Channel Model and Decoder With Memory for DNA Data Storage With Nanopore Sequencing

Belaid Hamoum, Elsa Dupraz

Channel Model and Decoder with Memory for DNA Data Storage with Nanopore Sequencing

Belaid Hamoum[1] and Elsa Dupraz[2]

[1] Lab-STICC, CNRS UMR 6285, Université Bretagne-Sud, Lorient, France

[2] IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238, France

Abstract

This paper investigates channel coding for DNA data storage, a recent and emerging paradigm in which both substitution errors and synchronization errors (insertions, deletions) are introduced in the stored data. The paper first proposes a novel and accurate statistical channel model to represent errors introduced by the DNA data storage support. The channel model takes the form of a k-order Markov Model which allows to capture both memory in the sequence of channel errors, and statistical dependency between errors and input sequence. The model probabilities are inferred on two sets of data: one set of experimental data, and one set of genomic data. In both cases, we observe a superior accuracy, evaluated from the Kulback-Leibler divergence, of our channel model compared to existing ones. Second, the paper investigates the design of error-correction solutions dedicated to the proposed channel model. It considers a concatenated code construction built from a convolutional decoder which aims to correct synchronization errors, and from an outer LDPC code which corrects residual substitution errors. While this construction was already investigated for i.i.d. errors, this paper shows how to include all the knowledge of the proposed channel model into the convolutional decoder. Numerical results show that the proposed decoding algorithm significantly improves the decoding performance in terms of BER and FER, at the price of an increased decoding complexity.

Index Terms

DNA storage channel model, Coding for DNA data storage, Concatenated Codes, Convolutional Decoders, BCJR algorithm

I. Introduction

DNA data storage is seen as an emerging and promising alternative to conventional storage supports, due to higher density and greater durability. In DNA data storage, information is stored

by synthesizing a digital sequence into a DNA molecule built from a sequence of bases A,C,G,T. The original information can be retrieved by using a DNA sequencer such as the Oxford Nanopore Technology (ONT) MinION sequencer which we consider in this work. Recent progress in both the synthesis and sequencing technologies pave the way for the development of cost-effective and competitive DNA data storage solutions in the next few years.

Unfortunately, sequencing still introduces a large amount of substitution, insertion, and deletion errors in the read data. Insertions and deletions break sequence synchronization, and they cannot be corrected with conventional error-correction methods designed for data storage or wireless communications. Therefore, from a channel coding perspective, two key challenges should be addressed in order to make DNA data storage more practical: (i) develop accurate statistical channel models to represent errors that occur during the DNA data storage process, (ii) design powerful error-correction solutions that can handle the three types of errors. This paper aims to address both of these challenges in a cohesive manner, by first introducing an accurate channel model and then incorporating the model knowledge into the decoder.

Statistical channel models allow to run extensive in-silico simulations, before implementing expensive in-vitro experiments. Regarding DNA data storage, a wide range of works consider non-uniform independent and identically distributed (i.i.d.) error models [1]–[4]. But it is shown in [5] that the error statistics are highly dependent on the input sequence, and contain memory, making the i.i.d. assumption unrealistic. To account for these dependencies, a first set of simulators have employed Deep-Learning (DL) based approaches, such as DeepSimulator [6], [7], which utilizes Deep Neural Networks (DNNs), or the approach of [8] which uses Generative Adversarial Networks (GANs). However, these simulators are trained for specific sequence lengths, and the training process is computationally expensive. Furthermore, DL-based simulators are not completely appropriate in our context, since the channel probabilities are not explicit and cannot be incorporated into the design of the decoder. Alternatively, PBSIM2 [9] considers transition probabilities over a sequence of quality scores, and samples errors from the quality scores according to a fixed distribution between error events. But we observed on experimental data that the event distribution is not fixed and varies with the input sequence. In addition, BadRead [10] uses transition probabilities estimated from experimental data to randomly substitute length-k sequences with other length-k' sequences. However, in Badread, the user has to specify many parameters such as the average reads-length or the total amount of errors, and the channel

accuracy heavily depends on the considered parameters.

This paper introduces a new and accurate statistical channel model that is defined by a set of probabilities over the error events and does not require any additional parameter. Given that the MinION sequencer reads k symbols at a time, our channel model relies on a k-order Markov Model in order to take into account both memory in the successive error events, and statistical dependency between errors and input sequence. We describe how to train the model using two different datasets: one smaller dataset of experimental data passed through the entire DNA storage process, and one larger dataset of genomic data. Numerical evaluation of the Kullback-Leibler divergence demonstrates that our model more accurately represents the actual DNA data storage process than existing models [6], [7], [10]. Lastly, we introduce a dynamic channel model, which allows to smoothly adjust the channel probabilities inferred from the datasets, in order to simulate a wider range of error probabilities in numerical experiments. It is worth mentioning that in our previous work [11] where the channel model was introduced, only one dataset was used for training, and the dynamic channel model was not presented.

Second, this paper addresses the design of efficient error-correction solutions for DNA data storage. The first error-correction solutions proposed for synchronization errors were based on Varshamov-Tenengolts (VT) codes [12], which can correct only one deletion or one insertion. These solutions were further extended to correct one burst of insertions and deletions [13], [14], or a few errors [15], [16], or to consider constrained codes together with the correction of a single error [17]. While solutions that guarantee the correction of a fixed number of errors are interesting from a theoretical perspective, they may not be sufficient in our context, since our channel model reveals a large and random amount of errors of the three types. Therefore, recent works consider either Convolutional Codes (CC) [18]–[20], Low Density Parity Check (LDPC) codes [21]–[24], or Polar codes [25], which use modified decoders that can correct not only substitutions, but also insertions and deletions. It is worth mentioning that the above works only consider non-uniform i.i.d. channel models.

In this work, we start from the concatenated code construction of [19], which is built from an inner CC designed to re-synchronize the sequence (*i.e.*, eliminate all insertions and deletions, correct a part of the substitutions), and from an outer LDPC code which corrects the remaining substitution errors. We consider this construction due to its good error-correction capabilities for error rates between 1% and 10% (when using an i.i.d. channel), which aligns with the error

rates observed in our model. In this paper, we propose to modify the convolutional decoder of [19] so as to incorporate the statistics of our channel model. This involves expanding the space of states used in the BCJR algorithm, and adapting the forward recursion, backward recursion, and branch metric computation based on our model. Numerical results show that although the modified BCJR decoder has a much larger complexity than in the i.i.d. case, it leads to significant performance improvement in terms of Bit Error Rate (BER) and Frame Error Rate (FER) compared to the decoder of [19]. It is worth noting that a similar work on adapting the concatenated code construction for our channel model [11] was done in [26]. However, [26] only introduced the new space of states, while we describe the practical implementation of all the decoding steps. In addition, [26] evaluated the theoretical performance of the decoder in terms of achievable information rate, while we present numerical BER and FER values to evaluate the practical decoder performance.

