weighted modulo sum $\sum_{i=1}^{n}(\sum_{j=1}^{i} j^p)c_i \bmod 3tn^{p+1}$ for $p = 0, 1, \ldots, 6t$. Similarly, one can define indicator vectors such that the indicator vector of a sequence has at least $t-1$ 0s between any two 1s. The resulting redundancy is then $4t\log_2 n + o(\log_2 n)$, which is asymptotically at most four times the optimal.

Despite the progress on codes correcting t deletions, several problems remain open. How to construct minimal-redundancy deletion codes, which can also be decoded efficiently? How to approach or improve the existential redundancy bound? How to efficiently correct a combination of deletions and substitutions? The third problem is crucial in DNA storage applications as errors are normally a combination of deletions, insertions, and substitutions. Though the problem was investigated and codes combining the deletion codes above and the substitution codes were proposed, are there more redundancy efficient methods?

Sliced Channel

One feature that fundamentally distinguishes DNA storage from traditional storage is that in traditional systems, the codeword is a single long sequence, whereas in DNA storage, the codeword is a set of unordered short sequences.

To see this, we briefly describe the workflow of DNA storage systems, as shown in Figure 1. As mentioned earlier, in DNA storage information is encoded into sequences of four letters, $\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$, which are synthesized into the respective DNA molecules. The synthesized DNA molecules are then placed in a solution inside a vial. In the reading phase, the sequences are amplified by a process called polymerase chain reaction (PCR), which generates many more copies of the nucleotide sequences in the vial. The copies are then sampled and read through a sequencing process, producing many potentially erroneous copies of the sequences that were originally synthesized. By using clustering and reconstruction algorithms, the copies generated from the same sequence are clustered, and the corresponding sequence is reconstructed. Finally,

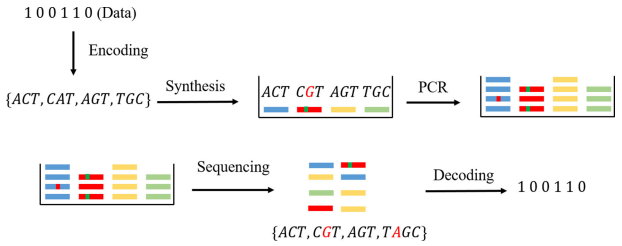

Figure 1

Illustration of the processes in a typical in-vitro DNA storage system. The data (1,0,0,1,1,0) are encoded into a set of short sequences $\{ACT, CAT, AGT, TGC\}$ of nucleotides. Errors occur during the synthesis of the short sequences, turning the nucleotide A in CAT to G. The synthesized sequences, including the erroneous sequence "CGT," are amplified in the PCR process, generating multiple noisy copies of the synthesized sequences. During the sequencing process, these noisy copies are read and clustered such that each cluster consists of noisy copies of a synthesized sequence. Then, an estimate $\{ACT, CGT, AGT, TAGC\}$ of the synthesized sequences is obtained from the clusters of noisy copies, where $TAGC$ is an erroneous estimate of TGC. Finally, the estimated set of synthesized sequences is decoded into data.

the reconstructed sequences are decoded to retrieve the data. Due to technological constraints in the above processes, only short DNA molecules (≈ 100 nucleotides) can be synthesized and sequenced, meaning that information can only be encoded into a collection of short sequences. Moreover, the DNA molecules stored in the same vial are unordered; they all float in the same solution without any knowledge regarding which comes prior.

Note that errors occur in the collection of unordered short nucleotide sequences. This gives rise to the question of how to correct errors when the codeword is "sliced" into multiple unordered pieces. This brings new aspects to classic error correction setups, where the information is encoded into a codeword that is a single sequence, and retrieved from a noisy copy of that sequence. In the context of coding for DNA storage, the codeword is sliced into multiple unordered pieces, normally of equal lengths, which presented both noisy and unordered to the decoder.

In the sliced codeword setting, we may either think of a codeword as a single sequence that it then sliced into multiple unordered pieces, or a priori consider the codeword as the set of those pieces; both approaches are equivalent. In this article, we choose the latter, i.e., we assume that the codeword is a set of M short sequences, each of length L. In existing DNA storage systems, L is of the order of magnitude 100 and M is of the order of magnitude 10^4 to 10^9, based on the size of the data.

In DNA storage, the types of errors include: 1) deletions, insertions, and substitutions, which occur in either of the

sequences, and were discussed in Section II. 2) sequence loss due to the fact that some DNA molecules may not be sampled during the sequencing process. As a result, the sequence contained in the DNA molecule is missing. The following is an example of coding over a set of short binary sequences.

Example 4. Assume that the given data are encoded to $M = 4$ sequences of length $L = 6$, say $\{110001, 100100, 101010, 111111\}$, which are placed together in a vial. Then, noisy copies of the codeword can be $\{100000, 1100001, 10101\}$, after deletion, insertion, substitution errors, and sequence loss that occur in any sequence in the set. Note that the identity of a noisy copy is not known. For example, the noisy copies 100000, 1100001, 10101 can be obtained from 110001, 100100, 101010, respectively, or from 100100, 110001, and 111111, respectively.

The setting of coding over a set of sequences can be considered as a generalization of the classic setting of coding over a single sequence, where the set contains only a single sequence and there is no sequence loss. Also, a similar ordering issue often arises in network packet transmission, where due to varied network delays and changes in routing, packets might arrive not in the same order they were sent.

To understand the problem of how to handle unordered sets of sequences, we focus on the basic setting where the codeword is a set of M different binary sequences of length L, and focus on substitution errors; more complex settings follow similar ideas. For example, one can transform a code that uses $\{0, 1\}$ to one which uses $\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$ using the mapping mentioned in the introduction. Further, to correct deletion and insertion errors, one can combine the codes for this basic setting and the deletion codes discussed in Section II. To combat sequence loss, it is possible to add an "outer" code, such as the Reed–Solomon code. Intuitively, correcting errors in the sliced codeword setting (over a set of sequences) is more difficult than correcting errors in a single sequence, since in the former setting, the information about the index of each sliced piece is lost. One natural way to correct errors in the set of sequences is to use error correction codes to protect each sequence independently. This is efficient when each sequence roughly has the same amount of errors. In the case when some sequences have no errors, or some sequences have much more errors than average, the method may be inefficient since one has to protect every sequence from the largest number of errors possible.

To deal with the loss of the index information of each unordered sequence, another natural (and possibly the most common) approach is to use extra redundancy to index each sequence. That is, in each sequence, dedicate the first $\log_2 M$ (out of L) bits to record the index among the total M sequences. This gives an order to the sequences based on their indices, and reduces the problem of coding over an unordered set of sequences to that of coding over a single sequence.

It can be shown that the simple index-based scheme asymptotically approaches the best information rate, i.e., the channel capacity, for coding over an unordered set of sequences. More specifically, index-based schemes achieve the asymptotically optimal information rate in probabilistic settings, where a fraction of sequence loss and substitution errors randomly occur. Partly for this reason, indexing schemes are used in most of the recent DNA storage experiments, where extra bases are dedicated to index each sequence, and Reed–Solomon codes are used to correct errors in the bases. In addition, many code constructions were proposed based on the indexing schemes.

