DNA Merge-Sort: A Family of Nested Varshamov-Tenengolts Reassembly Codes for Out-of-Order Media

Sajjad Nassirpour, Ilan Shomorony, *Member, IEEE*, and Alireza Vahid, *Senior Member, IEEE*

***Abstract*— Motivated by the DNA storage paradigm, we consider the torn-paper channel (TPC), which models data storage in long DNA molecules and breaks the input sequence into a random number of out-of-order variable-length non-overlapped fragments. We propose a computationally-efficient code construction for this model. More specifically, we introduce a family of nested Varshamov-Tenengolts (VT) codes to merge and sort the fragments in order to recover the stored data. We numerically show that our scheme (i) obtains rates that are higher than in prior results, (ii) has a decoding complexity that is cubic in the number of codeword fragments, which is significantly lower than the complexity of the brute-force approach, and (iii) offers decreasing and negligible error rates as the codeword length increases. We also propose a new construction for VT codes, quantify the number of required parity bits, and show that our approach requires fewer parity bits compared to known results.**

***Index Terms*— DNA storage, sequence assembly, unordered communication, codeword fragmentation, shuffling channel.**

I. Introduction

THE International Data Corporation estimates that by 2025 the global data storage demand will grow to 175 Zettabytes [1]. Due to this ever-growing demand for data storage, several alternative storage media have recently been proposed. In particular, Deoxyribonucleic Acid (DNA) has received significant attention as a viable medium for archival data storage. Several results [2], [3], [4], [5], [6], [7] have presented DNA-based storage prototypes, which as of now are capable of storing in the order of hundreds of Megabytes of data. The input binary data is encoded and mapped onto the four nucleotides in DNA (Adenine, Cytosine, Guanine, and Thymine), and the DNA molecules are then stored in a solution. DNA-based storage has the potential to provide information densities and information lifetimes far exceeding what is possible with state-of-the-art technologies [4], [6].

Currently, synthesizing long DNA molecules is challenging due to the state of the technology and the resulting higher error rates and costs [6], [8]. Hence, data is stored on short DNA molecules in a solution, where the DNA molecules are spatially out-of-order. This setup has been modeled mathematically as the *shuffling channel*, where data is synthesized on short DNA molecules (strings), and the output of the channel includes a multi-set of randomly shuffled strings [9], [10], [11], [12]. The decoder's goal is to recover the input data. The information-theoretic capacity of the shuffling channel is understood [13], and the capacity is achievable through indexing the short DNA molecules and using the remaining space for data bits. Additionally, several results have studied specific aspects of DNA storage caused by noise and imperfect writing/reading procedures, which lead to insertions, deletions, and substitutions of nucleotides in individual DNA molecules [9], [11], [14].

It is then a natural question to ask how the dynamics of the problem would change if synthesizing long DNA molecules became technologically and economically viable. To understand this problem, authors in [15] and [16] introduced the torn-paper channel (TPC) in which a length-n input data is stored on a long sequence that is then broken into a random number of out-of-order variable-length non-overlapped fragments, where the length of each fragment follows a Geometric(p_n) distribution. The breaking of the input sequence into fragments models either natural breaks of the physical DNA molecule or fragmentation that is induced by the sequencing library preparation (e.g., via sonication [17]). In [15] and [16], the authors derive the capacity of the TPC and compare it with the capacity of the shuffling channel. To make this comparison meaningful, the fixed-length strings in the shuffling channel are assumed to be $1/p_n$ in length. The capacity expressions are then given by [16]:

$$C_{\text{TPC}} = e^{-\alpha}, \quad C_{\text{shuffle}} = (1-\alpha)^{+}, \tag{1}$$

where $\alpha = \lim_{n\to\infty} p_n \log_2(n)$ and $(x)^{+} = \max(0, x)$. A comparison between the capacity expressions reveals an interesting result as $C_{\text{TPC}} > C_{\text{shuffle}}$, meaning that at least from a capacity point of view, storing data on long DNA molecules is advantageous. The capacity boost comes from the tail of

Manuscript received 26 April 2023; revised 17 September 2023; accepted 13 November 2023. Date of publication 21 November 2023; date of current version 19 March 2024. The work of Ilan Shomorony was supported in part by the National Science Foundation (NSF) under Grants CCF-2007597 and CCF-2046991. The associate editor coordinating the review of this article and approving it for publication was E. Rosnes. *(Corresponding author: Sajjad Nassirpour.)*

Sajjad Nassirpour is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182 USA (e-mail: snassirpour@sdsu.edu).

Ilan Shomorony is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: ilans@illinois.edu).

Alireza Vahid is with the Department of Electrical and Microelectronic Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA (e-mail: alireza.vahid@rit.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2023.3335409.

Digital Object Identifier 10.1109/TCOMM.2023.3335409

the geometric distribution, which ensures having the length of a part of the fragments significantly greater than the expected value of the length of each fragment (i.e., $1/p_n$). Recently, an adversarial TPC has been studied in [18], where an information string is adversarially broken by a specific TPC into random fragments such that the length of each fragment is bounded. The findings in [18] show that the capacity of this channel is determined by the upper-bound of p_n. Thereafter, [19] assumes partial overlap between read sub-strings in the adversarial TPC and extends the code reconstruction in two ways: First, for a single string, [19] establishes an upper bound on achievable code rates and develops an efficient construction with asymptotically optimal rates. Next, in the second case, it studies a scenario involving multiple strings reconstructed together. The authors in [19] also derive a lower bound on the length of the read sub-strings and present two constructions for multi-strand reconstruction codes. Further, motivated by nanopore sequencing, a TPC with lost pieces is studied in [20] wherein a fraction of the fragments do not appear at the output.

The main challenge in the TPC is reassembling the original string in a computationally-efficient manner in order to recover the input data. The capacity in [15] and [16] was achieved via random coding, which is computationally prohibitive. One natural idea is to include "signatures" within the codeword to enable simpler reassembly as suggested in [16]. More precisely, the authors in [16] propose interleaving a *de Bruijn* sequence within the codeword, which achieves a rate in the order of $e^{-4\alpha}$, which is lower than $e^{-\alpha}$ (the TPC capacity). We refer to the rate obtained in [16] through the interleaved-pilot scheme as the lower-bound achievable rates in the rest of this paper. Another thought is to use code constructions that could inherently reveal the position of a fragment (assuming it is long enough) without mixing other sequences. The authors in [21] present a more practical encoding/decoding mechanism based on an embedded structure of Varshamov-Tenengolts (VT) codes to recover data from the output of the TPC. However, the results of [21] are limited to $\alpha = 0$ (i.e., capacity is 1). In this paper, we aim to consider the full range of α and propose practical codes to encode/decode the input data bits in the TPC. We will further explain how the results can be extended to a 4-letter alphabet in the nucleotide domain in section VI. In general, code construction for the TPC, and more broadly, out-of-order media, remains an interesting coding challenge.

Our contribution includes the following. We focus on the entire range for α (i.e., $\alpha \geq 0$) and present a family of nested Varshamov-Tenengolts codes for reassembling out-of-order codeword fragments. In particular, we merge multiple individual VT codewords with unique residues into a new VT codeword and repeat this process to arrive at the final codeword to be stored in the DNA molecule. We show numerically that the rate obtained by our proposed method is in the order of $e^{-1.65\alpha}$, which is higher than the lower-bound rate in [16]. Further, we define decoding complexity as the number of permutations that the decoder uses to determine the original data sequence. We present two decoding strategies: (i) a greedy algorithm, which aims to reassemble the entire sequence, and (ii) an opportunistic algorithm, which first finds the exact locations of larger fragments with high confidence and then fills up the remainder using the greedy algorithm. The results show that our method significantly reduces the computational complexity compared to the brute-force approach, which checks all the possible permutations. In particular, the complexity of our approach grows in proportion to M^3, while the brute-force method's complexity approximately increases as M^M, with M being the number of fragments. Moreover, we show through the simulations that our encoding/decoding scheme provides negligible error rates as the codeword length increases. Finally, we present a new construction for individual VT codewords with linear complexity that needs fewer parity bits compared to prior results [22].

The rest of the paper is organized as follows. In Sections II and III, we formulate the problem and summarize VT codes, respectively. In Sections IV and V, we explain our proposed encoder and decoder, respectively. In Section VI, we explain how the results can be extended to a 4-letter alphabet. Then, we provide details on why our encoding/decoding scheme contains nested VT code and erasure code in Section VII. We show our simulation results in Section VIII, and Section IX concludes the paper.

II. Problem Setting

Our goal is to create practical codes for DNA data storage modeled as the TPC depicted in Fig. 1 in order to reliably communicate input message W chosen uniformly at random from $\{1, 2, \ldots, 2^{nR}\}$.

The TPC can be described as follows [16]. The channel first breaks the (binary) length-n input sequence, $\mathbf{x}$, into M random-length non-overlapped fragments (where $1 \leq M \leq n$ is random itself), and then shuffles these fragments resulting in an out-of-order multi-set of output fragments. We use $\mathbf{y}_j$, L_j, and $S(\mathbf{x}) = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M\}$ to denote the j^{th} output fragment, its corresponding length, and the multi-set of all fragments (i.e., channel output), respectively. We note that $M = 1$ implies that the input sequence has gone through the TPC perfectly (i.e., no error).

We assume that $\bar{L}_1, \bar{L}_2, \ldots$ is an independently and identically distributed (i.i.d.) (infinite) sequence of Geometric(p_n) random variables. Then M is defined as the largest integer such that $\sum_{j=1}^{M-1} \bar{L}_j \leq n-1$. Finally, we set $L_j = \bar{L}_j$ for $j = 1, \ldots, M-1$ and $L_M = n - \sum_{j=1}^{M-1} \bar{L}_j$. Notice that, due to the truncation, the resulting distribution of the fragment lengths is not technically Geometric (since it is bounded by n). However, for large n, they are very close to Geometric.

Finally, we assume the decoder is aware of the statistics of the channel, *i.e.*, knows p_n, and uses $S(\mathbf{x})$ to find an estimate $\hat{W}$ of the input message W. An error occurs if $\hat{W} \neq W$, and the probability of decoding error is equal to:

$$e_n = \Pr\left\{\hat{W} \neq W\right\}. \tag{2}$$

We say that rate R is achievable if there exist encoders and decoders such that $e_n \to 0$ as $n \to \infty$. Then, the capacity of the TPC is defined as the supremum of all achievable rates. However, proving $e_n \to 0$ is challenging due to the intricate nature of our code construction for the TPC.

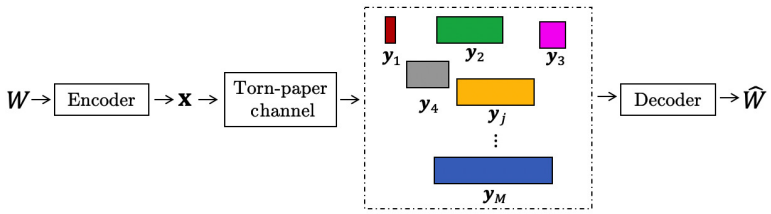


Fig. 1. The torn-paper channel breaks the input sequence, $\mathbf{x}$, into a random number of variable-length non-overlapped fragments.

In light of this challenge, we provide practical insight into the problem of reconstructing the out-of-order data fragments. To address this, we propose a family of practical nested VT codes to encode/decode the data bits for finite block lengths and use simulations to evaluate the performance. Our main goal is to create a code that reduces errors as much as possible, ideally approaching zero. In real-world scenarios, there is a trade-off between the computational complexity of encoding and decoding associated with the code and the error rate. Hence, in this paper, we aim to design our code with acceptable error rate and complexity. As a result, we call R, for which the error rate is evaluated through numerical analysis as a computed rate to distinguish it from the achievable rate defined above.

III. Background on Varshamov-Tenengolts Codes and a New Construction

In this part, we present a summary of Varshamov-Tenengolts (VT) codes, which were initially introduced for asymmetric Z-channels in [23] as given below. We then propose a new VT code construction that we will use in the rest of the paper.

Definition 1 [22]: For $0 \leq r \leq n$, the Varshamov-Tenengolts (VT) code, $VT_r(n)$, is a set of binary encoded strings with length n, which is given by:

$$VT_r(n) = \left\{ \boldsymbol{x} \in \{0,1\}^n : \sum_{i=1}^{n} i x_i \equiv r \mod (n+1) \right\}, \quad (3)$$

where x_i is the i^{th} element of $\boldsymbol{x}$, the sum is evaluated as an ordinary integer summation, and r is referred to as the residue.