The outline of the paper is as follows. Section II describes the DNA data storage workflow. Section III-B introduces the channel model with memory. Section IV presents the considered concatenated code construction. Section V explains how to integrate our channel model into convolutional decoders. Section VI shows numerical results.

II. DNA DATA STORAGE

This section outlines the main two physical processes involved in DNA data storage: synthesis and sequencing. In what follows, we use $[\![1, J]\!]$ to denote the set of integers between 1 and J.

A. Chemical Synthesis

DNA synthesis consists of converting digital data into DNA strands, where each DNA strand is formed by different combinations of nucleotides A, C, G, and T [27]. There exists different synthesis techniques, each with different constraints (maximum length for the input sequences, costs, etc.). In this work, we consider chemical synthesis [27], [28], in which oligonucleotides (short DNA molecules) are synthesized and then assembled to form the ordered sequences of nucleotides. The synthesis produces thousands of DNA molecules, each representing a copy of the same synthesized sequence. In this work, we assume that synthesis does not introduce any error, as observed in [5]. Formally, in what follows, we use $\mathbf{x} = (x_1, \cdots, x_N)$ to denote the sequence of digits to be synthetized, where each digit x_t takes values in a quaternary alphabet.

Depending on the purpose, we either consider that $x_t \in \{A, C, G, T\}$, or that $x_t \in \text{GF}(4)$, where $\text{GF}(4)$ is the Galois Field of order 4, and there is a one-to-one correspondence between the alphabet $\{A, C, G, T\}$ and $\text{GF}(4)$.

B. Nanopore Sequencing

DNA sequencing consists of reading the DNA strands to produce a digital signal. In this work, we consider the ONT MinION sequencer, which has the ability to read at high speed but with a high rate of error [29]. In the MinION sequencer, DNA strands pass through a so-called nanopore. For each group of k successive nucleotides

$$\boldsymbol{k}\mathbf{mer}_t = (x_{t-k+1}, x_{t-k+2}, \cdots, x_t), \tag{1}$$

called k-mer, the nanopore outputs a specific electrical current level c_t. The next k-mer

$$\boldsymbol{k}\mathbf{mer}_{t+1} = (x_{t-k+2}, x_{t-k+3}, \cdots, x_{t+1}), \tag{2}$$

produces the current level c_{t+1}. Therefore, two successive levels c_t and c_{t+1} are produced from two k-mers with $k-1$ bases in common. A significant number of DNA strands are processed by the sequencer, which outputs a large number of current levels sequences.

Then, a software called basecaller transforms the sequences of current levels into digital sequences. In this work, we employed Guppy and Bonito basecallers, which are both built from Deep-Learning approaches. We use $\mathbf{y}^{(j)}$ to denote the J digital sequences output by the basecaller, where $j \in [\![1, J]\!]$, and $y_t^{(j)}$ takes values in a quaternary alphabet $\{A, C, G, T\}$ or $\text{GF}(4)$. The read sequence $\mathbf{y}^{(j)}$ is of length $P^{(j)}$, and typically $P^{(j)} \neq N$. This is because the sequencer and the basecaller not only introduce substitutions, but also insertions and deletions in the read sequences, and error patterns vary from sequence to sequence [5].

III. Channel model

From an information-theoretic perspective, the successive three previous steps (synthesis, sequencing, basecalling) can be modeled as a channel. This section introduces a new channel model, which turns out to be very useful for the design and performance evaluation of error-correction codes dedicated to DNA data storage. The software for the channel simulator is available online: https://github.com/BHam-1/DNArSim.

A. Notation

In what follows, we use Ins, Del, Sub, as abbreviations for Insertion, Deletion, Substitution, respectively. In addition, Match stands for "no error". In order to represent the channel effect, we consider an event sequence $\mathbf{e}$ of length N, where $e_t \in \{\text{Ins}, \text{Del}, \text{Sub}, \text{Match}\}$ is the channel event at position t. Interestingly, the sequences $\mathbf{e}$ and $\mathbf{x}$ have the same length N, and event e_t applies to input symbol x_t. More into details, $e_t =$ Del means that the symbol x_t is deleted from the sequence, and $e_t =$ Sub means that the base value of x_t is replaced by another base value. In addition, $e_t =$ Ins means that one or several symbols are inserted just after x_t. We denote by L_t the insertion length, that is the number of symbols inserted at position t. To make this notation clear, we introduce the following example.

Example : *Consider an input sequence of length* $N = 7$*:*

$$\mathbf{x} = [A, C, G, A, T, G, A],$$

and one output sequence of length $P = 8$*:*

$$\mathbf{y} = [A, C, C, C, G, T, T, A].$$

The corresponding sequence of channel events that generated $\mathbf{y}$ *from* $\mathbf{x}$ *is the sequence of length* $N = 7$*:*

$$\mathbf{e} = [\textit{Match}, \textit{Ins}, \textit{Match}, \textit{Sub}, \textit{Match}, \textit{Del}, \textit{Match}].$$

The channel inserted two symbols after x_2*, substituted symbol* x_4*, and deleted* x_6*. In addition the insertion length at position* 2 *is* $L_2 = 2$*.*

B. Proposed channel model with memory

Our proposed channel model captures the statistical dependency between the event sequence $\mathbf{e}$ and the input sequence $\mathbf{x}$, by considering that event e_t depends on $\boldsymbol{k}\mathbf{mer}_t$. This is consistent with the way the MinION sequencer operates, by reading one k-mer at a time. The MinION sequencer typically uses $k = 6$, but our model is flexible and can be adapted to different values of k, which will affect the model accuracy. Thus, our channel can be viewed as a Markov model of order k. Our model also captures some internal memory in $\mathbf{e}$, by considering that previous event e_{t-1} can affect current event e_t. This allows to represent bursts of errors.