One of the problems which need to be addressed for indexing schemes is to protect against errors in the index. One way is to encode the indices such that they are far from each other in Hamming distance (i.e., have many distinct bits). In this way, the indices are more robust to substitution errors. However, it requires more redundancy in the indices. To resolve this issue, another approach is to use data to protect errors in indices. When the Hamming distance between two indices is small, meaning that they are ambiguous under substitution errors, it is required the data in the corresponding two sequences have a large enough Hamming distance. With this constraint, the decoder can distinguish two sequences based on their data, if it fails to decode their indices.

While index-based schemes achieve asymptotically the optimal information rate in probabilistic settings, how do they perform in deterministic settings? In deterministic settings, the number of errors is bounded and zero-error decoding is required. When the number of errors is not large, it is reasonable to look at the redundancy, rather than the information rate of a scheme, which approaches one with a small number of errors. Note that different from information rate, which measures the ratio between the amount of information and the number of symbols used to store the information, redundancy measures the difference between the two. But how should we define "redundancy" in the unordered sequence setting? To study this question, we define the redundancy of a code as $\log_2 \binom{2^L}{M} - \log_2 |\mathcal{C}|$, where $|\mathcal{C}|$ is the size of the code (i.e., number of codewords) and $\log_2 |\mathcal{C}|$ is the number of information bits the code can represent. This definition measures how many extra bits are needed for error correction, and would be zero in the case of no errors. Under this definition, the extra redundancy needed for indexing in an index-based scheme is at least $\log_2 \binom{2^L}{M} - M(L - \log_2 M)$ (which is linear in M) even when there are no errors.

Can one use less redundancy than that? Using counting arguments, one can show that the optimal redundancy for correcting a total number of t substitution errors across all M unordered sequences of length L is at most $2t \log_2(ML) + o(\log_2(ML))$ for small t (e.g., a constant), and

at most linear in $t\log_2(ML/t)$ for large t (e.g., a fraction of ML). In addition, the optimal redundancy is at least $t\log_2(ML) + o(\log_2(ML))$ for small t, and at least linear in $t\log_2(ML/t)$ for large t. The upper and lower bounds are orderwise the same. Note that in the classic setting of correcting t substitution errors over a single sequence of length ML (which is equivalent to M ordered sequences of length L), similar counting arguments also yield $2t\log_2(ML) + o(\log_2(ML))$ for small t, and linear in $t\log_2(ML/t)$ for large t. In addition, the lower bound on redundancy is orderwise the same as the upper bound. This result has a surprising implication: it costs almost the same amount of redundancy to correct errors over a set of unordered and ordered sequences! This is highly surprising, since the unordered case is intuitively more complex.

We now compare the redundancy bound $2t\log_2(ML) + o(\log_2(ML))$ to that of the index-based schemes presented earlier, which is linear in M. Since M is much larger than L, the redundancy in the index-based schemes is much larger than bound $2t\log_2(ML) + o(\log_2(ML))$ whenever

$$t = o\left(\frac{M}{\log_2(ML)}\right).$$

In what follows we describe ideas that close this gap, i.e., provide codes correcting substitution errors with almost optimal redundancy, and thus are better than index-based schemes for this many substitution errors.

The idea is to use the data itself for the purpose of indexing, or alternatively, encoding data inside the index. Specifically, we use the lexicographic order of the data for indexing, that is, the prefix in each sequence is used for indexing, while also containing data.

Example 5. Let the codeword be $\{1001101, 0101100, 1010001, 0001001\}$, where the first three bits are the prefix in each sequence used for indexing. Then one can order the set of sequences in the codeword by 0001001,0101100,1001101,1010001 in ascending lexicographic order of the prefixes.

Using prefixes for indexing is similar to index-based schemes. Yet, unlike index-based schemes, the prefixes also encode information. In order to make the indexing of the prefixes robust from substitution errors, the collection of prefixes in all sequences constitutes a code with large minimum Hamming distance. This is similar to protecting the indices from errors in the index-based schemes. The difference is that the indexing prefixes here encode information. Information is encoded into prefixes through different choices of the codes, as shown in Figure 2. The construction of codes for the prefixes can be done using a greedy algorithm, so that the prefixes in the code are generated bit-by-bit. The scheme of using data to index avoids

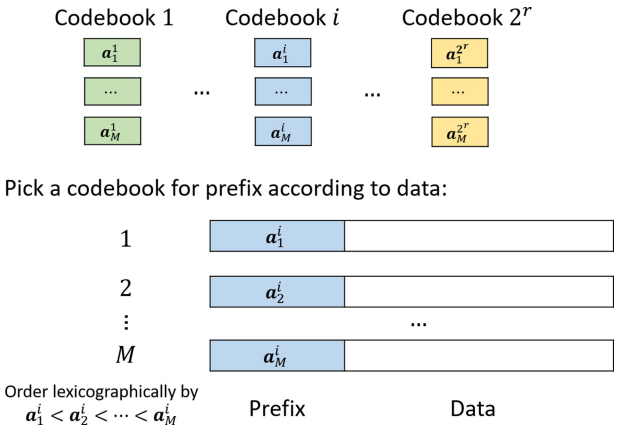

Figure 2
Illustration of codes that use data for indexing.

the index bits, and achieves redundancy that is linear in $t\log_2(ML)$, i.e., almost optimal. One can also combine it with the deletion correcting codes discussed in Section II, and enable deletion and/or insertion correction as well. Finally, some questions about the unordered setting remain unanswered: What is the optimal redundancy for a *large* number of errors? How to construct efficient codes that achieve orderwise optimal redundancy?

Duplication

Unlike previous sections, this section is motivated by storing information in the DNA of living organisms. The process involves synthesizing DNA sequences, which are then inserted into the DNA of living organisms. We can then sequence the DNA extracted from these organisms, or more likely, their descendants, to read the information. Thus, the DNA storage channel in this case corrupts data not only due to synthesis and sequencing errors, but also due to naturally occurring biological processes that mutate the DNA.

We now focus on the errors (mutations) introduced by biological processes. It is well known that when cells divide, the genetic material is replicated. However, the DNA replication process is not without noise, and the resulting copy may be corrupted by several error types. These include substitution, where a base is replaced by another (point mutation), as well as insertions and deletions of blocks. These types of errors have been studied to various extents by existing literature. Another type of error is duplication, whereby a copy of a substring of the DNA is inserted. Duplications accumulate over time, and it has been found that the majority of human DNA is duplicated. Since this error type is rarely found in electronic communication, it has not been studied in the coding theory community, and in what follows, we focus on it solely.