In [24], it was shown that VT codes could correct single deletions in the data bits, and in fact, VT codes have been used in deletion channels extensively [25], [26], [27], [28], [29]. In [22], the author describes a VT encoder that generates codeword $\mathbf{x}$ from data bits $\mathbf{d}$ ($\mathbf{d}$ is a binary representation of input message W) with linear complexity. More precisely, [22] constructs VT codeword $\mathbf{x}$ as:

$$\mathbf{x} = [\mathbf{d} \quad \mathbf{p}], \quad (4)$$

where $\mathbf{d}$ is the data bits with length n_d, $\mathbf{p}$ is the parity bits with length n_p, and n_p equals to:

$$n_p = \left\lceil \sqrt{2n_d + \frac{9}{4}} + \frac{1}{2} \right\rceil, \quad (5)$$

where $\lceil . \rceil$ is the ceiling function.

1) New VT Construction: We introduce a new construction for an individual VT code as it will form the building block of our nested structure. The following lemma quantifies the number of parity bits in our VT code construction. We also provide a detailed description of how our VT encoder obtains parity bits $\mathbf{p}$ with linear complexity in Appendices B and C.

Lemma 1: Given any data-bit sequence of length n_d, it is possible to construct a VT codeword following (3) with length n and residue $0 \leq r \leq n$, where $n = n_d + n_p$ for $n_p = \left\lceil \sqrt{2n_d + \frac{1}{4}} + \frac{1}{2} \right\rceil$.

Proof: The proof is deferred to Appendix A. □

According to Lemma 1, our proposed VT construction requires fewer parity bits than (5).

IV. Proposed Nested VT Encoder

In this work, we merge and stack individual VT codes with unique residues in several layers to outline our ℓ-layer nested VT code. We use superscript $(l), 1 \leq l \leq \ell$ to denote the parameters in the l^{th} layer of the nested VT code.

We design our ℓ-layer encoder to map the input data bits, $\mathbf{d}$, with length $n_d = m^{\ell-1} d_{sec}$ into codeword $\mathbf{x}$ of length n, where d_{sec}, ℓ, and m are integer encoding parameters, which affect the computed rate, decoding complexity, and error rate. We derive upper and lower bounds on the rate associated with the nested VT code and describe how to select the encoding parameters in Section VII. Here, our proposed nested VT encoder applies the following steps to create $\mathbf{x}$:

A. Breaking Input Data Bits d Into $m^{\ell-1}$ sections

The proposed nested VT encoder considers $\mathbf{d}$ with length n_d as the input data bits. Our nested VT codeword includes ℓ layers, and in each layer, m VT codewords are merged to form a new data section in the following layer. Thus, we need $m^{\ell-1}$ data sections in the first layer. We assume that data sections in the first layer possess identical lengths denoted as d_{sec}, ensuring a consistent framework for further operations. As a result, we have $n_d = m^{\ell-1} d_{sec}$, and we break $\mathbf{d}$ into $m^{\ell-1}$ individual data sections to create the equal-length data sections;

B. Encode Each Data Section

In this step, we encode the i^{th} data section in the l^{th} layer, $\mathbf{d}_i^{(l)}, 1 \leq l \leq \ell, 1 \leq i \leq m^{\ell-l}$, to $\mathbf{x}_i^{(l)}$, the i^{th} VT codeword in the l^{th} layer with residue, $r_i^{(l)} = i - 1$, by following the VT codeword construction in (4) as:

$$\mathbf{x}_i^{(l)} = [\mathbf{d}_i^{(l)} \quad \mathbf{p}_i^{(l)}], \qquad 1 \leq l \leq \ell, 1 \leq i \leq m^{\ell-l}, \quad (6)$$

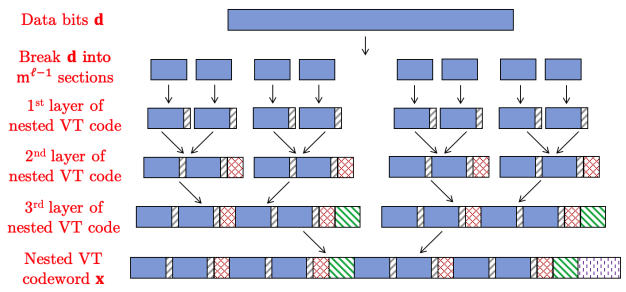


Fig. 2. Our proposed nested VT encoder with $\ell = 4$ and $m = 2$.

where $\mathbf{p}_i^{(l)}$ represents the parity bits of the i^{th} VT codeword in the l^{th} layer with length $\tilde{p}_l$ and $\tilde{p}_l = \left\lceil \sqrt{2\tilde{n}_{l-1} + \frac{1}{4}} + \frac{1}{2} \right\rceil$ using Lemma 1. We use $\tilde{n}_l$ to denote the length of $\mathbf{x}_i^{(l)}$, where $\tilde{n}_l = m \times \tilde{n}_{l-1} + \tilde{p}_l$ and $\tilde{n}_0 = d_{sec}$;

C. Merging the Codewords

In layer $l, 1 \leq l \leq \ell - 1$, we merge every m VT codewords to create a new data section for layer $l+1$. Specifically, we set:

$$\mathbf{d}_i^{(l+1)} = [\mathbf{x}_i^{(l)}, \mathbf{x}_{i+1}^{(l)}, \ldots, \mathbf{x}_{i+m-1}^{(l)}], \quad 1 \leq i \leq m^{\ell-l-1}. \tag{7}$$

This way, the number of data sections is reduced by a factor of m in each layer. Then, we use the same approach to encode the data sections in the new layer. Ultimately, the VT codeword of layer ℓ will be the output of the nested VT encoder (i.e., $\mathbf{x} = \mathbf{x}_1^{(\ell)}$). Fig. 2 depicts an example of the nested VT encoder with $\ell = 4$ and $m = 2$.

Notice that [21], [30] propose a quasi-systematic VT encoder for a single VT code by placing the parity bits at positions $2^i, i = 0, 1, \ldots, \log_2(n)$, which requires $\log_2(n) + 1$ parity bits. However, this method is limited to scenarios with data lengths of $2^k - k - 1$, where k is an integer and $k \geq 2$, while in this paper, we generate VT codes in different layers using variable-length data bits.

As we discussed above, $r_i^{(l)} = i - 1, 1 \leq l \leq \ell, 1 \leq i \leq m^{\ell-l}$, meaning that $\max_{l,i}\{r_i^{(l)}\} = r_{m^{\ell-1}}^{(1)} = m^{\ell-1} - 1$. On the other hand, according to Lemma 1, $0 \leq r_{m^{\ell-1}}^{(1)} \leq \tilde{n}_1$. As a result, we need the following condition to ensure the feasibility of encoding each data section with a unique residue:

$$\tilde{n}_1 \geq m^{\ell-1} - 1. \tag{8}$$

We use Algorithm 1 to describe how our proposed encoder works. First, we break $\mathbf{d}$ into $m^{\ell-1}$ sections. Next, in each encoding layer, we do the following tasks: (1) For all data sections, we encode $\mathbf{d}_i^{(l)}$ to VT codeword $\mathbf{x}_i^{(l)}$ based on residue $r_i^{(l)} = i - 1$; (2) If $l \neq \ell$, we merge every m VT codewords together to create a new data section for the next layer. Finally, we define the output of the last layer as the nested VT codeword.

We provide an example in Fig. 3 with $\ell = 2$, $m = 2$, and $d_{sec} = 7$ (i.e., $n_d = 14$) to illustrate our nested VT encoder. We first break $\mathbf{d}$ into two sections as $\mathbf{d}_1^{(1)}$ and $\mathbf{d}_2^{(1)}$. Then, we apply VT codes with $r_1^{(1)} = 0$ and $r_2^{(1)} = 1$ to $\mathbf{d}_1^{(1)}$ and $\mathbf{d}_2^{(1)}$, respectively, to obtain $\mathbf{x}_1^{(1)}$ and $\mathbf{x}_2^{(1)}$ in the first layer. Then, we merge $\mathbf{x}_1^{(1)}$ and $\mathbf{x}_2^{(1)}$ to construct $\mathbf{d}_1^{(2)}$ in the second layer. Finally, we apply a VT code with $r_1^{(2)} = 0$ to attain $\mathbf{x} = \mathbf{x}_1^{(2)}$ as the nested VT codeword.

Algorithm 1 Encoding Algorithm

Input: $\mathbf{d}$;
Output: $\mathbf{x}$;
1: Break $\mathbf{d}$ into $m^{\ell-1}$ sections;
2: **for** $l \in [1, \ell]$ **do**
3: **for** $i \in [1, m^{\ell-l}]$ **do**
4: Encode $\mathbf{d}_i^{(l)}$ to $\mathbf{x}_i^{(l)}$ using $r_i^{(l)} = i - 1$;
5: **if** $l \neq \ell$ **then**
6: Merge every m codewords to create a new data section for layer $l + 1$;
7: $\mathbf{x} = \mathbf{x}_1^{(\ell)}$ is the nested VT codeword.

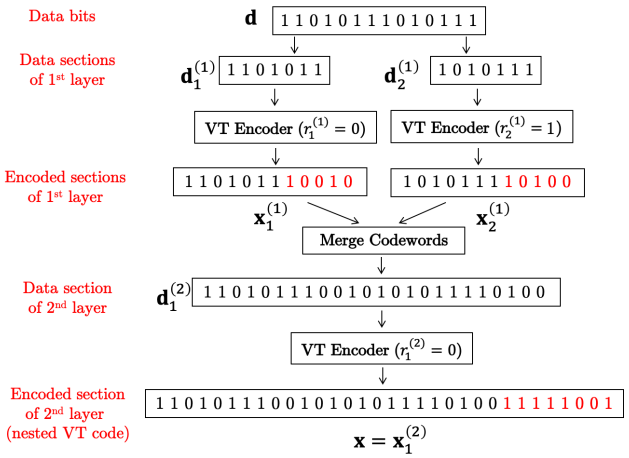


Fig. 3. An example of the nested VT encoder when $\ell = 2$, $m = 2$, and $d_{sec} = 7$.

V. Proposed Nested VT Decoder

We aim to merge and sort the out-of-order codeword fragments, create a combination of them that satisfies (a subset of) the VT conditions in different layers, and recover the input data. We name this combination $\hat{\mathbf{x}}$ as an estimated version of $\mathbf{x}$, and then remove the parity bits from $\hat{\mathbf{x}}$ to recover $\hat{\mathbf{d}}$.

We propose two decoding strategies in this work to obtain $\hat{\mathbf{x}}$: (i) a greedy algorithm that scans all fragments to merge and sort them and then reassemble the entire codeword. Based on Fig. 4, in each round of decoding, it seeks to add one fragment to the current sequences and keeps the sequences that satisfy a subset of the VT conditions in different layers; (ii) an opportunistic algorithm that defines the sufficiently large fragments (SLFs), determines (with high confidence) the exact location of each SLF, and uses the greedy algorithm to fill up the remainder, as shown in Fig. 4. Ultimately, after finding $\hat{\mathbf{x}}$, both decoding strategies remove the parity bits from $\hat{\mathbf{x}}$ to recover $\hat{\mathbf{d}}$. We will show later in Section VIII that the opportunistic algorithm provides lower complexity than the greedy algorithm.

A. Opportunistic Algorithm

To track the fragments during the decoding process, we allocate M labels as $\mathbf{y}_j, j = 1, 2, \ldots, M$ to the fragments. In this

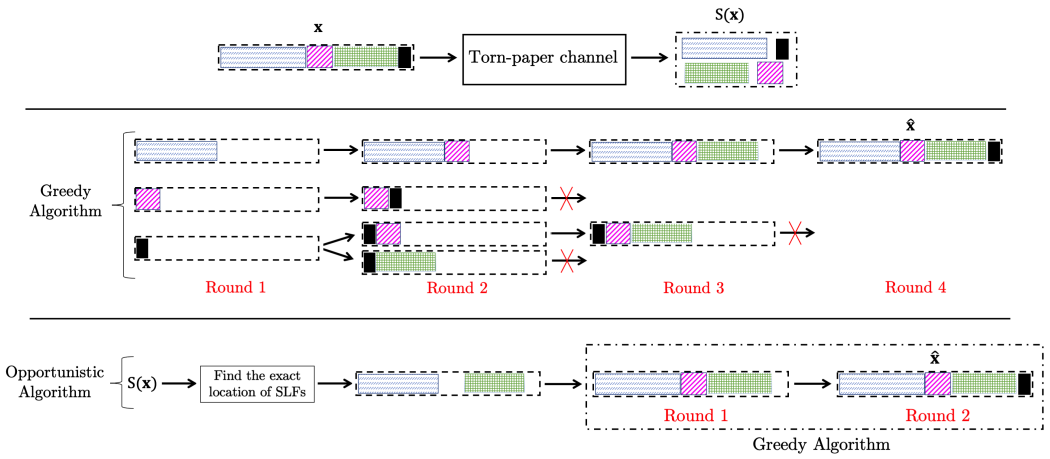


Fig. 4. Decoding strategies based on the proposed greedy and opportunistic algorithms.

part, we demonstrate the details of the opportunistic algorithm. To do so, we first define some useful parameters.