Our statistical channel model is described by the following set of conditional probability distributions. First, the conditional probability distribution

$$\mathbb{P}(e_t|\boldsymbol{k}\mathbf{mer}_t, e_{t-1}) \tag{3}$$

of events $e_t \in \{\text{Ins}, \text{Del}, \text{Sub}, \text{Match}\}$, given $\boldsymbol{k}\mathbf{mer}_t$ and previous event e_{t-1}, captures the error dependency with the current k-mer, and allows to consider bursts of errors through the dependency to e_{t-1}. Second, given an insertion $e_t = \text{Ins}$ at position t, the conditional probability distribution

$$\mathbb{P}(L_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Ins}) \tag{4}$$

of the insertion length $L_t \in [\![1, L_{\max}]\!]$ depends on the read $\boldsymbol{k}\mathbf{mer}_t$. Note that a deletion at position t is always of length 1. Third, given a substitution $e_t = \text{Sub}$,

$$\mathbb{P}(B_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Sub}) \tag{5}$$

is the conditional probability distribution to substitute the last base x_t of $\boldsymbol{k}\mathbf{mer}_t$ by the base B_t, where $B_t \neq x_t$. We further assume that the previous three probability distributions (3), (4), (5), do not vary with $t \in [\![k, N-1]\!]$.

For $t \in [\![2, k]\!]$, since no complete k-mer was observed already, we consider conditional probability distributions

$$\mathbb{P}(e_t|x_t), \mathbb{P}(L_t|x_t, e_t = \text{Ins}), \mathbb{P}(B_t|x_t, e_t = \text{Sub}), \tag{6}$$

which only depend on the input value x_t. Finally, we observed from experimental data that the probabilities to get an insertion or a deletion are higher at the first position $t = 1$ and at the last one $t = N$, compared with middle positions $t \in [\![2, N-1]\!]$, and that these probabilities do not depend much on the input sequence. Therefore, we allow for different probability distributions

$$\mathbb{P}(e_1), \mathbb{P}(L_1|e_1 = \text{Ins}) \tag{7}$$

and

$$\mathbb{P}(e_N), \mathbb{P}(L_N|e_N = \text{Ins}), \tag{8}$$

for $t = 1$ and $t = N$, respectively. Given that substitutions are not affected by the previous remark, we still consider $\mathbb{P}(B_1|x_1, e_1 = \text{Sub})$ and $\mathbb{P}(B_N|\boldsymbol{k}\mathbf{mer}_N, e_N = \text{Sub})$ at the first and last position, respectively.

C. Model training

The next step is to estimate all the previous probability distributions from some sets of data. In this work, we trained the model using two distinct sets of data, each with specific characteristics: one set of experimental data, and one set of genomic data.

1) Set of experimental data (SetE): We first used a set of experimental data, callet SetE, that passed through the full DNA data storage process described in Section II. This set was generated from $U = 9$ input sequences $\mathbf{x}^{(u)}$ ($u \in [\![1, U]\!]$), called the reference sequences, each providing J_u output sequences $\mathbf{y}^{(u,j)}$ ($u \in [\![1, U]\!]$, $j \in [\![1, J_u]\!]$), for a total of $J = \sum_{u=1}^{9} J_u = 34604$ read sequences obtained after basecalling. In addition, 6 of the reference sequences have $N = 500$ nucleotides, and the 3 other ones have length $N = 1000$ nucleotides.

As a first step for training, each read sequence $\mathbf{y}^{(u,j)}$ was aligned with its reference sequence $\mathbf{x}^{(u)}$. The alignments were done using the ggsearch36 tool from the FASTA software [30]. This tool performs global to global alignments, that is to say that the whole read (from first to last base) is aligned against the whole reference. In a second step, we used the aligned reads to estimate conditional probabilities involving each k-mer contained in the sequences $\mathbf{x}^{(u)}$. For instance, we estimated the conditional probabilities $\mathbb{P}(e_t = D | \boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D')$ as

$$\frac{\mathbb{N}(e_t = D, \boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D')}{\mathbb{N}(\boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D')} \tag{9}$$

where $\mathbb{N}(.)$ counts the number of occurences of the event over all the aligned-read pairs $(\mathbf{x}^{(u)}, \mathbf{y}^{(u,j)})$, $D, D' \in \{\text{Ins}, \text{Del}, \text{Sub}, \text{Match}\}$, and $\boldsymbol{\beta}$ is a specific k-mer that can be any sequence of k bases. The other probability terms were estimated by following the same approach. At the end, the overall error probability over this dataset is high, about 10%.

The first advantage of this set of experimental data is that the reference sequences $\mathbf{x}^{(u)}$ are perfectly known, which makes the comparisons between the reads and the reference very reliable. On the other hand, this set contains only a small amount of data, due to high DNA synthesis costs. This leads to plenty of unobserved combinations $(\boldsymbol{k}\mathbf{mer}_t, e_{t-1} = E)$, over the 4^{k+1} possible ones. In our case, when a given combination $(\boldsymbol{k}\mathbf{mer}_t, e_{t-1} = E)$ was left unobserved, we estimated the corresponding probabilities by averaging over all observed combinations.

2) Set of genomic data (SetG): Genomic data are much more accessible than experimental data, as they can be obtained from various genomic databases. In this work, we considered a set of genomic data called SetG, that contains $U = 7$ strains (sub-types) [31] of the *Streptococcus*

thermophilus bacteria [32]. In this case, the reference sequences $\mathbf{x}^{(u)}$ contain about 10^6 bases, while the reads $\mathbf{y}^{(u,j)}$ correspond to subsequences of $\mathbf{x}^{(u)}$, and contain between 10^3 and 10^5 bases. This makes it impossible to use global alignments. In addition, there is no ground truth that states from which part of the reference $\mathbf{x}^{(u)}$ a given read $\mathbf{y}^{(u,j)}$ is produced. Thus, for the genomic dataset, we performed local alignments with a tool called minimap2 [33], which is very efficient in performing local alignments of very long sequences in a timely manner. In addition, we eliminated from the training set the reads $\mathbf{y}^{(u,j)}$ with a low-scoring alignment. We then used the same approach described in (9) in order to estimate all the probability terms. The overall error probability over this dataset is about 3%, which is significantly lower than for the experimental dataset. Note that since for SetE the reference sequences $\mathbf{x}^{(u)}$ and the reads $\mathbf{y}^{(u,j)}$ have the same length, global alignment is equivalent to local alignment.

The main advantage of genomic data is that it offers a significantly larger quantity of data. All possible combinations $(e_t = D, \boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D')$ were observed, with a sufficient amount of each. However, this dataset introduces another type of bias since the training discards low-scoring alignments, *i.e.*, sequences that have a high amount of errors. This explains why it gives a lower error probability compared to the experimental dataset.