What kind of duplications are possible? Several biological mechanisms are known to create duplications in the process

of replicating the DNA. We illustrate a few here. Perhaps the simplest one (though not necessarily the most common) is *tandem duplication*. This mutation process takes a substring and inserts a duplicate of it immediately after its original location, for example

$$\mathsf{ACCTAGGA} \Longrightarrow \mathsf{AC}\underline{\mathsf{CTA}}\overline{CTA}GGA \quad \text{(tandem)}$$

where the underlined part is the substring being duplicated, and the overlined part is the inserted duplication. In *interspersed duplication*, the duplicated part is inserted anywhere in the sequence, for example

$$\mathsf{ACCTAGGA} \Longrightarrow \mathsf{AC}\underline{\mathsf{CTA}}GG\overline{CTA}A \quad \text{(interspersed)}.$$

Another duplication process, called *reverse-complement duplication* (r.c. duplication), takes the substring to duplicate, and inserts a reversed and complemented duplicate of it immediately after its location, for example

$$\mathsf{ACCTAGGA} \Longrightarrow \mathsf{AC}\underline{\mathsf{CTA}}\overline{TAG}GGA \quad \text{(r.c.)}.$$

Here we use the Watson–Crick base pairing, making A and T complements of each other, and similarly, C and G. All of the processes mentioned above are the result of known biological mutation processes whose mechanisms we understand. For the sake of mathematical simplicity, and to better illustrate the intricacies of duplication processes, we introduce an artificial duplication process called *end duplication* which inserts the duplicated part at the end of the sequence, for example

$$\mathsf{ACCTAGGA} \Longrightarrow \mathsf{AC}\underline{\mathsf{CTA}}GGA\overline{CTA} \quad \text{(end)}.$$

As a final note on duplication processes, we emphasize that following a duplication process, another may occur, perhaps of a different type, and perhaps of a different length. Thus, over time, duplications accumulate, like layers of an onion. A naive inspection of a DNA sequence may only reveal the outer layer, namely, the last duplications made, and only after removing those, older occurrences become visible.

We can now formalize the description of the duplication channel. We shall be working over some finite alphabet Σ (which in the case of DNA molecules will be $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$). We store a sequence $x \in \Sigma^*$, where Σ^* is the set of all finite length sequences over Σ. The channel then applies any number of duplications, resulting in a string $y \in \Sigma^*$. We denote this process as $x \Longrightarrow^* y$. The set of all possible mutated outcomes, given that x was stored, is called the *descendant cone of* x, and is denoted by $D^*(x)$. Conversely, the set of all possible strings that may be mutated by the channel into y is called the *ancestor cone of* y, and is denoted by $A^*(y)$.

When faced with such a channel, our goal is to construct error-correcting codes that can undo the duplications and recover the original stored sequence. General coding-theoretic principles guide us to define an error-correcting code as a set of sequences $C \subseteq \Sigma^n$, whose descendant cones are disjoint, namely, for any $c, c' \in C$, $D^*(c) \cap D^*(c') = \emptyset$. Thus, any corrupted sequence belongs to a single descendant cone of a valid codeword, and the decoding process simply outputs that codeword in response.

Many questions arise: How do we find a good error-correcting code? What makes a good error-correcting code? What is the best possible? How do we encode, and how do we decode? How does the answer depend on the type and parameters of the duplication processes? In what follows we briefly outline partial answers to these questions, and along the way, uncover connections to other motivating problems.

Know Thy Enemy—Understanding Descendant Cones

The first property of interest, when studying descendant cones, is knowing their size. Since our ultimate goal is constructing error-correcting codes, which are equivalent to packing descendant cones without overlap, finding their size may help us bound the parameters of such codes. The number of strings in any descendant cone is obviously infinite, and thus, we do not measure their size but rather the rate at which they grow with each mutation step. This property is called the *capacity*, and sometimes the *combinatorial entropy*.

Formally speaking, to compute the capacity of the descendant cone of $x \in \Sigma^*$, the definition calls for counting the number of descendants of length n, i.e., $|D^*(x) \cap \Sigma^n|$. Taking $\log_2$ of this number and dividing by n gives us the exponential growth rate we are after. Thus

$$\mathsf{cap}(x) = \limsup_{n\to\infty} \frac{1}{n} \log_2 |*|D^*(x) \cap \Sigma^n.$$

A large capacity indicates a fast growing descendant cone, and similarly, a small capacity indicates slow growth. Packing fast growing descendant cones may be more difficult, resulting in smaller, less efficient, error-correcting codes.

The capacity may obviously depend on the alphabet size, the starting sequence x, and the duplication rules. To illustrate the subtleties of the latter, fix an alphabet Σ and a starting sequence x. First consider the end-duplication system, in which each mutation copies a fixed-length substring of length k to the end. It has been shown that this has full capacity, i.e., $\mathsf{cap}_k^{\text{end}}(x) = \log_2|\Sigma|$, which is the highest possible value the capacity may have, indicating the highest possible growth rate for a descendant cone. We now tweak a single parameter—instead of end duplication, we consider tandem duplication, namely the duplicated sequence of fixed length k is inserted immediately after its original position. With this minute change, the capacity vanishes completely, i.e., $\mathsf{cap}_k^{\text{tan}}(x) = 0$, indicating subexponential growth of the descendant cone.

One might argue that the capacity is perhaps too harsh: it takes into account what is *possible*, where instead it should take into account what is *probable*. We can describe the mutations as a stochastic process. We start with the initial sequence $x \in \Sigma^*$, and then at each round, a randomly chosen duplication rule (duplicating substring of fixed length k) is applied to a randomly selected position. We can set, to our liking, the distributions from which the duplication rule and the location are chosen. We denote the resulting sequence after n mutations by $S_n(x)$, and observe that it is a random variable. We can then define the *entropy* of $S_n(x)$ as

$$H(S_n(x)) = -\sum_{w \in \Sigma^*} \Pr(S_n(x) = w) \log_2 \Pr(S_n(x) = w).$$

With this, the *entropy rate* of the entire system

$$h(S(x)) = \limsup_{n \to \infty} \frac{1}{n} H(S_n(x)).$$

Loosely speaking, $h(S(x))$ measures the amount of information generated by an application of a random duplication rule. Using standard information-theoretic arguments, one can show that the capacity bounds the entropy rate from above, namely

$$h(S(x)) \leq \mathsf{cap}(x).$$

Once again, we demonstrate the intricacies of string-duplication systems by showing how even the smallest of changes create dramatically different results. For the sake of this demonstration, we focus on the reverse-complement string duplication system over the binary alphabet $\Sigma = \{0, 1\}$. We further assume for simplicity that the initial string is $x = 0$, all duplications are of the same fixed length $k = 1$, and that their location is chosen uniformly and independently in each round. We emphasize that the fact the locations are chosen uniformly does not mean that $S_n(x)$ is distributed uniformly. For example, there is only one way of deriving 0111 from $x = 0$

$$0 \Longrightarrow \underline{0\bar{1}} \Longrightarrow \underline{0\bar{1}}1 \Longrightarrow \underline{0\bar{1}}11$$

and the probability of this happening is exactly $1 \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$. However, there are two ways of deriving 0101

$$0 \Longrightarrow \underline{0\bar{1}} \Longrightarrow \underline{0\bar{1}}1 \Longrightarrow 0\underline{1\bar{0}}1$$
$$0 \Longrightarrow \underline{0\bar{1}} \Longrightarrow 0\underline{1\bar{0}} \Longrightarrow 0\underline{1\bar{0}}1$$

and we get 0101 with probability $\frac{1}{3}$. Interestingly, the entropy-rate we are after is connected to the asymptotics of permutation signatures, and the best we know is that for duplication length $k = 1$

$$0.8689 \leq h(S(x = 0)) \leq 0.9067 \; < \; \mathsf{cap}(x = 0) = 1.$$

Importantly, while the capacity is full (i.e., "most" sequences are obtainable via carefully chosen derivation paths), not all outcomes are probable, and the entropy rate is strictly less than 1. To further complicate matters, consider duplication of length $k = 2$ and an equally long starting sequence, $x = 00$. In this case, it has been shown that

$$0 = h(S(x = 00)) \; < \; \mathsf{cap}(x = 00) = 1.$$

Namely, while again, "most" sequences are obtainable, only very few are probable. This surprising result was obtained by proving that with high probability, $S_n(x = 00)$ is eventually almost entirely an alternating sequence of $0101\ldots$. A simulation of this fact is shown in Figure 3.