$N_{l,i}$: It denotes the position of the last bit in $\mathbf{x}_i^{(l)}, 1 \leq l \leq \ell, 1 \leq i \leq m^{\ell-l}$. Based on the nested VT code structure, up to $N_{l,i}$, there are i codewords in layer l with length $\tilde{n}_l$, and the parity bits of some possible codewords in the layers above l. Hence, $N_{l,i}$ can be determined as follows:

$$N_{l,i} = \begin{cases} i\tilde{n}_l + \sum_{l'=l+1}^{\ell-1} \left(\left\lceil \frac{i}{m^{l'-l}} \right\rceil - 1 \right)^+ \tilde{p}_{l'}, & 1 \leq l < \ell - 1, \\ i\tilde{n}_l, & \ell - 1 \leq l \leq \ell. \end{cases} \tag{9}$$

1) Sufficiently Large Fragments (SLFs): We divide $S(\mathbf{x})$ into two subsets: (i) $S_1(\mathbf{x})$, which contains the SLFs, and each SLF is required to be long enough to guarantee that at least m' codewords from layer 1 are fully inside the SLF, for some m' such that $2 \leq m' \leq m^{\ell-1}$. The SLF can be located either at the beginning/in the middle or at the end of the nested VT code. Therefore, the minimum length of an SLF can be expressed as:

$$\tilde{L}(m') = \max\left\{ \tilde{L}_1(m'), \tilde{L}_2(m') \right\}, \tag{10}$$

where $\tilde{L}_1(m')$ is the minimum length of an SLF, if it is available at the beginning/in the middle of the nested VT code. Here, we have:

$$\tilde{L}_1(m') = \max_{1 \leq j \leq m^{\ell-1}-m'} \{ N_{1,j+m'} - N_{1,j} + \tilde{n}_1 - 1 \}. \tag{11}$$

If the SLF is located at the end of nested VT code, its minimum length is equal to $\tilde{L}_2(m')$, which is defined as follows:

$$\tilde{L}_2(m') = \sum_{l'=2}^{\ell} \tilde{p}_{l'} + N_{1,m^{\ell-1}} - N_{1,m^{\ell-1}-m'+1} + \tilde{n}_1; \tag{12}$$

(ii) $S_2(\mathbf{x})$, which includes the non-SLFs having the length smaller than $\tilde{L}(m')$.

2) High-Level Overview of Finding the Exact Location of Each SLF: Our decoder wishes to obtain P_j^* as the exact location of $\mathbf{y}_j \in S_1(\mathbf{x})$. The main idea is to extract the starting points j' in the fragment, for which $\mathbf{y}_j$ meets all possible VT conditions after $y_{j,j'}$ with respect to L_j, where $y_{j,j'}$ is the j'^{th} bit of $\mathbf{y}_j$. We save these starting points, and if only one of them leads to the highest number of satisfied VT conditions, we call it j'^* and use that to determine P_j^*; else, we declare that the decoder fails to find the exact location of $\mathbf{y}_j$.

We explain how our decoder obtains P_j^* with high confidence in Algorithm 2. We provide further details below.

In Algorithm 2, we create $\bar{\mathbf{y}}$ with length $\tilde{n}_1$ starting from $y_{j,j'}, 1 \leq j' \leq L_j - m'\tilde{n}_1 + 1$. Next, we compute $\bar{r} - 1$ as the residue of $\bar{\mathbf{y}}$ and consider $\nu = 1$ as the number of satisfied VT conditions. Then, in line 5 to 12, we focus on the other possible VT codewords in $\mathbf{y}_j$ after $\bar{\mathbf{y}}$. Specifically, while $\bar{r} < m^{\ell-1} - m' + 2$, $\nu \geq 1$, and $j' + (N_{1,\bar{r}+1} - N_{1,\bar{r}-\nu+1}) + \tilde{n}_1 - 1 \leq L_j$, we do the following: (1) define $\bar{N} = (N_{1,\bar{r}+1} - N_{1,\bar{r}-\nu+1})$ as the location where the new VT codeword begins; (2) create a new $\bar{\mathbf{y}}$ from $y_{j,j'+\bar{N}}$; (3) if the residue of $\bar{\mathbf{y}}$ is equal to $\bar{r}$, we increase ν and $\bar{r}$ by one; else, we set $\nu = 0$, meaning that it cannot satisfy the VT condition of the new codeword, and this breaks the while loop. Next, if $\nu \geq m'$, we consider a new row of matrix $\mathbf{P}$ as $[j'\ \bar{r}\ \nu]$. Then, we define $\mathbf{P}^*$ as the rows of $\mathbf{P}$ with the highest value of ν. Finally, if $\mathbf{P}^*$ includes one row, we calculate $P_j^* = N_{1,\bar{r}^*-\nu^*+1} - \tilde{n}_1 + 2 - j'^*$ as the exact location of $\mathbf{y}_j$; otherwise, we set $P_j^* = -1$ and the algorithm fails to find the exact location of $\mathbf{y}_j$.

In Fig. 5, we show an example of how Algorithm 2 determines the exact location of each SLF. We assume a codeword with length 41, generated from a 2-layer nested VT encoder with $m = 4$ and $d_{sec} = 4$, which is broken into three fragments by the TPC. First, we assign random labels to each fragment. Next, we consider $m' = 2$, resulting in $\tilde{L}(2) = 25$ and consequently $\mathbf{y}_1$ is an SLF since $L_1 = 28 > 25$. Algorithm 2 then starts with $j' = 1$, defines $\bar{\mathbf{y}}$, and computes $\bar{r}$ and ν for $\bar{\mathbf{y}}$, which are equal to 5 and 1, respectively. Since $\bar{r} = 5$ violates the condition of the while loop in Algorithm 2 (line 5), $j' = 1$ cannot lead to the correct location. Therefore, the algorithm increases j' by one and repeats the process. This process continues until $j' \leq 13$. As Fig. 5 shows, $j' = 7$ is the only case that meets $\nu \geq m'$ and leads to $\bar{r} = 3$ and $\nu = 2$. As a result, we have $\mathbf{P}^* = [7\ 3\ 2]$, meaning that the exact location of $\mathbf{y}_1$ is equal to 3.

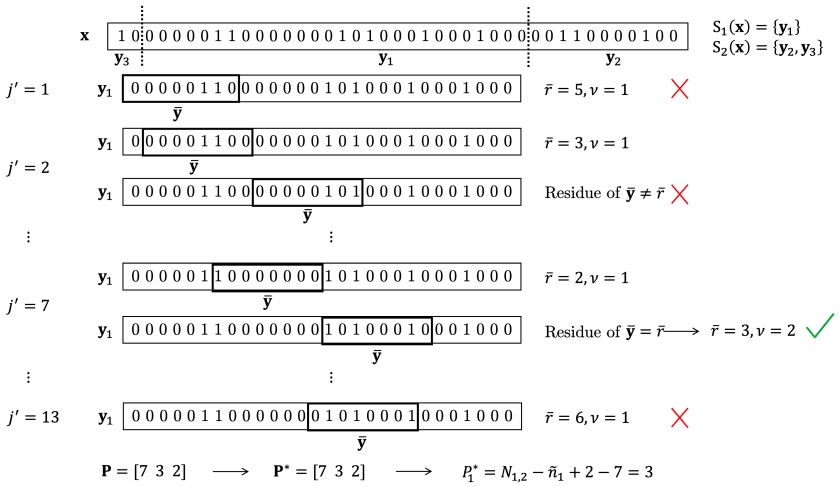


Fig. 5. An example of finding the exact location of an SLF in Algorithm 2 when $n = 41$, $M = 3$, $\ell = 2$, $m = 4$, and $d_{sec} = 4$.

Algorithm 2 Exact Location Algorithm for $\mathbf{y}_j$

Input: $\mathbf{y}_j$;
Output: P_j^*;
for $j' \in [1, L_j - m'\tilde{n}_1 + 1]$ **do**
  $\bar{\mathbf{y}} = [y_{j,j'}, y_{j,j'+1}, \ldots, y_{j,j'+\tilde{n}_1-1}]$;
  $\bar{r} - 1 =$ residue of $\bar{\mathbf{y}}$;
  $\nu = 1$;
  while $j' + (N_{1,\bar{r}+1} - N_{1,\bar{r}-\nu+1}) + \tilde{n}_1 - 1 \leq L_j$, $\bar{r} < m^{\ell-1} - m' + 2$, and $\nu \geq 1$, **do**
    $\bar{N} = (N_{1,\bar{r}+1} - N_{1,\bar{r}-\nu+1})$;
    $\bar{\mathbf{y}} = [y_{j,j'+\bar{N}}, \ldots, y_{j,j'+\bar{N}+\tilde{n}_1-1}]$;
    if residue of $\bar{\mathbf{y}} = \bar{r}$ **then**
      $\nu \leftarrow \nu + 1$;
      $\bar{r} \leftarrow \bar{r} + 1$;
    else
      $\nu = 0$;
  if $\nu \geq m'$ **then**
    New row of $\mathbf{P} = [j'\ \bar{r}\ \nu]$;
$\mathbf{P}^* =$ Rows of $\mathbf{P}$ with the highest value of ν;
if $\mathbf{P}^*$ is a vector (i.e., $\mathbf{P}^* = [j'^*\ \bar{r}^*\ \nu^*]$) **then**
  $P_j^* = N_{1,\bar{r}^*-\nu^*+1} - \tilde{n}_1 + 2 - j'^*$;
else
  $P_j^* = -1$ (i.e., The algorithm fails.)

3) Candidate Combination (CC): Suppose $\mathbf{\Gamma}$ is a sequence generated from the elements of $S(\mathbf{x})$, with the goal of constructing either a part or the entirety of $\hat{\mathbf{x}}$, starting from the beginning of $\hat{\mathbf{x}}$; specifically, $\mathbf{\Gamma}$ aims to be equal to the first γ bits of $\hat{\mathbf{x}}$, where γ denotes the length of $\mathbf{\Gamma}$ and is obtained as follows:

$$\gamma = \sum_{j' \in \mathcal{Y}_{\mathbf{\Gamma}}} L_{j'},\ \mathcal{Y}_{\mathbf{\Gamma}} = \left\{ j' \in \{1, 2, \ldots, M\} \,|\, \mathbf{y}_{j'} \in \mathbf{\Gamma} \right\}. \quad (13)$$

Then, we check the VT condition of every codeword $\mathbf{x}_i^{(l)}, 1 \leq l \leq \ell, 1 \leq i \leq m^{\ell-l}$ corresponding to $N_{l,i}$ if $N_{l,i} \leq \gamma$. Finally, we call $\mathbf{\Gamma}$ a CC if it satisfies VT conditions for all the above codewords and $\gamma < P_j^*, \forall j \in S_1(\mathbf{x}) - \mathbf{\Gamma}$.

We provide an example to make the definition of a CC clear. Consider $d_{sec} = 24$, $m = 2$, $\ell = 4$, and $\mathbf{\Gamma}$ is a sequence of some fragments in $S(\mathbf{x})$ with length $\gamma = 153$. Based on (9), we have $N_{1,1} = 32, N_{1,2} = 64, N_{1,3} = 108, N_{1,4} = 140, N_{2,1} = 76$, and $N_{2,2} = 152$, which are smaller than $\gamma = 153$. Therefore, if $\mathbf{\Gamma}$ meets VT conditions of $\mathbf{x}_1^{(1)}, \mathbf{x}_2^{(1)}, \mathbf{x}_3^{(1)}, \mathbf{x}_4^{(1)}, \mathbf{x}_1^{(2)}$, and $\mathbf{x}_2^{(2)}$, and $\gamma < P_j^*, \forall j \in S_1(\mathbf{x}) - \mathbf{\Gamma}$, it will be a CC.