D. Dynamic channel model

The two datasets provide two realistic channel models, with similar underlying characteristics but varying error rates. This will allow for the evaluation of the proposed error-correction codes under different conditions. To expand the performance evaluation even further, we propose a modified version of our channel model, which enables to consider various and intermediate levels of errors. This modified channel is referred to as the dynamic channel model.

Starting from a set of conditional probabilities obtained from one or the other training dataset, we introduce a unique scaling parameter α, and evaluate a new set of probabilities $\widetilde{\mathbb{P}}^{(\alpha)}(e_t = D|\boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D')$ defined for $D \in \{\text{Ins}, \text{Del}, \text{Sub}\}$ as

$$\begin{aligned}\widetilde{\mathbb{P}}^{(\alpha)}(e_t = D|\boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D') \\ = \alpha \mathbb{P}(e_t = D|\boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D'),\end{aligned} \tag{10}$$

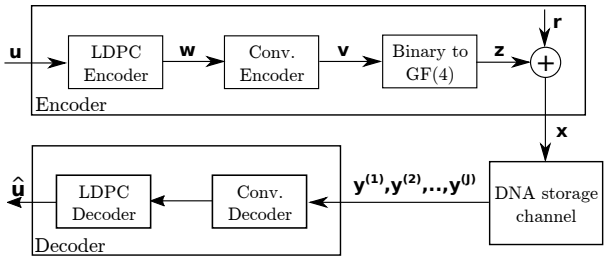

Fig. 1. Concatenated coding scheme.

and for $D = \text{Match}$ as

$$\widetilde{\mathbb{P}}^{(\alpha)}(e_t = D|\boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D') \\ = (1-\alpha)\mathbb{P}(e_t = D|\boldsymbol{k}\mathbf{mer}_t = \boldsymbol{\beta}, e_{t-1} = D'). \tag{11}$$

By construction, this new set of probabilities satisfies the normalization condition. It also allows to consider different channel error rates by varying the parameter α, as will be shown in the simulation section. For simplicity, we do not change other probability terms $\mathbb{P}(L_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Ins})$ and $\mathbb{P}(B_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Sub})$, given that they have less impact (for the first one), or no impact (for the second one) on the overall error probability.

IV. Concatenated codes

This section describes the concatenated code construction of [19], which we consider in this paper. As outer code, we consider an LDPC code defined by its parity check matrix H with rate R_0. According to [19], the considered LDPC code may be either binary or non-binary. In this section as well as in our numerical simulations, we only consider the case of a non-binary LDPC code in GF(4) which matches the channel alphabet. As inner code, we consider a binary CC with generator polynomial Δ and rate R_i. This inner code takes k bits as input, provides n bits as outputs, and has m registers. The CC aims to resynchronize the sequence, which means correcting all insertions and deletions, and it also corrects a part of the substitutions. The LDPC code corrects residual substitution errors. The concatenated coding scheme is depicted in Figure 1. We now describe its encoding and decoding steps.

A. Encoders

The outer LDPC code takes as input a quaternary sequence $\mathbf{u}$ of length K, and outputs a sequence of symbols in GF(4), which is then converted into a binary sequence $\mathbf{w}$ of length N_0. The inner convolutional encoder outputs a binary sequence $\mathbf{v}$ of length N_b, which is then converted into a sequence $\mathbf{z}$ of length $N = N_b/2$ in GF(4). Finally, the sequence $\mathbf{x}$ is computed as

$$\mathbf{x} = \mathbf{z} + \mathbf{r}, \tag{12}$$

where $\mathbf{r}$ is a random offset sequence of length N with symbols in GF(4), and where the sum is in GF(4). It was shown in [34] that the offset sequence $\mathbf{r}$ significantly improves the convolutional decoder performance under synchronization errors. At the end, the sequence $\mathbf{x}$ is synthetized as a DNA molecule.

B. Decoders

After sequencing, the DNA storage channel outputs the J sequences $\mathbf{y}^{(j)}$ in GF(4). We apply the inner convolutional decoder to $J' < J$ sequences $\mathbf{y}^{(j)}$. This decoder consists of a BCJR algorithm [35] which takes as inputs sequences in GF(4), and outputs Log Likelihood Ratios (LLRs) over bits. It also takes into account the offset sequence. We then convert the LLRs over bits onto LLRs over quaternary symbols, and pass them to a standard non-binary LDPC decoder, assuming that the convolutional decoder corrected all insertions and deletions, and that residual substitution errors are statistically independent. The LDPC decoder outputs a sequence $\hat{\mathbf{u}}$ of length K. In what follows, we introduce an updated version of the convolutional decoder of [19], [34], in order to take all the knowledge provided by our channel model into account.

V. Convolutional decoders

When designing the convolutional decoder, the first difficulty resides in the fact that insertions and deletions break the Markov property in the sequence of states [34]. Therefore, the convolutional decoder of [34] introduces an additional drift variable that restores the Markov property.

In this section, we first present the decoder of [34] which assumes an i.i.d. channel model. Then, we adapt this decoder to account for: (i) the memory in the sequence of error events, (ii)

the dependency between error events and k-mers. We begin by describing the decoders for a single sequence $\mathbf{y}$ of length P, and later explain how to handle multiple sequences $\mathbf{y}^{(j)}$.

A. State-of-the-art decoder with drifts (Dec1)

The decoder of [19], [34] is referred to as Dec1 in the remaining of the paper. We provide a detailed description of this decoder in order to establish notation and ensure that the paper is self-contained. When introducing the two other decoders, we will only highlight the differences compared to Dec1.

1) States of the decoder: The successive internal states of the CC are denoted s_t, where $t \in [\![0, N]\!]$. For instance, if the CC has 4 states, s_t takes values in $\{S_0, S_1, S_2, S_3\}$. Further, [34] introduces an additional state variable d_t, called the drift. The drift d_t represents the delay at time t in the sequence, that is

$$d_t = \text{Nb(INS)}_t - \text{Nb(DEL)}_t, \tag{13}$$

where Nb(INS)_t (respectively Nb(DEL)_t) is the number of insertions (respectively deletions) that occurred before transmitting the symbol x_t. Between time instants t and $t+1$, we assume that there is a maximum of $I_{\max}$ insertions and of 1 deletion. As a result, d_{t+1} lies in the interval $[\![d_t - 1, d_t + I_{\max}]\!]$. Overall, between time instants $t = 0$ and $t = N$, we assume that d_t lies in the interval $[\![D_{\min}, D_{\max}]\!]$. Both $I_{\max}$, $D_{\min}$, and $D_{\max}$, are parameters of the decoder. Overall, the state of the decoder is denoted by the pair

$$\boldsymbol{\sigma}_t = (s_t, d_t). \tag{14}$$

Figure 2 shows the trellis of the decoder, with state $\boldsymbol{\sigma}_t$ evolving between successive time instants $t = 0$, $t = 1$, and $t = 2$.