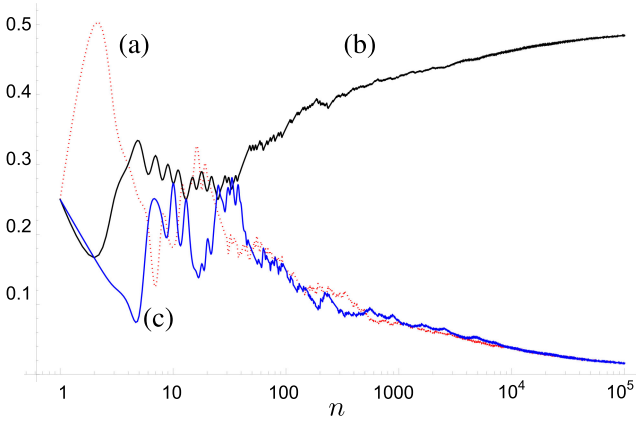

Figure 3

Example simulation of the reverse-complement string-duplication system with starting sequence $x = 00$, and duplication length $k = 2$, showing for $S_n(00)$ the frequencies of the substrings (a) 00, (b) 01 (which equals that of 10), and (c) 11.

The last two properties we shall describe are perhaps more interesting from a bio-informatics perspective. Consider the following question: We are given some distant ancestor that humans evolved from, and this ancestor does not have the DNA substring that codes for a specific protein humans have. Can tandem duplication *alone* mutate the ancestor's DNA sequence into a sequence that contains the instructions for that protein? In our mathematical framework of string-duplication systems, we say a system with a starting sequence $x \in \Sigma^*$ is *fully expressive* if any given sequence $y \in \Sigma^*$ appears as a substring of some descendant of x. Returning to our previous example of end duplication versus tandem duplication (both of some fixed length k), it was shown that tandem duplication is not fully expressive, whereas end duplication is. To put this in context, imagine the following challenge: We are given Tolstoy's "War and Peace" (which we consider as a very long sequence of symbols). We can duplicate substrings of some fixed length, say, $k = 200$. Our goal is to create a substring, which is Shakespeare's "Macbeth." When the duplicated parts are inserted next to their original position (tandem duplication), this is impossible. However, when the

duplicated parts are placed at the end, the challenge is solvable (albeit, the procedure might be extremely lengthy)!

The final property we would like to mention is that of distance to the root. When reading a mutated version $y \in \Sigma^*$ of the stored sequence $x \in \Sigma^*$, our goal is to reverse the mutation process and find x. This may be performed by undoing duplications, a process called *deduplication*. The process stops when no further deduplications are possible, and the sequence at that point is called a *root* of y. The number of deduplication steps is called the distance to a root from y. In the binary case $\Sigma = \{0, 1\}$, with unbounded tandem duplication, the root is always one of six options $\{0, 1, 01, 10, 010, 101\}$. In this setting, we let $f(n)$ denote the maximum distance to the root of a binary sequence of length n. Surprisingly

$$0.045 \leq \lim_{n\to\infty} \frac{f(n)}{n} \leq 0.4$$

and the lower bound in fact holds for all but an exponentially small fraction of sequences of length n. Thus, the vast majority of sequences have a distance to the root that is linear in their length. This result remains essentially the same even if the duplication process is imprecise.

Constructing Error-Correcting Codes

Armed with a better understanding of descendant cones, we may now approach the problem of designing error-correcting codes for string-duplication channels. Two main issues are of interest: finding a good code (i.e., making sure descendant cones of distinct codewords are disjoint), and finding an efficient decoding algorithm. Throughout this section, we shall consider the tandem-duplication string-duplication channel as an example.

When the duplication length is fixed at some length k, we are indeed fortunate. If the alphabet contains exactly q letters, we may assume without loss of generality that $\Sigma = \mathbb{Z}_q$, namely, the ring of integers with addition modulo q. It has been suggested to view any string after taking the k-step discrete derivative ∂_k, i.e., for each i, subtracting the letter in position $i-k$ from the letter in position i. Thus, $\partial_k x = x0^k - 0^k x$, where 0^k denotes a run of k zeros, and subtraction is symbolwise over $\mathbb{Z}_q$. This operation is invertible. Moreover, in the derivative domain, a tandem duplication of length k manifests as an insertion of 0^k. As a consequence, we obtain the following:

- Any sequence $x \in \Sigma^*$ has a unique root.
- The unique root of a sequence x may be reached by deduplications performed in any order.
- Two sequences, x and x', have intersecting descendant cones if and only if they have the same root.

We note in passing that these assertions are not true even if we relax our setting minutely. For example, if we allow deduplications of any length (instead of a fixed length) then the sequence 210121010 does not have a unique root:

$$210\underline{12101}0 \longrightarrow 2101\underline{0} \longrightarrow 210$$
$$21012101\underline{0} \longrightarrow 2101210$$

where $\longrightarrow$ denotes a deduplication, and the underlined part is dedeuplicated.

Returning to our search for codes that correct tandem duplications of fixed length k, the three properties listed above lead us to the following solution: Construct a code by taking as codewords all the irreducible sequences of length n over $\mathbb{Z}_q$ (where an irreducible sequence is a sequence that is its own root, namely, it does not contain any duplications of length k). With a small tweak this code can be made optimal, and allows information storage at a rate of

$$\log_2 q - \frac{(q-1)\log_2 e}{q^{k+1}}(1 + o(1)).$$

Decoding is simple, since by the properties above, we can deduplicate in any order we wish, until reaching the unique root which must be the transmitted sequence.

In essence, since the duplications introduced by biological processes are part of the evolutionary process, we can think of the error-correcting codes we described as evolution-correcting codes.

Bibliographic Notes

There are several implementations [3], [9], [10], [14], [21], [24], [31], [37], [38], [77], [86], [99], [117], [118], [123] of in-vitro and in-vivo DNA storage demonstrating their potential and motivating the error and channel models considered in this article. A detailed description of the errors and the channel can be found in [45], [60]. We also refer to [28], [88], [119] for a broader overview of different aspects in DNA storage.

Deletion Codes

The study of codes correcting deletions and insertions was introduced in the seminal papers [69], [84], where it was shown in [69] that deletion codes correct a combination of deletions and insertions. The upper and lower bounds on the optimal redundancy of deletion codes were also given in [69]. The VT codes were proposed in [109]. An algebraic generalization of VT codes with redundancy linear in code length was presented in [47] and further extended in [44]. The first codes correcting a number of linear in code length deletions based on concatenated code structures were proposed in [83] and were improved in [41] and [40]. Using the

concatenated code construction, Brakensiek et al. [11] proposed the first codes correcting a small number of deletions with redundancy logarithmic in the code length. For two deletions, the result in [11] was improved by [36], [39], [92]. The first-orderwise optimal codes correcting a constant number of deletions were given by [18], [90], [91].