Remark 1: We note that any sequence $\mathbf{\Gamma}$ that is generated from the elements of $S(\boldsymbol{x})$ with length γ is also a CC if $\gamma < P_j^*, \forall j \in S_1(\boldsymbol{x}) - \mathbf{\Gamma}$ and γ is less than $N_{1,1}$ (the position of the last bit in $\boldsymbol{x}_1^{(1)}$) since it includes no codeword and needs to wait for other fragments in future steps.

Remark 2: For consistency, we consider $\mathbf{\Gamma} = \phi$ (empty sequence) as a CC in the first round of decoding.

4) Helpful Fragment: We say $\mathbf{y}_j$ with length L_j is a helpful fragment if the combination of $\mathbf{y}_j$ and any CC creates a new CC.

5) Limited Memory: If the TPC breaks $\mathbf{x}$ into a large number of fragments, our decoder requires checking a vast number of possible cases, resulting in high computational complexity. To tackle this issue, we define a limited memory with parameter $\tau \in \mathbb{Z}^+$. After every τ iterations, our decoding algorithm keeps the CCs with the highest number of satisfied VT conditions. If the algorithm cannot resume with any of the kept CCs in the subsequent iterations, it increases τ by one and repeats the process. Our simulation results show that this method reduces the complexity of our approach significantly compared to a brute-force search, see Section VIII.

6) High-Level Overview of the Opportunistic Algorithm: We use Algorithm 2 to find the exact location of the SLFs. Then, it runs multiple rounds of a searching algorithm between all fragments to find $\hat{\mathbf{x}}$, which satisfies all VT conditions. In the first round, it searches through all fragments to get helpful fragment(s) and consequently CCs. Then, the decoder seeks to detect new helpful fragment(s) to merge with CCs to create new longer CCs. Finally, the decoder finds $\hat{\mathbf{x}}$ if it is a CC with length n. When our decoder obtains $\hat{\mathbf{x}}$, the rest of the decoding procedure will be simple.

In particular, the decoder removes the parity bits from $\hat{\mathbf{x}}$ to get $\hat{\mathbf{d}}$.

In Algorithm 3, we demonstrate how our proposed nested VT decoder performs the decoding procedure.

Algorithm 3 Decoding Algorithm

Input: $S(\mathbf{x})$, $S_1(\mathbf{x})$, $S_2(\mathbf{x})$;
Output: $\hat{\mathbf{d}}$;

1: $\tau = 1$;
2: **for** $\mathbf{y}_j \in S_1(\mathbf{x})$ **do**
3: Run Algo. 2 to calculate P_j^*;
4: **if** $P_j^* \neq -1$ **then**
5: Save P_j^*;
6: **else**
7: Transfer $\mathbf{y}_j$ from $S_1(\mathbf{x})$ to $S_2(\mathbf{x})$;
8: $K = 1$; $\mathbf{\Gamma}_1 = \phi$; $\beta_1 = S(\mathbf{x})$; $\tau' = 0$;
9: $Rep = 0$;
10: **for** $k \in [1, K]$ **do**
11: **for** $\mathbf{y}_j \in \beta_k$ **do**
12: **if** $\mathbf{y}_j$ is a helpful fragment **then**
13: $Rep \leftarrow Rep + 1$;
14: $\mathbf{\Gamma}_{K+Rep} = [\mathbf{\Gamma}_k \ \mathbf{y}_j]$ is a new CC;
15: $\beta_{K+Rep} = S(\mathbf{x}) - \mathbf{\Gamma}_{K+Rep}$;
16: **if** Length $\mathbf{\Gamma}_{K+Rep} = n$ **then**
17: $\hat{\mathbf{x}} = \mathbf{\Gamma}_{K+Rep}$;
18: Go to line 30;
19: Delete $\mathbf{\Gamma}_k$ and $\beta_k, 1 \leq k \leq K$;
20: $\tau' \leftarrow \tau' + 1$;
21: **if** $\tau' \geq \tau$ **then**
22: Save the CC(s) with the highest number of satisfied VT conditions;
23: $\tau' = 0$;
24: $K = Rep$;
25: **if** $K = 0$ **then**
26: $\tau \leftarrow \tau + 1$;
27: Go to line 8;
28: **else**
29: Go to line 9;
30: $l = \ell$
31: **while** $l > 1$ **do**
32: **for** $i \in [1, m^{\ell - l}]$ **do**
33: Omit parity bits from $\hat{\mathbf{x}}_i^{(l)}$ to get $\hat{\mathbf{d}}_i^{(l)}$;
34: $\hat{\mathbf{d}}_i^{(l)} = [\hat{\mathbf{x}}_i^{(l-1)}, \hat{\mathbf{x}}_{i+1}^{(l-1)}, \ldots, \hat{\mathbf{x}}_{i+m-1}^{(l-1)}]$;
35: $l \leftarrow l - 1$;
36: Decode $\hat{\mathbf{d}}_i^{(1)}$ from $\hat{\mathbf{x}}_i^{(1)}$ $\forall\ 1 \leq i \leq m^{\ell-1}$;
37: Merge $m^{\ell-1}$ data sections, as $\hat{\mathbf{d}}$.

We initialize the algorithm using $\tau = 1$. Next, for each SLF, we run Algorithm 2 to obtain its exact location. If the algorithm finds the exact location, we save its location; otherwise, we transfer that SLF from $S_1(\mathbf{x})$ to $S_2(\mathbf{x})$. Then, we set $K = 1$, $\mathbf{\Gamma}_1 = \phi$, $\beta_1 = S(\mathbf{x})$, and $\tau' = 0$, where K is the number of CCs, $\mathbf{\Gamma}_k, 1 \leq k \leq K$ illustrates the k^{th} CC, $\beta_k = S(\mathbf{x}) - \mathbf{\Gamma}_k$, and τ' is an index that follows the limited memory. Thereafter, we define Rep, initialized to zero, to denote the number of new CCs constructed by merging $\mathbf{\Gamma}_k$ and helpful fragments. Then, for each $\mathbf{\Gamma}_k$, we search between all $\mathbf{y}_j \in \beta_k$ to find the helpful fragments. For each $\mathbf{y}_j$, we merge $\mathbf{y}_j$ with $\mathbf{\Gamma}_k$ (i.e., $[\mathbf{\Gamma}_k \ \mathbf{y}_j]$) and check if $[\mathbf{\Gamma}_k \ \mathbf{y}_j]$ is a CC utilizing $S_1(\mathbf{x})$ and the exact locations of the SLFs. Then, if $[\mathbf{\Gamma}_k \ \mathbf{y}_j]$ is a CC, meaning that $\mathbf{y}_j$ is a helpful fragment, we increase Rep by one, add $\mathbf{\Gamma}_{K+Rep} = [\mathbf{\Gamma}_k \ \mathbf{y}_j]$ as a new CC, and consider $\beta_{K+Rep} = S(\mathbf{x}) - \mathbf{\Gamma}_{K+Rep}$. Next, we check the length of $\mathbf{\Gamma}_{K+Rep}$. If its length is equal to n, we say $\hat{\mathbf{x}} = \mathbf{\Gamma}_{K+Rep}$ and go to line 30. After checking all CCs, we remove the first K CCs and their corresponding β's because these K CCs have created new CCs with more elements in lines 10 to 18. Then, in line 20, we increase index τ' by one and check whether we have used the whole memory or not. If so, we keep only the CC(s), which provide the highest number of satisfied VT conditions and put $\tau' = 0$. Next, we use $K = Rep$ as the number of remained CCs. If $K = 0$, we realize that no CC met the VT conditions using limited memory τ. Hence, we set $\tau = \tau + 1$ and restart the decoding procedure. Otherwise, we go to line 9 and continue with the remained CCs.

We remove the parity bits of $\hat{\mathbf{x}}$ in lines 30 to 37. Specifically, we set $l = \ell$ and while $l > 1$, do the following: For the i^{th} codeword in the l^{th} layer, where $1 \leq i \leq m^{\ell - l}$, we remove the parity bits from the end of $\hat{\mathbf{x}}_i^{(l)}$ to decode $\hat{\mathbf{d}}_i^{(l)}$. In the next line, we break $\hat{\mathbf{d}}_i^{(l)}$ into m sections and consider each section as a codeword of layer $l-1$. Next, we decrease l by one and repeat lines 32 to 34. Then, we recover $m^{\ell-1}$ data sections of layer 1 by removing the parity bits from corresponding codewords in layer 1. Finally, we merge these $m^{\ell-1}$ data sections and obtain $\hat{\mathbf{d}}$.

In Fig. 6, we illustrate a decoding example based on the opportunistic algorithm, where $\mathbf{x}$ is the same as the nested VT codeword in Fig. 5, and the TPC breaks it into $M = 3$ different fragments. The decoder first obtains that $\mathbf{y}_1$ is an SLF and then applies Algorithm 2 to find the exact location of $\mathbf{y}_1$, which is $P_1^* = 3$. Thereafter, it follows Algorithm 3 to attain a CC with length $n = 41$, which satisfies all VT conditions. Specifically, in round 1, after initializing the decoding parameters, it scans all the fragments to determine the helpful fragments. As the figure shows, only $\mathbf{y}_3$ is a helpful fragment; thus, it set $Rep = 1, \mathbf{\Gamma}_2 = [\mathbf{y}_3]$, and $\beta_2 = \{\mathbf{y}_1, \mathbf{y}_2\}$. In the second round, the algorithm deletes $\mathbf{\Gamma}_1$ and β_1 and repeats the decoding procedure. This round reveals that $\mathbf{y}_1$ is a new helpful fragment, and thus we update the decoding parameters accordingly. Finally, in round 3, the opportunistic algorithm identifies $\hat{\mathbf{x}} = \mathbf{\Gamma}_2 = [\mathbf{y}_3 \ \mathbf{y}_1 \ \mathbf{y}_2]$ with length 41 meets all VT conditions, and hence declares it as the estimation of $\mathbf{x}$.

Remark 3: In the greedy algorithm, we set $S_1(\boldsymbol{x}) = \phi$ (i.e., $S_2(\boldsymbol{x}) = S(\boldsymbol{x})$), which results in skipping Algorithm 2, and then implement Algorithm 3 to decode the data bits.

VI. Extension to a 4-Letter Alphabet

As we mentioned earlier, the TPC is motivated by DNA-based data storage, where data is stored in the nucleotide domain characterized by a 4-letter alphabet (A, T, G, and C). In [21], the authors devise a low-complexity encoding/decoding scheme based on VT codes that suits the 4-letter alphabet in DNA-based data storage. In this section, we follow

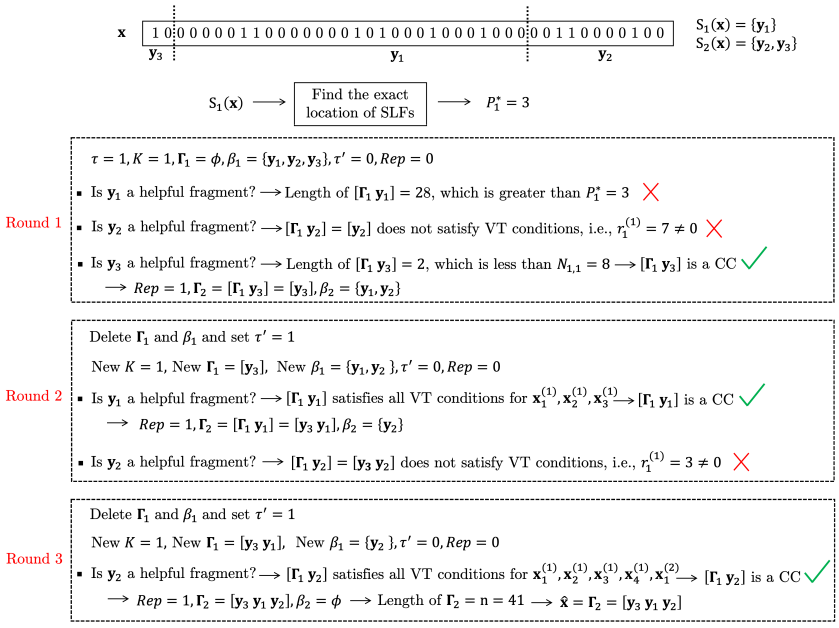


Fig. 6. An example of the opportunistic decoding algorithm when $n = 41$, $M = 3$, $\ell = 2$, $m = 4$, and $d_{sec} = 4$.