2) A posteriori *probability computation:* The CC decoder aims to compute *a posteriori* probabilities $\mathbb{P}(w_t|\mathbf{y})$, where

$$\mathbb{P}(w_t|\mathbf{y}) = \frac{\mathbb{P}(w_t, \mathbf{y})}{\mathbb{P}(\mathbf{y})} \tag{15}$$

and

$$\mathbb{P}(w_t, \mathbf{y}) = \sum_{(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) : w_t} \mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}). \tag{16}$$

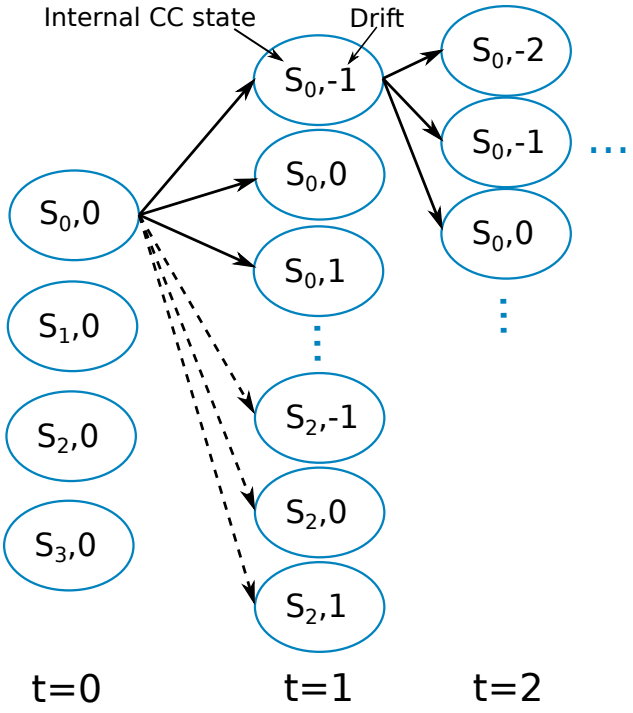

Fig. 2. Partial trellis diagram for Dec1, for time instants $t = 0$, $t = 1$, $t = 2$, with $I_{\max} = 1$ and $d_{t+1} \in [\![d_t - 1, d_t + 1]\!]$

During probability computation, the drift variable d_t allows to maintain a Markov property in the sequence of states in the sense that $\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$ can be decomposed as

$$\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \alpha_t(\boldsymbol{\sigma}_t)\beta_{t+1}(\boldsymbol{\sigma}_{t+1})\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) \tag{17}$$

where

$$\alpha_t(\boldsymbol{\sigma}_t) = \mathbb{P}(\mathbf{y}_1^{(t-1)n+dn}, \boldsymbol{\sigma}_t) \tag{18}$$

$$\beta_{t+1}(\boldsymbol{\sigma}_{t+1}) = \mathbb{P}(\mathbf{y}_{tn+d'n+1}^{T} | \boldsymbol{\sigma}_{t+1}) \tag{19}$$

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \mathbb{P}(\mathbf{y}_{(t-1)n+dn+1}^{tn+d'n}, \boldsymbol{\sigma}_{t+1} | \boldsymbol{\sigma}_t) \tag{20}$$

are evaluated from a forward recusion, backward recursion, and branch metric computation, which is the typical approach used in the BJCR algorithm. In the previous equations and in what follows, d refers to the current drift value d_t, and d' refers to the next drift value d_{t+1}. We often use d, d' instead of d_t, d_{t+1}, for improved readability of the equations.

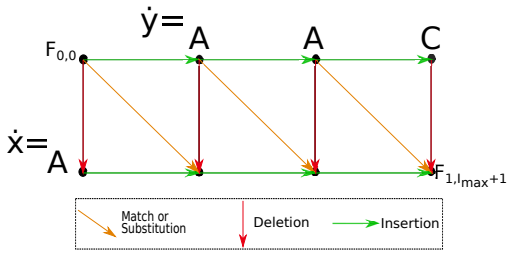

Fig. 3. Lattice structure used to compute branch metrics in Dec1.

3) Forward and Backward recursions: The forward recursion computes $\alpha_t(\boldsymbol{\sigma}_t)$ for all $t \in [\![1, N]\!]$ as

$$\alpha_t(\boldsymbol{\sigma}_t) = \sum_{\boldsymbol{\sigma}_{t-1}} \alpha_{t-1}(\boldsymbol{\sigma}_{t-1})\gamma_t(\boldsymbol{\sigma}_{t-1}, \boldsymbol{\sigma}_t) \tag{21}$$

where $\alpha_0(\boldsymbol{\sigma}_0)$ is initialized as

$$\alpha_0(\boldsymbol{\sigma}_0) = \begin{cases} 1, & \text{if } \boldsymbol{\sigma}_0 = (0,0) \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

The backward recursion computes $\beta_{t-1}(\boldsymbol{\sigma}_t)$ for all $t \in [\![0, N-1]\!]$ as

$$\beta_t(\boldsymbol{\sigma}_t) = \sum_{\boldsymbol{\sigma}_{t+1}} \beta_{t+1}(\boldsymbol{\sigma}_{t+1})\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) \tag{23}$$

where $\beta_N(\boldsymbol{\sigma}_N)$ is initialized as

$$\beta_N(\boldsymbol{\sigma}_N) = \begin{cases} 1, & \text{if } \boldsymbol{\sigma}_N = (0, P-N) \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

4) Branch metric computation: The branch metric $\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$ can be expressed as