The algebraic generalizations of VT codes discussed in Section II for correcting deletions were presented in [90], [91], and [92]. Compared to the VT generalizations in [44] and [47] that require linear redundancy, the generalizations in [90], [91], and [92] are capable of correcting a constant number of deletions with asymptotically at most four times the optimal redundancy, which is a logarithm of the code length. Combining the codes in [91] and the VT codes, Song et al. [97] further improved the redundancy in [91] from

$$4t\log n + o(\log n)$$

to

$$(4t-1)\log n + o(\log n),$$

where t is the number of deletions and n is the code length. Besides the above code constructions, existential bounds improving the results in [69] were recently presented in [2].

Other related problems include: systematic deletion codes [5], [18], [22], [42], [78], [91], nonbinary deletion codes [23], [43], [62], [70], [71], [90], [107], deletion codes with randomized decoding [6], [13], [48], [51], channel capacity of deletion channels [19], [25], [26], [29], [52], [53], [55], [100], [108], [110], codes correcting a combination of deletions, insertions, substitutions, and transpositions [12], [17], [34], [36], codes correcting a burst of deletions [63], [82], [98], [112], codes for sticky insertions [27], [73], and codes correcting asymmetric deletions [101], [113]. In addition, the application of edit distance in natural language processing and biological data analysis can be found in [81] and [116]. See [20], [75], [76], [95] for a broader review of this topic.

Sliced Channel

The model of encoding information into a set of unordered and equal-length sequences was introduced in [46], where it was shown that index-based schemes achieve the channel capacity when there are sequence losses. Later, the channel capacity analysis was extended to channels with both sequence loss and substitution errors [66], [67], [87], [115]. The protection of indices against errors assuming index-based schemes was addressed in [64], [85], and [96].

The definition of redundancy measuring the extra redundancy needed for error protection was introduced in [65], where it was shown that the redundancy for index-based schemes is linear in the number of sequences. The orderwise optimal redundancy under this definition was obtained in [94], where the idea of using data for indexing was proposed. By using this idea, code constructions were presented in [93] to achieve orderwise optimal redundancy and in [114] to obtain improved results upon those in [65].

Other related problems include: permutation channels [56], [57], [61], [74], [102], [111], codes for reconstruction from substrings [15], [35], [54], [80], [120], and torn-paper channels [4], [79], [89].

Duplication

Early attempts of in-vivo information storage can be found in [24] and [117]. Proofs of concepts for storing information in the DNA of living organisms were provided in [86] and [123]. In [60], it was pointed out that the majority of the human DNA contains duplications. The string-duplication channels were introduced in [32] and the capacity and expressiveness of several duplication rules were studied. These results were extended in [49] and [58]. The stochastic channel model, called a Pólya string model, was introduced in [30], and further studied in [7], [33], and [72]. In particular, Farnoud et al. [33] developed a parameter-estimation scheme based on this model. The distance to the root in tandem duplication channels was studied in [1].

Error-correcting codes for string-duplication channels were first studied in [50]. The work was followed by many others, among them works studying: tandem duplication [16], [59], [68], [124], [125], Levenshtein reconstruction for uniform tandem duplication [121], [122], noisy tandem duplication [103], [104], [105], [106], palindromic duplications [68], [125], and reverse-complement duplications [7].

Acknowledgment

This work was supported by National Science Foundation under Grant CCF-1816965 and Grant CCF-1717884.

References

[1] N. Alon, J. Bruck, F. Farnoud, and S. Jain, "Duplication distance to the root for binary sequences," *IEEE Trans. Inf. Theory*, vol. 63, no. 12, pp. 7793–7803, Dec. 2017.

[2] N. Alon, G. Bourla, B. Graham, X. He, and N. Kravitz, "Logarithmically larger deletion codes of all distances," *IEEE Trans. Inf. Theory*, to be published, doi: 10.1109/TIT.2023.3304565.

[3] P. L. Antkowiak et al., "Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction," *Nature Commun.*, vol. 11, no. 1, 2020, Art. no. 5345.

[4] D. Bar-Lev, S. M. E. Yaakobi, and Y. Yehezkeally, "Adversarial torn-paper codes," *IEEE Trans. Inf. Theory*, vol. 69, no. 10, pp. 6414–6427, Oct. 2023.

[5] D. Belazzougui, "Efficient deterministic single round document exchange for edit distance," 2015, *arXiv:1511.09229*.

[6] D. Belazzougui and Q. Zhang, "Edit distance: Sketching, streaming, and document exchange," in *Proc. IEEE 57th Annu. Symp. Found. Comput. Sci.*, 2016, pp. 51–60.

[7] E. Ben-Tolila and M. Schwartz, "On the reverse-complement string-duplication system," *IEEE Trans. Inf. Theory*, vol. 68, no. 11, pp. 7184–7197, Nov. 2022.

[8] E. R. Berlekamp, "The technology of error-correcting codes," *Proc. IEEE*, vol. 68, no. 5, pp. 564–593, May 1980.

[9] M. Blawat et al., "Forward error correction for DNA data storage," *Procedia Comput. Sci.*, vol. 80, pp. 1011–1022, 2016.

[10] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A DNA-based archival storage system," in *Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2016, pp. 637–649.

[11] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Trans. Inf. Theory*, vol. 64, no. 5, pp. 3403–3410, May 2018.

[12] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, "Correcting a single indel/edit for DNA-based data storage: Linear-time encoders and order-optimality," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3438–3451, Jun. 2021.

[13] D. Chakraborty, E. Goldenberg, and M. Koucký, "Low distortion embedding from edit to hamming distance using coupling," *Electron. Colloq. Comput. Complexity*, vol. 22, 2015, Art. no. 111.

[14] S. Chandak et al., "Improved read/write cost tradeoff in DNA-based data storage using ldpc codes," in *Proc. IEEE 57th Annu. Allerton Conf. Commun., Control, Comput.*, 2019, pp. 147–156.

[15] Z. Chang, J. Chrisnata, M. F. Ezerman, and H. M. Kiah, "Rates of DNA sequence profiles for practical values of read lengths," *IEEE Trans. Inf. Theory*, vol. 63, no. 11, pp. 7166–7177, Nov. 2017.

[16] Y. M. Chee, J. Chrisnata, H. M. Kiah, and T. T. Nguyen, "Efficient encoding/decoding of GC-balanced codes correcting tandem duplications," *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 4892–4903, Aug. 2020.

[17] K. Cheng, Z. Jin, X. Li, and K. Wu, "Block edit errors with transpositions: Deterministic document exchange protocols and almost optimal binary codes," 2018, *arXiv:1809.00725*.

[18] K. Cheng, Z. Jin, X. Li, and K. Wu, "Deterministic document exchange protocols and almost optimal binary codes for edit errors," *J. ACM*, vol. 69, no. 6, pp. 1–39, 2022.

[19] M. Cheraghchi, "Capacity upper bounds for deletion-type channels," *J. ACM*, vol. 66, no. 2, pp. 1–79, 2019.