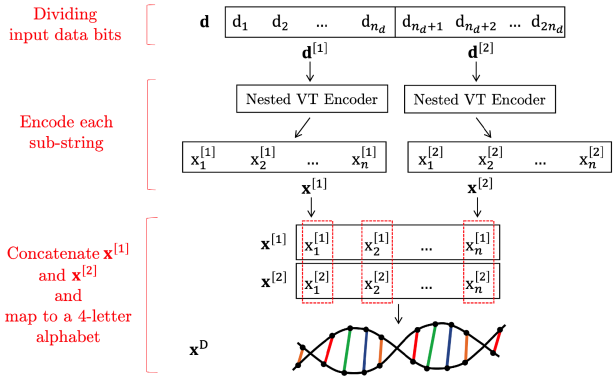


Fig. 7. Our proposed encoder for a 4-letter alphabet.

the methodology in [21] and explain how our proposed nested VT encoding/decoding scheme in the binary domain can be extended to a 4-letter alphabet.

A. 4-Letter Alphabet Encoder

We consider data bit string $\mathbf{d}$ in binary domain with length $2n_d$ as the input data bits. Then, we use our nested VT encoder, described in Section IV, to build the new encoder. Fig. 7 depicts our proposed encoder for a 4-letter alphabet. The process is as follows:

1) Dividing Input Data Bits: We divide data bit string $\mathbf{d}$ into two data bit sub-strings, $\mathbf{d}^{[1]}$ and $\mathbf{d}^{[2]}$, each containing n_d bits;

2) Encode Each Sub-String: We apply our nested VT encoder to encode $\mathbf{d}^{[1]}$ and $\mathbf{d}^{[2]}$ individually. We use $\mathbf{x}^{[1]}$ and $\mathbf{x}^{[2]}$ to denote the nested VT codeword sub-strings corresponding to $\mathbf{d}^{[1]}$ and $\mathbf{d}^{[2]}$, respectively;

3) Concatenate $\boldsymbol{x}^{[1]}$ and $\boldsymbol{x}^{[2]}$ and Map to a 4-Letter Alphabet: Finally, we place $\mathbf{x}^{[1]}$ on top $\mathbf{x}^{[2]}$ and map every two bits with the same location in different sub-strings, as shown in Fig. 7, into one letter exploiting the following mapping rule: $00 \rightarrow A, 01 \rightarrow T, 10 \rightarrow G$, and $11 \rightarrow C$. Here, $\mathbf{x}^{\mathrm{D}}$ represents the n-length codeword in a 4-letter alphabet.

B. 4-Letter Alphabet Decoder

Based on Fig. 8, the TPC randomly breaks codeword $\mathbf{x}^{\mathrm{D}}$ into M out-of-order fragments. To recover $\hat{\mathbf{x}}^{\mathrm{D}}$, the estimated version of $\mathbf{x}^{\mathrm{D}}$, we first assign M labels as $\mathbf{y}_j^{\mathrm{D}}, j = 1, 2, \ldots, M$ to the fragments and use L_j to denote the length of $\mathbf{y}_j^{\mathrm{D}}$, the j^{th} fragment. Next, we define $S(\mathbf{x}^{\mathrm{D}})$ as a multi-set that includes all fragments. Thereafter, we map all the fragments in $S(\mathbf{x}^{\mathrm{D}})$ back to the binary domain and use $\mathbf{y}_j^{[1]}$ and $\mathbf{y}_j^{[2]}$ as the binary equivalent of the codeword sub-strings in $\mathbf{y}_j^{\mathrm{D}}$. This mapping results in $S\left(\mathbf{x}^{[1]}, \mathbf{x}^{[2]}\right)$, which denotes a multi-set that contains the fragments in binary. Here, $\mathbf{y}_j^{[1]}$, $\mathbf{y}_j^{[2]}$, and $\mathbf{y}_j^{\mathrm{D}}$ have the same length, i.e., L_j. This is due to the fact that each letter in $\mathbf{y}_j^{\mathrm{D}}$ comprises two bits, one from $\mathbf{y}_j^{[1]}$ and the other from $\mathbf{y}_j^{[2]}$. Hence, the decoder considers $\{\mathbf{y}_j^{[1]}, \mathbf{y}_j^{[2]}\}$ as a pair and processes them such that, in a given solution, both $\mathbf{y}_j^{[1]}$ and $\mathbf{y}_j^{[2]}$ occupy the same position within different codeword sub-strings.

Similar to the nested VT decoder in the binary domain, we have two decoding strategies for a 4-letter alphabet as shown in Fig. 8: (i) greedy algorithm and (ii) opportunistic algorithm, where the former scans all fragments to merge and sort them and then reassemble the entire codeword, and the latter initially finds the exact location of each SLF with high confidence and uses the greedy algorithm to fill up the remainder. The ultimate aim of these decoding strategies is to find two reconstructed sequences $\hat{\mathbf{x}}^{[1]}$ and $\hat{\mathbf{x}}^{[2]}$ as the estimated version of nested VT codeword sub-strings $\mathbf{x}^{[1]}$ and $\mathbf{x}^{[2]}$, respectively, such that both sequences include the same order of binary fragments, e.g., with $M = 3$, we have $\hat{\mathbf{x}}^{[1]} = [\mathbf{y}_1^{[1]}, \mathbf{y}_3^{[1]}, \mathbf{y}_2^{[1]}]$ and $\hat{\mathbf{x}}^{[2]} = [\mathbf{y}_1^{[2]}, \mathbf{y}_3^{[2]}, \mathbf{y}_2^{[2]}]$. After obtaining $\hat{\mathbf{x}}^{[1]}$

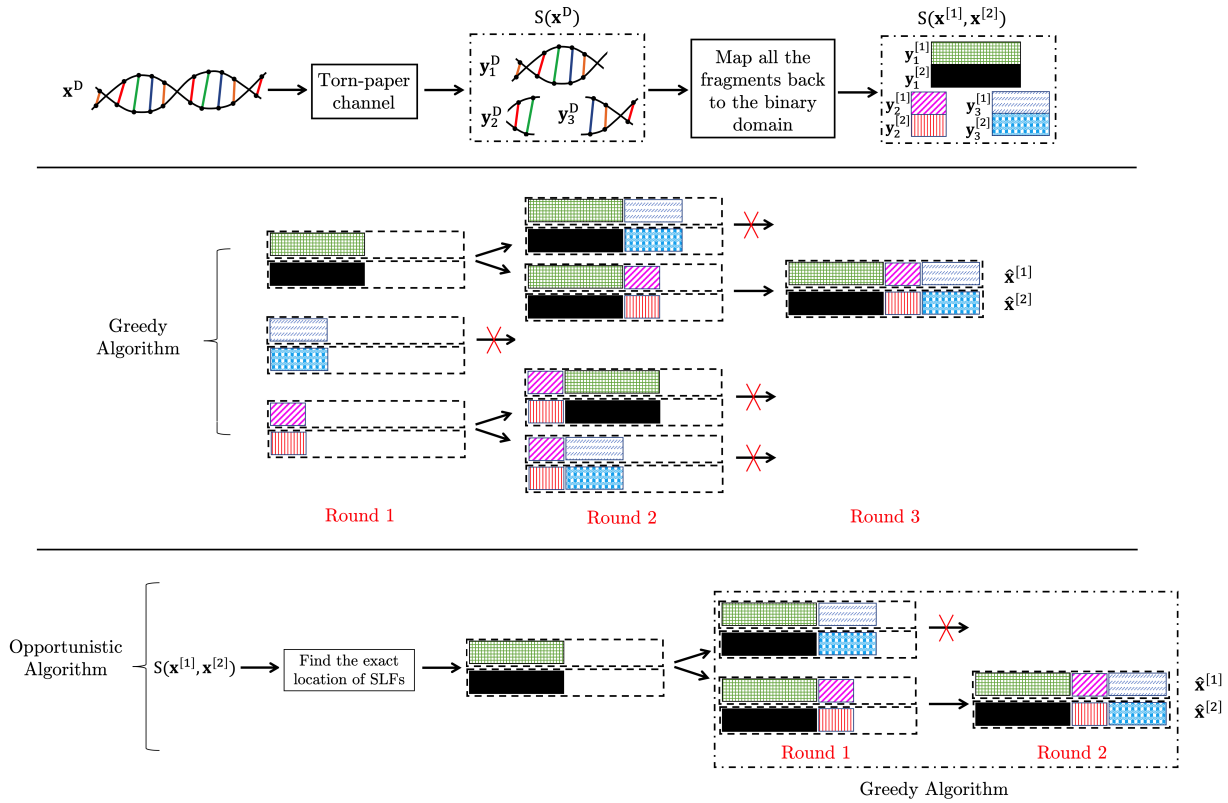


Fig. 8. Our proposed decoding strategies based on the greedy and opportunistic algorithms for a 4-letter alphabet when $M = 3$.

and $\hat{\mathbf{x}}^{[2]}$, we remove parity bits from each codeword sub-string individually to get $\hat{\mathbf{d}}^{[1]}$ and $\hat{\mathbf{d}}^{[2]}$. Consequently, we put $\hat{\mathbf{d}}^{[1]}$ and $\hat{\mathbf{d}}^{[2]}$ in a row to recover $\hat{\mathbf{d}}$ as the estimated version of $\mathbf{d}$.

The decoding parameters are mostly the same as what we explained in Section V, except for the definition of satisfying VT conditions. As we mentioned above, our decoder processes $\mathbf{y}_j^{[1]}$ and $\mathbf{y}_j^{[2]}$ together; therefore, we say VT conditions are satisfied if the VT conditions in both codeword sub-strings are satisfied simultaneously.

VII. Erasure Code and Nested VT Code

Ideally, we would like to recover $\mathbf{x}$ perfectly; however, for various reasons, we may not be able to obtain a unique sequence as the output of the decoder. A simple reason would be some of the fragments could fit at different positions, and this results in multiple reassembled sequences as the outputs of the nested VT decoder that satisfy all VT conditions. While this may seem like an error at first glance, we take the overlap of all the outputs as the final reassembled sequence, and our simulations show this if treated with care, will result in a correct reconstruction with high probability. Of course, taking the overlap results in some missing pieces, which can be viewed as erasures. For instance, in Fig. 9, the TPC breaks codeword $\mathbf{x}$ into $M = 7$ different fragments, in which two fragments contain only one bit each. By using our nested VT decoder, we obtain two strings Π_1 and Π_2, both of which meet all VT conditions, and removing the parity bits results in two distinct decoded data bits as $\hat{\mathbf{d}}_1$ and $\hat{\mathbf{d}}_2$. However, we take the overlap between $\hat{\mathbf{d}}_1$ and $\hat{\mathbf{d}}_2$ and conclude that $\hat{\mathbf{d}} = [1\ 1\ 0\ 1\ 0\ 1\ 1\ E\ E\ 1\ E\ 1\ E\ 1]$ is the recovered data bits where E denotes the erasure.