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1)n+dn+1}^{tn+d'n}, d_{t+1} | d_t, s_t, s_{t+1}) \tag{25}$$

where $\mathbb{P}(w_t) = 1/4$. The branch metric is evaluated by using an efficient algorithm based on a lattice structure [36], see an example in Figure 3. This lattice allows us to compute the probability to pass from state $\boldsymbol{\sigma}_t$ to state $\boldsymbol{\sigma}_{t+1}$, which corresponds to the emission of a certain symbol $\dot{x} = x_{(t-1)n+1}^{tn}$. We consider the observed sequence $\dot{\mathbf{y}} = \mathbf{y}_{(t-1)n+d+1}^{tn+d'}$, whose length corresponds to passing from drift d_t to drift d_{t+1}. We recursively compute the probabilities $F_{i,j}$ at lattice nodes $[i, j]$, such that $\forall i \in [\![0, 1]\!]$, $\forall j \in [\![0, I_{\max}+1]\!]$,

$$F_{i,j} = \mathbb{P}_d F_{i-1,j} + \frac{1}{4}\mathbb{P}_i F_{i,j-1} + Q(\dot{x}_i|\dot{y}_j)F_{i,j-1} \tag{26}$$

where

$$Q(\dot{x}_i|\dot{y}_j) = \begin{cases} \mathbb{P}_m, & \text{if } \dot{x}_i = \dot{y}_j \\ \frac{1}{3}\mathbb{P}_s, & \text{otherwise.} \end{cases} \tag{27}$$

In these expressions, $\mathbb{P}_d$ is the probability of a deletion, $\mathbb{P}_i$ is the probability of an insertion, $\mathbb{P}_m$ is the probability of a match, and $\mathbb{P}_s$ is the probability of a substitution. Note that in the decoder of [19], these probabilities come from the i.i.d. model. The computation is initialized as

$$F_{i,j} = \begin{cases} 1, & \text{if } i = 0 \text{ and } j = 0 \\ 0, & \text{if } i < 0 \text{ or } j < 0. \end{cases} \tag{28}$$

Moving vertically on the lattice means that a deletion occurred, represented by the first term in (26). Moving horizontally on the lattice means that an insertion occurred, represented by the second term in (26), where the factor $\frac{1}{4}$ represents the uniform probability to insert any base A, C, G, or T. Moving horizontally on the lattice means that either a match occurred if $\dot{x}_i = \dot{y}_j$, or a substitution occurred if $\dot{x}_i \neq \dot{y}_j$. Both cases are represented by the third term of (26), where $\frac{1}{3}$ represent the uniform probability to substitute the current base $\dot{x}$ by any of the three possible ones. Finally, after computing the last lattice node $F_{1,I_{\max}+1}$, we get

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \frac{1}{4} F_{1,d_{t+1}-d_t+1}. \tag{29}$$

Note that in our implementation, we first evaluate the lattice for the largest possible gap $I_{\max}+1$ between d_t and d_{t+1}, and we then extract partial values $F_{1,d_{t+1}-d_t+1}$ from the lattice computation, for $d_{t+1} - d_t + 1 < I_{\max} + 1$. This is more efficient than generating one lattice per possible value $d_{t+1} - d_t$.

B. Taking previous error events into account (Dec2)

We now extend the previous decoder to consider the memory between successive error events. This decoder will be referred to as Dec2 in the following. Note that this decoder is only presented here in order to next simplify the description of the third decoder taking into account our full channel model. Numerical simulation results will show that Dec2 does not improve the decoding performance compared to Dec1.

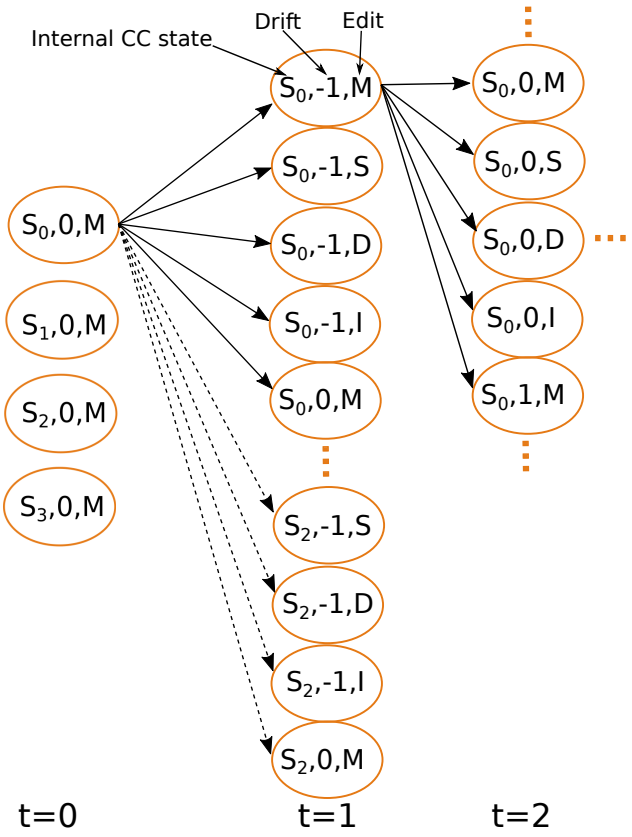

Fig. 4. Partial trellis diagram for Dec2, for time instants $t = 0$, $t = 1$, $t = 2$, with $I_{\max} = 1$ and $d_{t+1} \in [\![d_t - 1, d_t + 1]\!]$

1) States of the decoder: The state of the decoder is now given by a triplet

$$\boldsymbol{\sigma}_t = (s_t, d_t, e_t), \tag{30}$$

where $e_t \in \{\text{Ins}, \text{Del}, \text{Sub}, \text{Match}\}$ is the last edition observed before emitting symbol x_t. The edition variable e_t is added in order to preserve the Markov property when considering the previous error event e_{t-1}. This results in a larger trellis which has 4 times more nodes than for Decoder 1, see Figure 4 for an example. In addition, at time instant $t = 0$, we assume that the channel starts with a Match.

2) A posteriori *probability computation:* With this new state definition, relation (17) remains valid. To evaluate $\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$, we still use forward, backward recursions, and branch metric computation, each with modified expressions.