[20] M. Cheraghchi and J. Ribeiro, "An overview of capacity results for synchronization channels," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3207–3232, Jun. 2021.

[21] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[22] G. Cormode, M. S. Paterson, S. C. Sahinalp, and U. Vishkin, "Communication complexity of document exchange," 1999.

[23] D. Cullina and N. Kiyavash, "An improvement to Levenshtein's upper bound on the cardinality of deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 3862–3870, Jul. 2014.

[24] J. Davis, "Microvenus," *Art J.*, vol. 55, no. 1, pp. 70–74, 1996.

[25] S. N. Diggavi and M. Grossglauser, "On transmission over deletion channels," in *Proc. Annu. Allerton Conf. Commun. Control Comput.*, 2001, vol. 39, no. 1, pp. 573–582.

[26] R. L. Dobrushin, "Shannon's theorems for channels with synchronization errors," *Problemy Peredachi Informatsii*, vol. 3, no. 4, pp. 18–36, 1967.

[27] L. Dolecek and V. Anantharam, "Repetition error correcting sets: Explicit constructions and prefixing methods," *SIAM J. Discrete Math.*, vol. 23, no. 4, pp. 2120–2146, 2010.

[28] Y. Dong, F. Sun, Z. Ping, Q. Ouyang, and L. Qian, "DNA storage: Research landscape and future prospects," *Nat. Sci. Rev.*, vol. 7, no. 6, pp. 1092–1107, 2020.

[29] E. Drinea and M. Mitzenmacher, "Improved lower bounds for the capacity of iid deletion and duplication channels," *IEEE Trans. Inf. Theory*, vol. 53, no. 8, pp. 2693–2714, Aug. 2007.

[30] O. Elishco, F. Farnoud, M. Schwartz, and J. Bruck, "The entropy rate of some Pólya string models," *IEEE Trans. Inf. Theory*, vol. 65, no. 12, pp. 8180–8193, Dec. 2019.

[31] Y. Erlich and D. Zielinski, "DNA fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, pp. 950–954, 2017.

[32] F. Farnoud, M. Schwartz, and J. Bruck, "The capacity of string-duplication systems," *IEEE Trans. Inf. Theory*, vol. 62, no. 2, pp. 811–824, Feb. 2016.

[33] F. Farnoud, M. Schwartz, and J. Bruck, "Estimation of duplication history under a stochastic model for tandem repeats," *BMC Bioinf.*, vol. 20, no. 64, pp. 1–11, Feb. 2019.

[34] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu, "Beyond single-deletion correcting codes: Substitutions and transpositions," *IEEE Trans. Inf. Theory*, vol. 69, no. 1, pp. 169–186, Jan. 2023.

[35] R. Gabrys and O. Milenkovic, "Unique reconstruction of coded sequences from multiset substring spectra," in *Proc. IEEE Int. Symp. Inf. Theory*, 2018, pp. 2540–2544.

[36] R. Gabrys, E. Yaakobi, and O. Milenkovic, "Codes in the Damerau distance for deletion and adjacent transposition correction," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2550–2570, Apr. 2018.

[37] N. Goldman et al., "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[38] 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," *Angewandte Chemie Int. Ed.*, vol. 54, no. 8, pp. 2552–2555, 2015.

[39] V. Guruswami and J. Håstad, "Explicit two-deletion codes with redundancy matching the existential bound," *IEEE Trans. Inf. Theory*, vol. 67, no. 10, pp. 6384–6394, Oct. 2021.

[40] V. Guruswami and R. Li, "Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2016, pp. 620–624.

[41] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 1961–1970, Apr. 2017.

[42] B. Haeupler, "Optimal document exchange and new codes for insertions and deletions," in *Proc. IEEE 60th Annu. Symp. Found. Comput. Sci.*, 2019, pp. 334–347.

[43] B. Haeupler and A. Shahrasbi, "Synchronization strings: Codes for insertions and deletions approaching the singleton bound," in *Proc. 49th Annu. ACM SIGACT Symp. Theory Comput.*, 2017, pp. 33–46.

[44] M. Hagiwara, "On ordered syndromes for multi insertion/deletion error-correcting codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2016, pp. 625–629.

[45] R. Heckel, G. Mikutis, and R. N. Grass, "A characterization of the DNA data storage channel," *Sci. Rep.*, vol. 9, no. 1, pp. 1–12, 2019.

[46] R. Heckel, I. Shomorony, K. Ramchandran, and N. David, "Fundamental limits of DNA storage systems," in *Proc. IEEE Int. Symp. Inf. Theory*, 2017, pp. 3130–3134.

[47] A. S. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 1, pp. 305–308, Jan. 2002.

[48] U. Irmak, S. Mihaylov, and T. Suel, "Improved single-round protocols for remote file synchronization," in *Proc. IEEE 24th Annu. Joint Conf. Comput. Commun. Soc.*, 2005, vol. 3, pp. 1665–1676.

[49] S. Jain, F. Farnoud, and J. Bruck, "Capacity and expressiveness of genomic tandem duplication," *IEEE Trans. Inf. Theory*, vol. 63, no. 10, pp. 6129–6138, Oct. 2017.

[50] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the DNA of living organisms," *IEEE Trans. Inf. Theory*, vol. 63, no. 8, pp. 4996–5010, Aug. 2017.

[51] H. Jowhari, "Efficient communication protocols for deciding edit distance," in *Proc. Algorithms–ESA 20th Annu. Eur. Symp.*, 2012, pp. 648–658.

[52] A. Kalai, M. Mitzenmacher, and M. Sudan, "Tight asymptotic bounds for the deletion channel with small deletion probabilities," in *Proc. IEEE Int. Symp. Inf. Theory*, 2010, pp. 997–1001.

[53] Y. Kanoria and A. Montanari, "Optimal coding for the binary deletion channel with small deletion probability," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6192–6219, Oct. 2013.

[54] H. M. Kiah, G. J. Puleo, and O. Milenkovic, "Codes for DNA sequence profiles," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 3125–3146, Jun. 2016.

[55] A. Kirsch and E. Drinea, "Directly lower bounding the information capacity for channels with I.I.D. deletions and duplications," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 86–102, Jan. 2010.

[56] M. Kovačević and V. Y. Tan, "Codes in the space of multisets–coding for permutation channels with impairments," *IEEE Trans. Inf. Theory*, vol. 64, no. 7, pp. 5156–5169, Jul. 2018.

[57] M. Kovačević and D. Vukobratović, "Perfect codes in the discrete simplex," *Des., Codes Cryptogr.*, vol. 75, pp. 81–95, 2015.

[58] M. Kovačević, "Zero-error capacity of duplication channels," *IEEE Trans. Commun.*, vol. 67, no. 10, pp. 6735–6742, Oct. 2019.

[59] M. Kovačević and V. Y. F. Tan, "Asymptotically optimal codes correcting fixed-length duplication errors in DNA storage systems," *IEEE Commun. Lett.*, vol. 22, no. 11, pp. 2194–2197, Nov. 2018.

[60] E. S. Lander et al., "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.

[61] M. Langberg, M. Schwartz, and E. Yaakobi, "Coding for the ℓ_∞-limited permutation channel," *IEEE Trans. Inf. Theory*, vol. 63, no. 12, pp. 7676–7686, Dec. 2017.