To handle these erasures, as Fig. 10 shows, we first apply an erasure code to the data bits before feeding the sequence to the nested VT encoder. We note that since our nested VT code preserves the data structure (i.e., it places parity bits at the end of each codeword), we can apply the erasure code before our proposed encoder, and data recovery requires an erasure decoder at the end. This way, the total rate of our scheme with erasure and nested VT codes is equal to:

$$R_t = (1-\epsilon)R, \tag{14}$$

where $(1-\epsilon)$ is the erasure code rate. We use the lower and upper bounds on the computed rate, as (15) (in the middle of the next page), and the derivation is given in Appendix D.

$$\frac{m^{\ell-1}d_{sec}}{\left(m^{\frac{\ell-1}{2}}\sqrt{d_{sec}}+\frac{\sqrt{2.5}}{2}\frac{m^{\frac{\ell}{2}}-1}{\sqrt{m}-1}\right)^2} < R < \frac{2m^{\ell-1}d_{sec}}{\left(m^{\frac{\ell-1}{2}}\sqrt{2d_{sec}}+\frac{m^{\frac{\ell}{2}}-1}{\sqrt{m}-1}\right)^2}. \tag{15}$$

A. Encoding Parameter Selection

In order to set the values of the various parameters in our proposed encoder, one goal is to have a computed rate below the capacity of the TPC, *i.e.*, $R_t \leq C_{\mathrm{TPC}}$. Moreover, to ensure each codeword has a unique residue, from (8), we need $\tilde{n}_1 \geq m^{\ell-1}-1$. Further, our simulation findings reveal how different

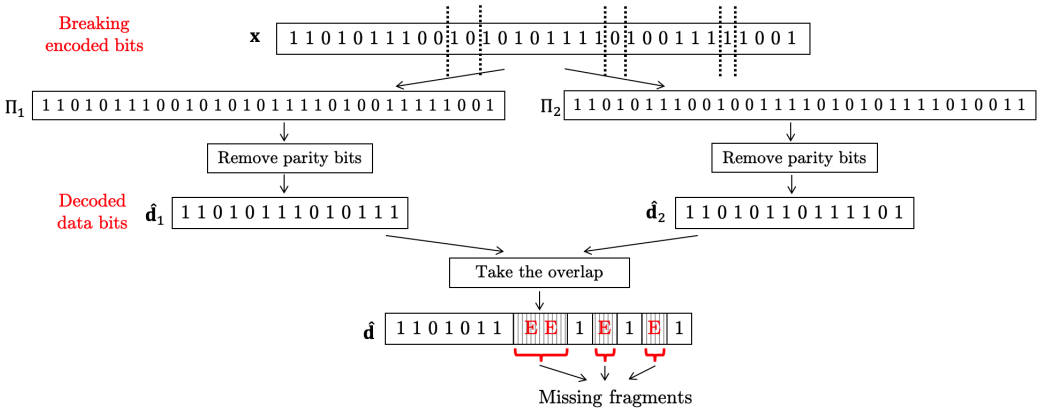


Fig. 9. An example of having more than one decoded data bits when $n = 32$, $M = 6$, $\ell = 2$, $m = 2$, and $d_{sec} = 7$.

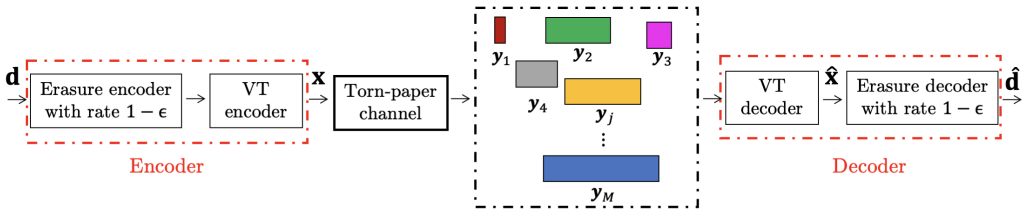


Fig. 10. Our proposed encoder/decoder scheme includes erasure and nested VT codes.

values of ℓ, m, and d_{sec} affect the computed rate, complexity, and error rate as shown in Table I when $n \approx 2^{10}$, $\alpha = 0.05$, and $\epsilon = 0.01$. Based on Table I, the error rate, which is attained via simulations, is minimized if d_{sec} is equal to the minimum possible value satisfying $\tilde{n}_1 \geq m^{\ell-1} - 1$. Also, the error rate and the decoding complexity are reduced by increasing m and decreasing d_{sec} when ℓ is constant. Similarly, if m is fixed, the error rate decreases by increasing ℓ and reducing d_{sec}. However, increasing m and ℓ negatively affects the computed rate, and raising ℓ reduces the computed rate much faster than increasing m. In addition, there is no specific trend for error rate and complexity when d_{sec} is fixed, and m and ℓ vary. As a result, we provide Algorithm 4 to determine the encoding parameters. Initially, we consider $\epsilon_0 = 0$ as the initial value of ϵ. Next, we use $\ell = 2$ and $m = 2$ and then set d_{sec} as its minimum possible value. Then, we compute R using ℓ, m, and d_{sec} and set $\epsilon = (1 - C_{\text{TPC}}/R)^+ + \epsilon_0$ to set $R_t \leq C_{\text{TPC}}$. Next, we calculate the error rate. If the error rate is acceptable, we declare the current values of ϵ, ℓ, m, and d_{sec} as the encoding parameters. Otherwise, if $m < m_{\max}$ where $m_{\max}$ is the maximum value of m, we increase m by one and go to line 3. If $m \geq m_{\max}$ and $\ell < \ell_{\max}$ where $\ell_{\max}$ is the maximum value of ℓ, we increase ℓ by one, set $m = 2$, and go to line 3. Finally, if raising ℓ and m does not result in an acceptable error rate, we set $\epsilon_0 \leftarrow \epsilon_0 + 0.1$ and go to line 2.

Fig. 11 compares the rate of our approach (i.e., erasure code + nested VT code) with nested VT code, the capacity of the TPC, and the lower-bound derived in [16] using de Bruijn sequences. We observe that our approach outperforms the interleaved-pilot scheme [16]. More precisely, our approach offers the computed rate equal to $0.86e^{-1.65\alpha}$ using the curve fitting toolbox in MATLAB, which is greater than the rate in the order of $e^{-4\alpha}$ in [16].

TABLE I

HOW DIFFERENT VALUES OF ℓ, m, AND d_{sec} CHANGE THE COMPUTED RATE, COMPLEXITY, AND ERROR RATE

ℓ	m	d_{sec}	R	Complexity	Error rate
2	2	409	0.8882	7650	0.029
3	2	185	0.8017	352	0.009
4	2	79	0.6818	42	0.0042
2	2	409	0.8882	7650	0.029
2	4	200	0.8630	512	0.0124
2	10	75	0.8126	38	0.0026
2	10	90	0.8280	84	0.007
3	3	90	0.7642	78	0.005
4	2	90	0.7038	88	0.004

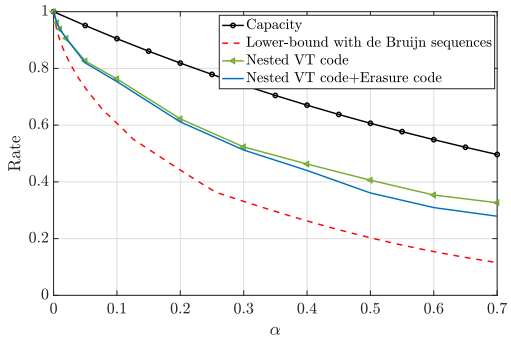


Fig. 11. A rate comparison between the nested VT code+erasure code, lower-bound of [16], nested VT code, and TPC capacity.

VIII. SIMULATION RESULTS

This section presents the simulation results of our work, which are done in MATLAB. We first compare the performance of our layered framework with the most related benchmark in [21]. Then, we analyze the error rate and complexity of our proposed encoding/decoding scheme. As our final step, we compare the complexity of our decoding strategies.

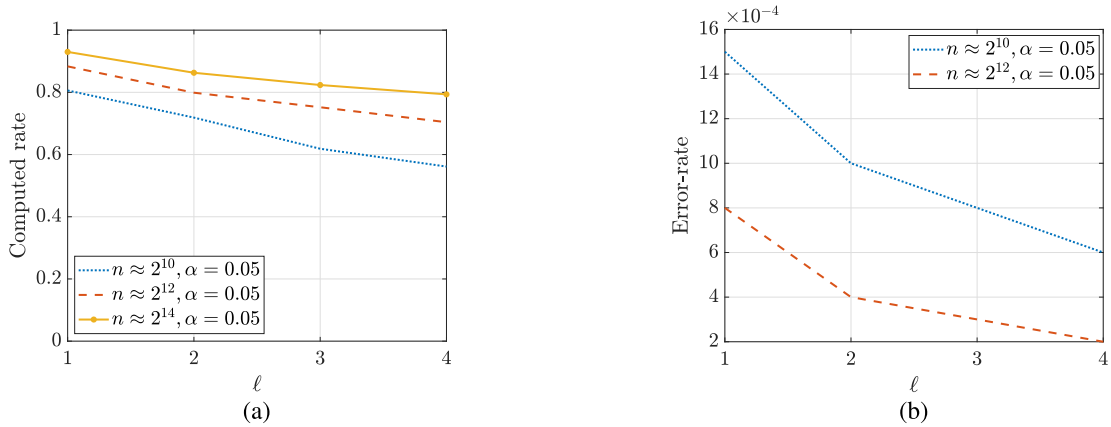


Fig. 12. A comparison between our nested VT code with $\ell \geq 2$ and the embedded VT code [21] with $\ell = 1$ in terms of (a) computed rate and (b) error rate when $n \in \{2^{10}, 2^{12}, 2^{14}\}$, $\alpha = 0.05$, and $\epsilon = 0.01$.

Algorithm 4 Encoding Parameter Selection Algorithm

Input: $\mathbf{d}, C_{\text{TPC}}, m_{\max}, \ell_{\max}$;
Output: $\ell, m, d_{sec}, \epsilon$;

1: Set $\epsilon_0 = 0$;
2: Set $\ell = 2$ and $m = 2$;
3: Set d_{sec} as its minimum possible value;
4: Compute R using ℓ, m, and d_{sec};
5: Consider $\epsilon = (1 - C_{\text{TPC}}/R)^{+} + \epsilon_0$;
6: Generate $\mathbf{x}$ using $\mathbf{d}$ and Algorithm 1;
7: Calculate the error rate;
8: **if** the error rate is acceptable **then**
9: $\ell, m, d_{sec}, \epsilon$ are the desired parameters;
10: **else**
11: **if** $m < m_{\max}$ **then**
12: $m \leftarrow m + 1$, go to line 3;
13: **else if** $m \geq m_{\max}$ and $\ell < \ell_{\max}$ **then**
14: $\ell \leftarrow \ell + 1, m = 2$, go to line 3;
15: **else**
16: $\epsilon_0 \leftarrow \epsilon_0 + 0.1$, go to line 2;

A. Simulation Setup

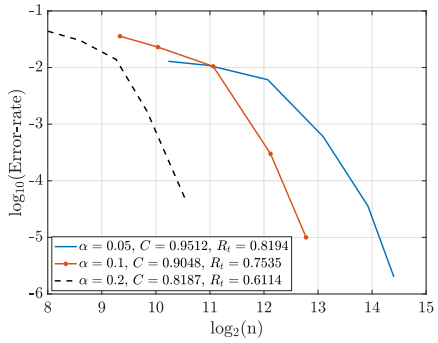
We assume input data bits $\mathbf{d}$ is a random binary string wherein each element is an i.i.d. Bernoulli$(1/2)$ random variable. Moreover, we consider the length of $\mathbf{d}$ to be $m^{\ell-1}d_{sec}$, and our nested encoder uses ℓ encoding layers to obtain codeword $\mathbf{x}$ with finite length n. Then, the TPC breaks codeword $\mathbf{x}$ into M out-of-order non-overlapped fragments where the length of each fragment is an independent random variable following a Geometric(p_n) distribution. Although this paper focuses on finite codeword lengths, we still restrict our attention to rates that are less than or equal to capacity C_{TPC} since those rates likely admit lower error probabilities. Therefore, in our numerical analysis, we need to know the value of C_{TPC}, which was derived analytically in [16] by letting $n \to \infty$. As a result, in order to characterize the capacity and consequently make an informed choice for an appropriate computed rate, we must let $n \to \infty$ in (1) (i.e., $C_{\text{TPC}} = e^{-\alpha}$) where $\alpha = \lim_{n\to\infty} p_n \log_2(n)$.

It is important to emphasize that p_n generally depends on the physical constraints of the TPC, and it can be obtained using general function $f(n)$. For the sake of simplicity, in this study, we assume $p_n = \alpha/\log_2(n)$, while it is worth noting that our findings can be extended to accommodate a broader range of functions for p_n beyond this specific assumption. Further, we use an erasure code with a rate of $1 - \epsilon$ before our nested VT encoder to indicate the role of missing pieces in our simulations.