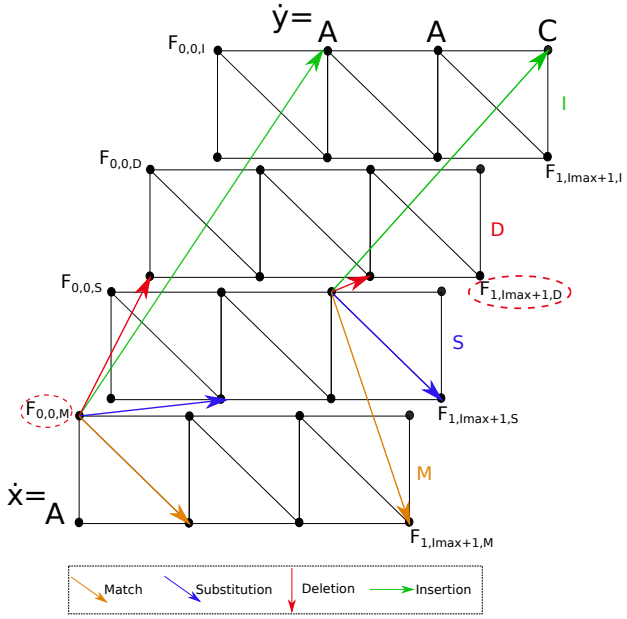

Fig. 5. 3D Lattice structure used to compute branch metrics in Dec2, for $e_t = M$ and $e_{t+1} = D$

3) Forward and Backward recursions: The forward recursion is still evaluated from (21), but the initialization is modified as

$$\alpha_0(\boldsymbol{\sigma}_0) = \begin{cases} 1, & \text{if } \boldsymbol{\sigma}_0 = (0, 0, M) \\ 0, & \text{otherwise.} \end{cases} \tag{31}$$

in order to consider the initial edit $e_0 = M$. In the same way, the backward recursion is still evaluated from (23), but the initialization is modified as

$$\beta_N(\boldsymbol{\sigma}_N) = \begin{cases} \mathbb{P}_m, & \text{if } \boldsymbol{\sigma}_N = (0, P-N, M) \\ \mathbb{P}_s, & \text{if } \boldsymbol{\sigma}_N = (0, P-N, S) \\ \mathbb{P}_d, & \text{if } \boldsymbol{\sigma}_N = (0, P-N, D) \\ \mathbb{P}_i, & \text{if } \boldsymbol{\sigma}_N = (0, P-N, I) \\ 0, & \text{otherwise.} \end{cases} \tag{32}$$

to take into account the final event e_N.

4) Branch metric computation: Branch metric computation is more impacted by the additional state variable e_t. The branch metric $\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$ is now calculated as

$$\begin{aligned}&\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) \\ &\quad = \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1)n+dn+1}^{tn+d'n}, d_{t+1}, e_{t+1}|d_t, e_t, s_t, s_{t+1}),\end{aligned} \tag{33}$$

and it is again evaluated recursively from a lattice. However, now a 3D lattice is needed to account for the additional paths resulting from the dependency to previous errors. This new lattice contains four plans, where each plan represents a particular edition $e_t \in \{M, S, D, I\}$, as shown in Figure 5. The moving rules are the same as in the lattice for Decoder 1, except for the fact that we can now move from a plan to another one, depending on the considered edition e_t. For instance, in the case of an insertion, we should move horizontally from toward the insertion plan, see green arrows in Figure 5.

The recursive computation is initialized as

$$F_{i,j,e} = \begin{cases} 1, & \text{if } i = 0 \text{ and } j = 0 \text{ and } e = e_t \\ 0, & \text{if } i < 0 \text{ or } j < 0. \end{cases} \tag{34}$$

We now define $\mathbb{P}_{e_1 \to e_2}$ as the probability to observe edition $e_2 \in \{M, S, D, I\}$ given that the previous edition was $e_1 \in \{M, S, D, I\}$. The probabilities at successive nodes in the lattice can be calculated recursively by using the following formulas:

$$\begin{aligned}F_{i,j,M} &= \mathbb{P}_{m\to m}F_{i-1,j-1,M} + \mathbb{P}_{s\to m}F_{i-1,j-1,S} \\ &\quad + \mathbb{P}_{d\to m}F_{i-1,j-1,D} + \mathbb{P}_{i\to m}F_{i-1,j-1,I} \\ F_{i,j,S} &= \frac{1}{3}(\mathbb{P}_{m\to s}F_{i-1,j-1,M} + \mathbb{P}_{s\to s}F_{i-1,j-1,S} \\ &\quad + \mathbb{P}_{d\to s}F_{i-1,j-1,D} + \mathbb{P}_{s}F_{i-1,j-1,I}) \\ F_{i,j,D} &= \mathbb{P}_{m\to d}F_{i-1,j,M} + \mathbb{P}_{s\to d}F_{i-1,j,S} \\ &\quad + \mathbb{P}_{d\to d}F_{i-1,j,D} + \mathbb{P}_{i\to d}F_{i-1,j,I} \\ F_{i,j,I} &= \frac{1}{4}(\mathbb{P}_{m\to i}F_{i,j-1,M} + \mathbb{P}_{s\to i}F_{i,j-1,S} \\ &\quad + \mathbb{P}_{d\to i}F_{i,j-1,D} + \mathbb{P}_{i\to i}F_{i,j-1,I})\end{aligned} \tag{35}$$

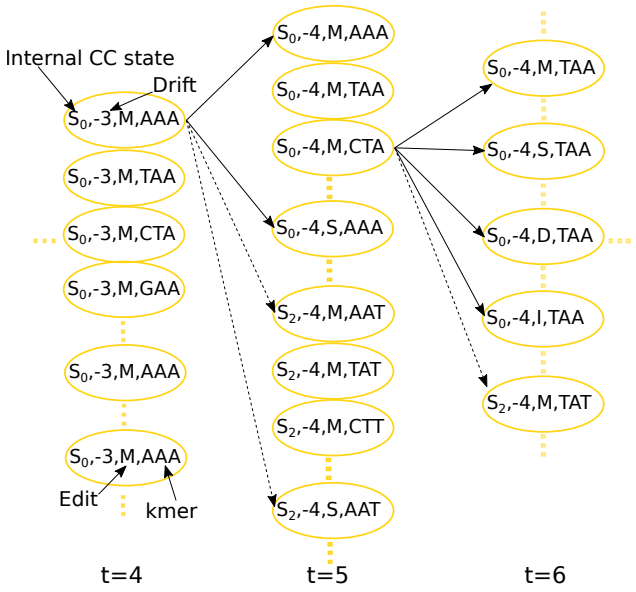

Fig. 6. Partial trellis diagram for Dec3, for time instants $t = 4$, $t = 5$, $t = 6$, with $I_{\max} = 1$ and $d_{t+1} \in [\![d_t - 1, d_t + 1]\!]$

where

$$\begin{cases} F_{i,j,S} = 0, & \text{if } \dot{x}_i = \dot{y}_j \\ F_{i,j,M} = 0, & \text{if } \dot{x}_i \neq \dot{y}_j. \end{cases} \tag{36}$$

At the end, we get

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \frac{1}{4} F_{1,d_{t+1}-d_t+1,e_{t+1}}. \tag{37}$$

C. Taking K-mers into account (Dec3)

In the third decoder, in addition to previous error events, we also consider the statistical dependency between the error event and the current k-mer. This decoder is referred to as Dec3.