[62] T. A. Le and H. D. Nguyen, "New multiple insertion-deletion correcting codes for non-binary alphabets," 2015, *arXiv:1502.02727*.

[63] A. Lenz and N. Polyanskii, "Optimal codes correcting a burst of deletions of variable length," in *Proc. IEEE Int. Symp. Inf. Theory*, 2020, pp. 757–762.

[64] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Anchor-based correction of substitutions in indexed sets," in *Proc. IEEE Int. Symp. Inf. Theory*, 2019, pp. 757–761.

[65] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding over sets for DNA storage," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2331–2351, 2019.

[66] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "An upper bound on the capacity of the DNA storage channel," in *Proc. IEEE Inf. Theory Workshop*, 2019, pp. 1–5.

[67] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Achieving the capacity of the DNA storage channel," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2020, pp. 8846–8850.

[68] A. Lenz, A. Wachter-Zeh, and E. Yaakobi, "Duplication-correcting codes," *Des., Codes Cryptogr.*, vol. 87, pp. 277–298, 2019.

[69] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Proc. Sov. Phys. Doklady, Sov. Union*, 1966, vol. 10, no. 8, pp. 707–710.

[70] V. I. Levenshtein, "Bounds for deletion/insertion correcting codes," in *Proc. IEEE Int. Symp. Inf. Theory*,, 2002, Art. no. 370.

[71] S. Liu and C. Xing, "Bounds and constructions for insertion and deletion codes," *IEEE Trans. Inf. Theory*, vol. 69, no. 2, pp. 928–940, Feb. 2022.

[72] H. Lou, M. Schwartz, J. Bruck, and F. Farnoud, "Evolution of k-MER frequencies and entropy in duplication and substitution mutation systems," *IEEE Trans. Inf. Theory*, vol. 66, no. 5, pp. 3171–3186, May 2020.

[73] H. Mahdavifar and A. Vardy, "Asymptotically optimal sticky-insertion-correcting codes with efficient encoding and decoding," in *Proc. IEEE Int. Symp. Inf. Theory*, 2017, pp. 2683–2687.

[74] A. Makur, "Coding theorems for noisy permutation channels," *IEEE Trans. Inf. Theory*, vol. 66, no. 11, pp. 6723–6748, Nov. 2020.

[75] H. Mercier, V. K. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 1, pp. 87–96, First Quarter 2010.

[76] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," 2009.

[77] L. Organick et al., "Scaling up DNA data storage and random access retrieval," *BioRxiv*, 2017, Art. no. 114553.

[78] A. Orlitsky, "Interactive communication of balanced distributions and of correlated files," *SIAM J. Discrete Math.*, vol. 6, no. 4, pp. 548–564, 1993.

[79] A. N. Ravi, A. Vahid, and I. Shomorony, "Capacity of the torn paper channel with lost pieces," in *Proc. IEEE Int. Symp. Inf. Theory*, 2021, pp. 1937–1942.

[80] N. Raviv, M. Schwartz, and E. Yaakobi, "Rank-modulation codes for DNA storage with shotgun sequencing," *IEEE Trans. Inf. Theory*, vol. 65, no. 1, pp. 50–64, Jan. 2019.

[81] D. Sankoff and J. B. Kruskal, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA, USA:Addison-Wesley Publication, 1983.

[82] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes correcting a burst of deletions or insertions," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 1971–1985, Apr. 2017.

[83] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Trans. Inf. Theory*, vol. 45, no. 7, pp. 2552–2557, Nov. 1999.

[84] F. Sellers, "Bit loss and gain correction code," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 35–38, 1962.

[85] T. Shinkar, E. Yaakobi, A. Lenz, and A. Wachter-Zeh, "Clustering-correcting codes," *IEEE Trans. Inf. Theory*, vol. 68, no. 3, pp. 1560–1580, Mar. 2021.

[86] S. L. Shipman, J. Nivala, J. D. Macklis, and G. M. Church, "CRISPR-Cas encoding of digital movie into the genomes of a population of living bacteria," *Nature*, vol. 547, pp. 345–349, Jul. 2017.

[87] I. Shomorony and R. Heckel, "DNA-based storage: Models and fundamental limits," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3675–3689, Jun. 2021.

[88] I. Shomorony et al., "Information-theoretic foundations of DNA data storage," *Found. Trends Commun. Inf. Theory*, vol. 19, no. 1, pp. 1–106, 2022.

[89] I. Shomorony and A. Vahid, "Torn-paper coding," *IEEE Trans. Inf. Theory*, vol. 67, no. 12, pp. 7904–7913, 2021.

[90] J. Sima and J. Bruck, "On optimal k-deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3360–3375, Jun. 2021.

[91] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2020, pp. 769–774.

[92] J. Sima, N. Raviv, and J. Bruck, "Two deletion correcting codes from indicator vectors," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2375–2391, Apr. 2020.

[93] J. Sima, N. Raviv, and J. Bruck, "Robust indexing-optimal codes for DNA storage," in *Proc. IEEE Int. Symp. Inf. Theory*, 2020, pp. 717–722.

[94] J. Sima, N. Raviv, and J. Bruck, "On coding over sliced information," *IEEE Trans. Inf. Theory*, vol. 67, no. 5, pp. 2793–2807, May 2021.

[95] N. J. Sloane, "On single-deletion-correcting codes," *Codes Designs*, vol. 10, pp. 273–291, 2002.

[96] W. Song, K. Cai, and K. A. S. Immink, "Sequence-subset distance and coding for error control in DNA-based data storage," *IEEE Trans. Inf. Theory*, vol. 66, no. 10, pp. 6048–6065, Oct. 2020.

[97] W. Song, N. Polyanskii, K. Cai, and X. He, "On multiple-deletion multiple-substitution correcting codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2021, pp. 2655–2660.

[98] Y. Sun, Y. Zhang, and G. Ge, "Improved constructions of permutation and multi-permutation codes correcting a burst of stable deletions," *IEEE Trans. Inf. Theory*, vol. 69, no. 7, pp. 4429–4441, Jul. 2023.

[99] S. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system," *Sci. Rep.*, vol. 5, no. 1, pp. 1–10, 2015.

[100] I. Tal, H. D. Pfister, A. Fazeli, and A. Vardy, "Polar codes for the deletion channel: Weak and strong polarization," *IEEE Trans. Inf. Theory*, vol. 68, no. 4, pp. 2239–2265, Apr. 2022.

[101] L. G. Tallini, N. Alqwaifly, and B. Bose, "Deletions and insertions of the symbol "0" and asymmetric/unidirectional error control codes for the l metric," *IEEE Trans. Inf. Theory*, vol. 69, no. 1, pp. 86–106, Jan. 2023.

[102] J. Tang and Y. Polyanskiy, "Capacity of noisy permutation channels," *IEEE Trans. Inf. Theory*, vol. 69, no. 7, pp. 4145–4162, Jul. 2023.

[103] Y. Tang and F. Farnoud, "Error-correcting codes for noisy duplication channels," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3452–3464, Jun. 2021.