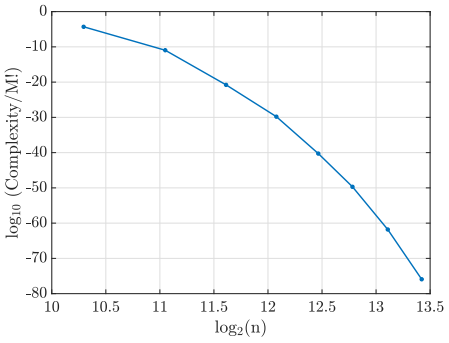
As we mentioned in Section VII, Algorithm 4 needs an acceptable error rate to determine the encoding parameters. It is worth noting that the acceptable error rate can vary depending on the target application. Given our motivation in durable data storage, such as DNA-based data storage and solid-state drives (SSDs), we adopt the acceptable error rate observed in SSDs, which falls within the range of 10^{-2} to 10^{-3} [31] as a benchmark. Furthermore, we determine the acceptable error rate such that the error rate decreases as the codeword length increases. To do this, we initially set the acceptable error rate in the order of 10^{-2} if n is small (i.e., $n < 2000$) and then determine the encoding parameters. Next, for $\bar{n}$ where $n < \bar{n} < 2000$, we set the acceptable error rate less than the error rate obtained using n. Thereafter, if the codeword length exceeds 2000, we set the initial value of the acceptable error rate in the order of 10^{-3} and repeat the same process.

B. Our Scheme vs. the Baseline in [21]

We compare our proposed encoding/decoding scheme with an embedded structure of VT codes [21] in terms of computed rate and error rate, where the former scheme uses $\ell \geq 2$ layers and the latter is a single-layer scheme with $\ell = 1$. Here, $n \in \{2^{10}, 2^{12}, 2^{14}\}$, $\alpha = 0.05$, and $\epsilon = 0.01$. Based on Fig. 12, the embedded structure provides a higher rate but suffers from a higher error rate, while our scheme gets a better error rate and a lower computed rate as ℓ increases due to using more VT codewords and thus more parity bits. Moreover, Fig. 12(a) depicts that the rate reduction with the number of layers disappears, and we get more flatted rate curves as n increases. Therefore, we increase the codeword length according to the number of layers to satisfy the desired error rate and have a negligible rate reduction simultaneously.



(a)



(b)

Fig. 13. (a) Error rates with different values of α; (b) The complexity of the opportunistic decoding algorithm when $\alpha = 0.05$, $\epsilon = 0.01$, $R_t = 0.8194$, $\ell = 3$, and $n \in (2^8, 2^{14})$.

C. Error Rate Analysis

Fig. 13(a) depicts the error rate of our solution as a function of the codeword length n on a logarithmic scale. Specifically, it includes three error rate curves when $(\alpha, C_{\text{TPC}}, \epsilon, R_t)$ are equal to $(0.05, 0.9512, 0.01, 0.8194)$, $(0.1, 0.9048, 0.01, 0.7535)$, and $(0.2, 0.8187, 0.02, 0.6114)$. As we can see in Fig. 13(a), the error rate decreases as n grows. This happens since when α is fixed and n increases, the average length of data fragments increases; thus, more VT codewords exist inside each fragment, which reduces the probability of having more than one sequence satisfying all VT conditions.

D. Complexity Analysis

In Fig. 13(b), we describe the complexity of our opportunistic decoding scheme in terms of the number of permutations that our approach explores during the decoding process, and we use the brute-force search as the baseline. The brute-force solution goes through $M!$ cases, which grows approximately (Stirling's approximation) as M^M. In Fig. 13(b), we consider $\alpha = 0.05$, $\epsilon = 0.01$, $R_t = 0.8194$, $\ell = 3$, and $n \in (2^8, 2^{14})$ and show the complexity of our approach divided by $M!$ versus the codeword length in a logarithmic fashion. A curve fitting of the results in Fig. 13(b) shows that the decoding complexity of our approach is proportional to M^3, which is significantly lower than the complexity of the brute-force method.

To obtain complexity and simulation runtime, we limit the decoding time to Δ. We say an error occurs if the decoder attains at least one $\hat{\mathbf{d}}$ during the given time, but $\hat{\mathbf{d}} \neq \mathbf{d}$. In Table II, we compare the number of cases that the decoding algorithm cannot recover any $\hat{\mathbf{d}}$ within Δ when $d_{sec} = 185$, $m = 3$, $\ell = 3$, $n = 2016$, $\alpha = 0.05$, $\epsilon = 0.01$, and $R_t = 0.8194$. The results show that these cases represent a small fraction of total cases and decrease as we increase Δ.

E. A Comparison Between the Proposed Decoding Algorithms

Finally, to compare the opportunistic and greedy algorithms, we consider a case when $\alpha = 0$, $\ell = 3$, and $n \approx 2^{13}$. Table III illustrates that the opportunistic algorithm reduces complexity, and its gain grows as m increases owing to having more SLFs.

TABLE II

NUMBER OF CASES WHERE OUR DECODER FAILS TO FIND ANY $\hat{\mathbf{d}}$ WITHIN Δ FOR: $d_{sec} = 185$, $m = 3$, $\ell = 3$, $n = 2016$, $\alpha = 0.05$, $\epsilon = 0.01$, AND $R_t = 0.8194$

Δ (sec)	Normalized # of failed cases
10	0.01
50	0.004
100	0.002
500	0.0008

TABLE III

COMPLEXITY RATIO BETWEEN THE OPPORTUNISTIC AND GREEDY ALGORITHMS WHEN $\alpha = 0$, $\ell = 3$, AND $n \approx 2^{13}$

d_{sec}	m	Complexity Ratio
294	5	0.81
200	6	0.57
143	7	0.44
107	8	0.29
83	9	0.25

According to Table III, this method is four times faster than the greedy algorithm when $d_{sec} = 83$ and $m = 9$.

IX. CONCLUSION

Motivated by recent advances in DNA-based storage, in this paper, we focused on the TPC and devised a practical encoding/decoding scheme based on a family of nested VT codes to retrieve data bits from the out-of-order fragments. We took into account practical considerations in real-world scenarios, such as the trade-off between the computational complexity of encoding and decoding associated with our nested VT code and the error rate. Subsequently, we obtained the computed rate in the TPC using simulations. We showed that our approach enables higher computed rates than the lower-bound in [16], provides two low-complexity decoding algorithms, which are significantly more efficient than the brute-force decoder, and the error rate reduces as the codeword length increases. Finally, we proposed a new method with linear complexity to build a single VT code that requires fewer parity bits compared to [22].

A. Future Work

This paper focused on a computed rate in the TPC due to the existing challenges in code reconstruction in the TPC. Hence, a natural extension for this work would be a rigorous analytical

proof of the achievable rate using nested VT codes in the TPC. Another future direction could involve reducing the length of parity bits in a single VT code. This could be accomplished by adapting the state-of-art approaches, e.g., quasi-systematic VT codes [21], [30] and shifted VT codes [32], thereby optimizing the coding scheme.

APPENDIX A
PROOF OF LEMMA 1

Proof: We will show later in Appendix B that δ is the residue of parity bits, which is given by:

$$\sum_{i'=1}^{n_p} i' x_{n+1-i'} \equiv \delta, \quad \mod (n+1), \tag{16}$$

where $0 \leq \delta \leq n$. We also know that $\frac{n_p(n_p+1)}{2}$ is the maximum value of $\sum_{i'=1}^{n_p} i' x_{n+1-i'}$, which happens if all parity bits are equal to 1. Hence, to find the binary representation of δ, we need:

$$\begin{aligned} &\frac{n_p(n_p+1)}{2} \geq n = n_d + n_p \\ &n_p^2 - n_p - 2n_d \geq 0 \rightarrow n_p \geq \sqrt{2n_d + \frac{1}{4}} + \frac{1}{2}. \end{aligned} \tag{17}$$

Since n_p is an integer number, we use $n_p = \left\lceil \sqrt{2n_d + \frac{1}{4}} + \frac{1}{2} \right\rceil$, and this completes the proof. □

APPENDIX B
FINDING PARITY BITS p

In this section, we demonstrate how to obtain parity bits **p** with length $n_p = \left\lceil \sqrt{2n_d + \frac{1}{4}} + \frac{1}{2} \right\rceil$ to ensure that codeword **x** meets desired residue $0 \leq r \leq n$. To do this, we define r' as:

$$\sum_{i=1}^{n_d} i d_i \equiv r' \mod (n+1), \tag{18}$$

where d_i is the i^{th} element of **d**. Moreover, we know that **d** represents the first n_d elements of **x**. Therefore, we have:

$$\begin{aligned} &\sum_{i=1}^{n} i x_i \\ &= \sum_{i=1}^{n_d} i x_i + \sum_{i=n_d+1}^{n} i x_i = \beta(n+1) + r' + \sum_{i=n_d+1}^{n} i x_i \\ &\overset{(i' \triangleq n+1-i)}{=} \beta(n+1) + r' + \sum_{i'=1}^{n-n_d} (n+1-i') x_{n+1-i'} \\ &= \underbrace{\beta(n+1)}_{a_1} + r' + \underbrace{(n+1)\sum_{i'=1}^{n_p} x_{n+1-i'}}_{a_2} - \sum_{i'=1}^{n_p} i' x_{n+1-i'}, \end{aligned} \tag{19}$$

where $\beta = \lfloor \frac{\sum_{i=1}^{n_d} i x_i}{n+1} \rfloor$ and $\lfloor . \rfloor$ is the floor function. We know that the residues of a_1 and a_2 in (19) are equal to zero. As a result, we have:

$$\sum_{i=1}^{n} i x_i \equiv r' - \delta, \quad \mod (n+1), \tag{20}$$

where δ is equal to:

$$\sum_{i'=1}^{n_p} i' x_{n+1-i'} \equiv \delta, \quad \mod (n+1). \tag{21}$$

Here, (21) shows that by calculating the residue of codeword **x** from the right-hand side (starting from the parity bits), δ represents the residue of parity bits string from the right-hand side. Therefore, to achieve desired residue r, all we need is to create parity bits **p** such that:

$$\delta = \begin{cases} r' - r, & r' \geq r, \\ (n+1) - (r - r'), & r' < r. \end{cases} \tag{22}$$

To derive parity bits **p**, we propose a three-step procedure as below:

Step 1: We define $k = n_p - i' + 1$ to show the position of the parity bit, which starts from the left side of parity bits. Then, we solve $n_p k - \frac{k(k-1)}{2} \leq \delta$ and obtain the largest non-negative integer value of k as k_{max}. Here, it is essential to show that $n_p k - \frac{k(k-1)}{2} \leq \delta$ is feasible for $n_p \geq 1$. To do this, we have:

$$\begin{aligned} n_p k - \frac{k(k-1)}{2} \leq \delta \overset{(a)}{\leq} \frac{n_p^2 + n_p}{2} \\ k^2 - (2n_p + 1)k + n_p^2 + n_p \geq 0, \end{aligned} \tag{23}$$

where (a) holds true since the maximum value of δ is equal to $(n_p^2 + n_p)/2$ when all bits of **p** are equal to 1. To solve the quadratic inequality in (23), we need to find k in the equality case. More precisely, we derive the roots of the following quadratic equality:

$$\begin{aligned} &k^2 - (2n_p + 1)k + n_p^2 + n_p = 0 \\ &k = \frac{2n_p + 1 \pm 1}{2} \\ &\rightarrow \begin{cases} k_1 = n_p, \\ k_2 = n_p + 1. \end{cases} \end{aligned} \tag{24}$$

Thus, based on k_1 and k_2 in (24), we know that either $k_{max} \leq k_1 = n_p$ or $k_{max} \geq k_2 = n_p + 1$ satisfies the inequality in (23). Moreover, we know $0 \leq k_{max} \leq n_p$ and k_{max} at most can be equal to n_p. As the result, it shows that always there is $0 \leq k_{max} \leq n_p$, which satisfies the quadratic inequality in (23), and this proves that $n_p k - \frac{k(k-1)}{2} \leq \delta$ is feasible for $n_p \geq 1$.

Step 2: In this step, we create an initial version of parity bits by placing one in position(s) $1 \leq k \leq k_{max}$. Thus, we have an initial parity bits such that $\sum_{i'=1}^{n_p} i' x_{n+1-i'} = n_p k_{max} - \frac{k_{max}(k_{max}-1)}{2}$;

Step 3: From Step 2, we have:

$$\sum_{i'=1}^{n_p} i' x_{n+1-i'} = n_p k_{max} - \frac{k_{max}(k_{max}-1)}{2} \leq \delta. \tag{25}$$

Therefore, to convert the inequality to equality in (25), we add $b = \delta - n_p k_{max} - \frac{k_{max}(k_{max}-1)}{2}$ to $\sum_{i'=1}^{n_p} i' x_{n+1-i'}$ in (25). To do this, we put 1 in the position of b.