[104] Y. Tang and F. Farnoud, "Error-correcting codes for short tandem duplication and edit errors," *IEEE Trans. Inf. Theory*, vol. 68, no. 2, pp. 871–880, Feb. 2021.

[105] Y. Tang, S. Wang, H. Lou, R. Gabrys, and F. Farnoud, "Low-redundancy codes for correcting multiple short-duplication and edit errors," *IEEE Trans. Inf. Theory*, vol. 69, no. 5, pp. 2940–2954, May 2023.

[106] Y. Tang, Y. Yehezkeally, M. Schwartz, and F. Farnoud, "Single-error detection and correction for duplication and substitution channels," *IEEE Trans. Inf. Theory*, vol. 66, no. 11, pp. 6908–6919, Nov. 2020.

[107] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (corresp.)," *IEEE Trans. Inf. Theory*, vol. 30, no. 5, pp. 766–769, Sep. 1984.

[108] J. Ullman, "On the capabilities of codes to correct synchronization errors," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 95–105, Jan. 1967.

[109] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," *Autom. Remote Control*, vol. 26, no. 2, pp. 286–290, 1965.

[110] R. Venkataramanan, S. Tatikonda, and K. Ramchandran, "Achievable rates for channels with deletions and insertions," *IEEE Trans. Inf. Theory*, vol. 59, no. 11, pp. 6990–7013, Nov. 2013.

[111] J. M. Walsh and S. Weber, "Capacity region of the permutation channel," in *Proc. IEEE 46th Annu. Allerton Conf. Commun., Control, Comput.*, 2008, pp. 646–652.

[112] S. Wang, Y. Tang, R. Gabrys, and F. Farnoud, "Permutation codes for correcting a burst of at most t deletions," in *Proc. IEEE 58th Annu. Allerton Conf. Commun., Control, Comput.*, 2022, pp. 1–6.

[113] S. Wang, V. K. Vu, and V. Y. Tan, "Codes for correcting t limited-magnitude sticky deletions," 2023, *arXiv:2302.02754*.

[114] H. Wei and M. Schwartz, "Improved coding over sets for DNA-based data storage," *IEEE Trans. Inf. Theory*, vol. 68, no. 1, pp. 118–129, Jan. 2022.

[115] N. Weinberger and N. Merhav, "The DNA storage channel: Capacity and error probability bounds," *IEEE Trans. Inf. Theory*, vol. 68, no. 9, pp. 5657–5700, Sep. 2022.

[116] Wikipedia, "Edit distance," [Online]. Available: https://en.wikipedia.org/wiki/Edit_distance

[117] P. C. Wong, K.-K. Wong, and H. Foote, "Organic data memory using the DNA approach," *Commun. ACM*, vol. 46, no. 1, pp. 95–98, 2003.

[118] S. Yazdi, R. Gabrys, and O. Milenkovic, "Portable and error-free DNA-based data storage," *Sci. Rep.*, vol. 7, no. 1, pp. 1–6, 2017.

[119] S. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Trans. Molecular, Biol. Multi-Scale Commun.*, vol. 1, no. 3, pp. 230–248, Sep. 2015.

[120] Y. Yehezkeally, D. Bar-Lev, S. Marcovich, and E. Yaakobi, "Generalized unique reconstruction from substrings," *IEEE Trans. Inf. Theory*, vol. 69, no. 9, pp. 5648–5659, Sep. 2023.

[121] Y. Yehezkeally and M. Schwartz, "Reconstruction codes for DNA sequences with uniform tandem-duplication errors," *IEEE Trans. Inf. Theory*, vol. 66, no. 5, pp. 2658–2668, May 2020.

[122] Y. Yehezkeally and M. Schwartz, "Uncertainty and reconstruction with list-decoding from uniform-tandem-duplication noise," *IEEE Trans. Inf. Theory*, vol. 67, no. 7, pp. 4276–4287, Jul. 2021.

[123] S. S. Yim, R. M. McBee, A. M. Song, Y. Huang, R. U. Sheth, and H. H. Wang, "Robust direct digital-to-biological data storage in living cells," *Nature Chem. Biol.*, vol. 17, no. 3, pp. 246–253, 2021.

[124] M. Zeraatpisheh, M. Esmaeili, and T. A. Gulliver, "Construction of tandem duplication correcting codes," *IET Commun.*, vol. 13, no. 15, pp. 2217–2225, 2019.

[125] M. Zeraatpisheh, M. Esmaeili, and T. A. Gulliver, "Construction of duplication correcting codes," *IEEE Access*, vol. 8, pp. 96150–96161, 2020.

Jin Sima received the B.Eng. and M.Sc. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2013 and 2016, respectively, and the Ph.D. degree in electrical engineering from California Institute of Technology Pasadena, CA, USA, in 2022.

He is a Postdoctoral Researcher with the Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign. His research interests include information and coding theory, machine learning, and theory of computation.

Dr. Sima is a recipient of the 2019 IEEE Jack Keil Wolf ISIT Student Paper Award, the 2020–2021 IEEE Communication Society Data Storage Best Paper Award, the 2022 Caltech Charles Wilts Prize for best doctoral thesis, and the 2023 Thomas M. Cover Dissertation Award.

Netanel Raviv (Senior Member, IEEE) received the B.Sc. degree in mathematics and computer science and the M.Sc. and Ph.D. degrees in computer science from the Technion–Israel Institute of Technology, Haifa, Israel, in 2010, 2013, and 2017, respectively.

He is an Assistant Professor with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO, USA. His research interests include applications of coding techniques to privacy, distributed computations, and machine learning.

Dr. Raviv was an awardee of the IBM Ph.D. fellowship for the academic year of 2015–2016, the first prize in the Feder family competition for best student work in communication technology in 2017, and the Lester-Deutsche Postdoctoral Fellowship.

Moshe Schwartz (Senior Member, IEEE) received the B.A. (*summa cum laude*) and M.Sc. and Ph.D. degrees in computer science from the Technion, Haifa, Israel, in 1997, 1998, and 2004 respectively.

He is a Professor with McMaster University. His research interests include algebraic coding, combinatorial structures, and digital sequences. He was a Fulbright Postdoctoral Researcher with UCSD and Caltech, and then a Professor at Ben-Gurion University of the Negev.

Dr. Schwartz received the 2009 IEEE Communications Society Best Paper Award in Signal Processing and Coding for Data Storage, and the 2020 NVMW Persistent Impact Prize. He has been serving as an Associate Editor and Area Editor for IEEE Transactions on Information Theory since 2014, and as an Editorial Board Member for the *Journal of Combinatorial Theory Series A* since 2021.

Jehoshua Bruck (Life Fellow, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion, Haifa, Israel, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1989.

He is the Gordon and Betty Moore Professor of computation and neural systems and electrical engineering with the California Institute of Technology, Pasadena, CA, USA. His current research interests include information theory and systems and the theory of computation in nature. His industrial experiences include working for IBM Research, cofounding and serving as the Chairman of: Rainfinity (acquired by EMC), XtremIO (acquired by EMC), and of MemVerge.

Dr. Bruck is a recipient of the Feynman Prize for Excellence in Teaching, the Sloan Research Fellowship, the National Science Foundation Young Investigator Award, the IBM Outstanding Innovation Award, and the IBM Outstanding Technical Achievement Award.