APPENDIX C
COMPLEXITY OF FINDING PARITY BITS P

In this appendix, we describe the complexity of our strategy to find parity bits. To explain the details, we divide the complexity into the following three parts:

(i) $\sum_{i=1}^{n} ix_i$*:* Here, we need to perform $(n-1)$ additions to get $\sum_{i=1}^{n} ix_i$. We note that we do not need multiplications as $x_i \in \{0,1\}$, which simply means add or do not add i. Furthermore, each addition is a constant-time operation. Thus, $\mathcal{O}(n)$ is the complexity of $\sum_{i=1}^{n} ix_i$;

(ii) Mod Function: The author in [33], shows that the computing complexity of mod function of $r \equiv \sum_{i=1}^{n} ix_i \bmod(n+1)$ is $\mathcal{O}(\text{length}(n+1) \times \text{length}(q))$ where q satisfies:

$$\sum_{i=1}^{n} ix_i = q(n+1) + r, \tag{26}$$

and $\text{length}(z) = \lfloor \log_2(z) \rfloor + 1$ describes the length of number z in binary domain. We know that $q \leq \frac{n}{2}$ because $\sum_{i=1}^{n} ix_i \leq \frac{n(n+1)}{2}$. Hence, the computing complexity of mod function is $\mathcal{O}(\text{length}(n+1) \times \text{length}(\frac{n}{2})) = \mathcal{O}(\left\lfloor \log_2(n+1) \right\rfloor + 1\right] \times \left\lfloor \log_2 \frac{n}{2} \right\rfloor + 1\right])$;

(iii) Solving Quadratic Inequality: In [34], the authors explain that the complexity of finding the roots of a quadratic polynomial with ϵ-error is equal to $\mathcal{O}(\log_{10} \epsilon)$.

We note that since all calculations are done in the binary domain, we need to take into account the complexity of converting the root of the quadratic equation into the binary domain, which is given as:

$$\mathcal{O}(\lfloor \log_2 k_{max} \rfloor) \leq \mathcal{O}(\lfloor \log_2 n_p \rfloor), \tag{27}$$

since $k_{max} \leq n_p$. As a result, the overall complexity of finding the root of the quadratic equation equals $\mathcal{O}(\lfloor \log_2 n_p \rfloor + 1)$.

According to the above results, we conclude that the complexity of our encoding method is also linear (i.e., $\mathcal{O}(n)$), similar to Sloane's approach [22].

APPENDIX D
DERIVATION OF EQUATION (15)

The computed rate of our nested VT code is given by:

$$R = \frac{m^{\ell-1} d_{sec}}{n}. \tag{28}$$

To obtain the lower and upper bounds on R, we need to find the lower and upper bounds of n. Therefore, we divide the proof into two parts:

A. Lower Bound on n

According to Lemma 1, the length of each codeword in the first layer is equal to:

$$\tilde{n}_1 = d_{sec} + \left\lceil \sqrt{2d_{sec} + \frac{1}{4}} + \frac{1}{2} \right\rceil. \tag{29}$$

Here, we use $\tilde{n}_l$ to denote the length of each codeword in the $l^{\text{th}}, 1 \leq l \leq \ell$ layer. Next, we have:

$$\tilde{n}_1 \overset{(a)}{\geq} d_{sec} + \sqrt{2d_{sec} + \frac{1}{4}} + \frac{1}{2} > \frac{(\sqrt{2d_{sec}}+1)^2}{2} \overset{(b)}{=} \frac{(\tilde{d}_{sec}+1)^2}{2}, \tag{30}$$

where (a) is correct because $\lceil u \rceil \geq u$ for any real number u and (b) follows $\tilde{d}_{sec} \triangleq \sqrt{2d_{sec}}$. Moreover, each codeword in the second layer contains m different codewords of the first layer, which means:

$$\tilde{n}_2 = m\tilde{n}_1 + \left\lceil \sqrt{2m\tilde{n}_1 + \frac{1}{4}} + \frac{1}{2} \right\rceil. \tag{31}$$

Similar to (30), we have:

$$\tilde{n}_2 > \frac{(\sqrt{2m\tilde{n}_1}+1)^2}{2} = \frac{(\sqrt{m}(\tilde{d}_{sec}+1)+1)^2}{2}. \tag{32}$$

Then, following the same rule, we can show:

$$\begin{aligned}
\tilde{n}_\ell &> \frac{(\overbrace{\sqrt{m}(\sqrt{m}\ldots\sqrt{m}}^{\ell-1 \text{ coefficients}}(\tilde{d}_{sec}+1)\overbrace{+1)+1)\ldots)+1)}^{\ell-1 \text{ coefficients}})^2}{2} \\
&= \frac{(m^{\frac{\ell-1}{2}}\tilde{d}_{sec} + m^{\frac{\ell-1}{2}} + m^{\frac{\ell-2}{2}} + \ldots + \sqrt{m} + 1)^2}{2} \\
&= \frac{1}{2}(m^{\frac{\ell-1}{2}}\tilde{d}_{sec} + \frac{m^{\frac{\ell}{2}}-1}{\sqrt{m}-1})^2 \\
&= \frac{1}{2}(m^{\frac{\ell-1}{2}}\sqrt{2d_{sec}} + \frac{m^{\frac{\ell}{2}}-1}{\sqrt{m}-1})^2.
\end{aligned} \tag{33}$$

Since the nested VT code in layer ℓ includes only one section, (33) is the lower bound on n.

B. Upper Bound on n

To derive an upper bound on n, we have:

$$\begin{aligned}
\tilde{n}_1 &\overset{(c)}{<} (d_{sec}+1) + \sqrt{2d_{sec} + \frac{1}{4}} + \frac{1}{2} \\
&< (d_{sec}+1) + 0.5 + \sqrt{2(d_{sec}+1)} \\
&= (\sqrt{d_{sec}+1} + \frac{\sqrt{2}}{2})^2 \overset{(d)}{<} (\sqrt{d_{sec}} + \frac{\sqrt{2.5}}{2})^2,
\end{aligned} \tag{34}$$

where (c) holds true since $\lceil u \rceil < u+1$ for any real number u. Then, we simplify (34) and obtain $d_{sec} > 35.288$, which means that the condition in (34) holds true if $d_{sec} \geq 36$.

Then, following the same steps in deriving the lower bound of n leads to:

$$\tilde{n}_\ell < (m^{\frac{\ell-1}{2}}\sqrt{d_{sec}} + \frac{\sqrt{2.5}}{2}\frac{m^{\frac{\ell}{2}}-1}{\sqrt{m}-1})^2. \tag{35}$$

Finally, we use (28), (33), and (35) to derive equation (15).

REFERENCES

[1] D. R.-J. G.-J. Rydning, J. Reinsel, and J. Gantz, "The digitization of the world from edge to core," *Framingham, Int. Data Corp.*, vol. 16, pp. 1–28, Nov. 2018.

[2] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1612, Sep. 2012.

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

[4] 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," *Angew. Chem. Int. Ed.*, vol. 54, no. 8, pp. 2552–2555, Feb. 2015.

[5] S. M. H. Tabatabaei 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, Sep. 2015.
[6] Y. Erlich and D. Zielinski, "DNA fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, pp. 950–954, Mar. 2017.
[7] L. Organick et al., "Random access in large-scale DNA data storage," *Nature Biotechnol.*, vol. 36, no. 3, pp. 242–248, Mar. 2018.
[8] 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.*, Mar. 2016, pp. 637–649.
[9] 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.
[10] I. Shomorony and R. Heckel, "Capacity results for the noisy shuffling channel," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 762–766.
[11] 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, Apr. 2020.
[12] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakohi, "Achieving the capacity of the DNA storage channel," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 8846–8850.
[13] R. Heckel, I. Shomorony, K. Ramchandran, and D. N. C. Tse, "Fundamental limits of DNA storage systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 3130–3134.
[14] 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, Jul. 2019.
[15] I. Shomorony and A. Vahid, "Communicating over the torn-paper channel," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.
[16] I. Shomorony and A. Vahid, "Torn-paper coding," *IEEE Trans. Inf. Theory*, vol. 67, no. 12, pp. 7904–7913, Dec. 2021.
[17] J. Sambrook and D. W. Russell, "Fragmentation of DNA by sonication: Figure 1," *Cold Spring Harbor Protocols*, vol. 2006, no. 4, Sep. 2006, Art. no. pdb.prot4538.
[18] D. Bar-Lev, S. Marcovich, E. Yaakobi, and Y. Yehezkeally, "Adversarial torn-paper codes," 2022, *arXiv:2201.11150*.
[19] 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.
[20] A. N. Ravi, A. Vahid, and I. Shomorony, "Capacity of the torn paper channel with lost pieces," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2021, pp. 1937–1942.
[21] S. Nassirpour and A. Vahid, "Embedded codes for reassembling non-overlapping random DNA fragments," *IEEE Trans. Mol., Biol. Multi-Scale Commun.*, vol. 7, no. 1, pp. 40–50, Mar. 2021.
[22] N. J. A. Sloane, "On single-deletion-correcting codes," in *Codes Designs*. Columbus, OH, USA: Ohio State Univ., 2000, pp. 273–291.
[23] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," (in Russian), *Automatika Telemkhanika*, vol. 161, no. 3, pp. 288–292, 1965.
[24] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Sov. Phys.-Dokl.*, vol. 10, no. 8, pp. 707–710, 1966.
[25] A. S. J. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 1, pp. 305–308, Jan. 2002.
[26] 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," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1804–1808, Mar. 2012.
[27] M. Abroshan, R. Venkataramanan, L. Dolecek, and A. G. I. Fàbregas, "Coding for deletion channels with multiple traces," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 1372–1376.
[28] G. Mappouras, A. Vahid, R. Calderbank, and D. J. Sorin, "GreenFlag: Protecting 3D-racetrack memory from shift errors," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 1–12.
[29] A. Vahid, G. Mappouras, D. J. Sorin, and R. Calderbank, "Correcting two deletions and insertions in racetrack memory," 2017, *arXiv:1701.06478*.
[30] K. Saowapa, H. Kaneko, and E. Fujiwara, "Systematic deletion/insertion error correcting codes with random error correction capability," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (EFT)*, Nov. 1999, pp. 284–292.
[31] B. S. Kim, J. Choi, and S. L. Min, "Design tradeoffs for SSD reliability," in *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*, 2019, pp. 281–294.
[32] 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.
[33] V. Shoup, *A Computational Introduction to Number Theory and Algebra*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[34] M.-H. Kim and S. Sutherland, "Polynomial root-finding algorithms and branched covers," *SIAM J. Comput.*, vol. 23, no. 2, pp. 415–436, Apr. 1994.

Sajjad Nassirpour received the B.Sc. degree in electrical engineering from Shiraz University, Shiraz, Iran, in 2011, the M.Sc. degree in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2014, and the Ph.D. degree in electrical engineering from the University of Colorado Denver, CO, USA, in 2023. He is currently a Post-Doctoral Fellow with San Diego State University, CA, USA. His current research interests include wireless communications, machine learning, and applications of coding theory in high-performance computer memory systems. He received the Best Paper Award at the IEEE Computing and Communication Workshop and Conference (CCWC) in 2020.

Ilan Shomorony (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Cornell University in 2014. He was a Post-Doctoral Scholar with UC Berkeley through the NSF Center for Science of Information (CSoI) until 2017. After that, he spent a year working as a Researcher and a Data Scientist with Human Longevity Inc., a personal genomics company. He is currently an Assistant Professor of electrical and computer engineering with the University of Illinois at Urbana–Champaign (UIUC), where he is also a member of the Coordinated Science Laboratory. His current research interests include information theory, communications, and computational biology. He received the NSF CAREER Award in 2021.

Alireza Vahid (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2009, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2012 and 2015, respectively. From 2015 to 2017, he was a Post-Doctoral Research Scientist with the Information Initiative at Duke University, Durham, NC, USA. From 2017 to 2023, he was an Assistant Professor of electrical engineering with CU Denver. He is currently a Gleason Endowed Associate Professor of electrical and microelectronic engineering with the Rochester Institute of Technology. His current research interests include network information theory, wireless communications, and data storage. He received the 2015 Outstanding Ph.D. Thesis Research Award from Cornell University, the Qualcomm Innovation Fellowship in 2013, the Laboratory Venture Challenge Award in 2019, and the SONY Faculty Innovation Award in 2021. He currently serves as an Associate Editor for IEEE COMMUNICATIONS LETTERS.