1) States of the decoder: The state of the decoder is now given by a quadruplet

$$\boldsymbol{\sigma}_t = (s_t, d_t, e_t, \boldsymbol{\eta}_t), \tag{38}$$

where $\boldsymbol{\eta}_t$ is a vector of length K which gives the current k-mer. Especially, if

$$\boldsymbol{\eta_t} = [\eta_1^{(t)}, \eta_2^{(t)}, \cdots, \eta_K^{(t)}],$$

then

$$\boldsymbol{\eta_{t+1}} = [\eta_2^{(t)}, \eta_3^{(t)}, \cdots, \eta_K^{(t)}, x_{t+1}],$$

where x_{t+1} is the symbol emitted at time instant $t+1$. At time $t=0$, this new state variable is initialized as $\boldsymbol{\eta_0} = \emptyset$. In addition, for $t < K$, we consider $\boldsymbol{\eta_t} = [\eta_1^{(t)}, \eta_2^{(t)}, \cdots, \eta_{t-1}^{(t)}, \eta_t^{(t)}]$. This results in a larger trellis which has 2^k times more nodes than the trellis of Dec2, see Figure 6.

2) A posteriori *probability computation:* With this new state definition, (17) remains valid. To evaluate the three terms in $\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$, we still use a forward recursion, backward recursion, and branch metric computation, which we now describe.

3) Forward and Backward recursions: The forward recursion is still evaluated from (21), with initialization given by (31). The backward recursion is still evaluated from (23), but the initialization is updated as

$$\beta_N(\boldsymbol{\sigma}_N) = \begin{cases} \mathbb{P}(M|\boldsymbol{\eta}_N), & \text{if } \boldsymbol{\sigma}_N = (0, P-N, M) \\ \mathbb{P}(S|\boldsymbol{\eta}_N), & \text{if } \boldsymbol{\sigma}_N = (0, P-N, S) \\ \mathbb{P}(D|\boldsymbol{\eta}_N), & \text{if } \boldsymbol{\sigma}_N = (0, P-N, D) \\ \mathbb{P}(I|\boldsymbol{\eta}_N), & \text{if } \boldsymbol{\sigma}_N = (0, P-N, I) \\ 0, & \text{otherwise.} \end{cases} \tag{39}$$

to take into account the k-mers.

4) Branch metric computation: The branch metric $\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$ is now evaluated as

$$\begin{aligned} &\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) \\ &= \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1)n+dn+1}^{tn+d'n}, d_{t+1}, e_{t+1}, \boldsymbol{\eta}_{t+1} | d_t, e_t, \boldsymbol{\eta}_t, s_t, s_{t+1}) \\ &= \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1)n+dn+1}^{tn+d'n}, d_{t+1}, e_{t+1} | d_t, e_t, \boldsymbol{\eta}_{t+1}, s_t, s_{t+1}) \end{aligned} \tag{40}$$

since $\boldsymbol{\eta}_{t+1}$ is entirely determined by $\boldsymbol{\eta}_t$ and by the transition from internal state s_t to s_{t+1}. The branch metric is evaluated from the same 3D lattice used for Decoder 2 and shown in Figure 5. However, compared to Decoder 2, the recursive computation over the lattice now takes into account the observed k-mer $\boldsymbol{\eta}_t$.

We now consider the probability $\mathbb{P}(e_{t+1}|\boldsymbol{\eta}_t, e_t)$ of edit $e_{t+1} \in \{M, S, D, I\}$ conditioned to the k-mer $\boldsymbol{\eta}_t$, and to the previous edit $e_t \in \{M, S, D, I\}$. We use the following formula to recursively

compute the probabilities throughout the lattice:

$$
\begin{aligned}
&F_{i,j,M} = \\
&\mathbb{P}(M|\eta_{t+1}, M)F_{i-1,j-1,M} + \mathbb{P}(M|\eta_{t+1}, S)F_{i-1,j-1,S} \\
&+ \mathbb{P}(M|\eta_{t+1}, D)F_{i-1,j-1,D} + \mathbb{P}(M|\eta_{t+1}, I)F_{i-1,j-1,I} \\
&F_{i,j,S} = \\
&\frac{1}{3}(\mathbb{P}(S|\eta_{t+1}, M)F_{i-1,j-1,M} + \mathbb{P}(S|\eta_{t+1}, S)F_{i-1,j-1,S} \\
&+ \mathbb{P}(S|\eta_{t+1}, D)F_{i-1,j-1,D} + \mathbb{P}(S|\eta_{t+1}, I)F_{i-1,j-1,I}) \\
&F_{i,j,D} = \\
&\mathbb{P}(D|\eta_{t+1}, M)F_{i-1,j,M} + \mathbb{P}(D|\eta_{t+1}, S)F_{i-1,j,S} \\
&+ \mathbb{P}(D|\eta_{t+1}, D)F_{i-1,j,D} + \mathbb{P}(D|\eta_{t+1}, I)F_{i-1,j,I} \\
&F_{i,j,I} = \\
&\frac{1}{4}(\mathbb{P}(I|\eta_{t+1}, M)F_{i,j-1,M} + \mathbb{P}(I|\eta_{t+1}, S)F_{i,j-1,S} \\
&+ \mathbb{P}(I|\eta_{t+1}, D)F_{i,j-1,D} + \mathbb{P}(I|\eta_{t+1}, I)F_{i,j-1,I})
\end{aligned} \tag{41}
$$

At the end, we get

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \frac{1}{4}F_{1,d_{t+1}-d_t+1,e_{t+1}}, \tag{42}$$

as for Dec2.

D. Decoding with several sequences

The previous three decoders consider only one output sequence $\mathbf{y}$, while the DNA storage channel produces J sequences $\mathbf{y}^{(j)}$. In [19], it is proposed to decode each sequence $\mathbf{y}^{(j)}$ independently and separately, and to aggregate the results *a posteriori* by relying on the following formula:

$$\mathbb{P}(w_t|\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \cdots, \mathbf{y}^{(J)}) = \frac{\prod_{j=1}^{J}\mathbb{P}(w_t, \mathbf{y}^{(j)})}{\mathbb{P}(w_t)^{J-1}}. \tag{43}$$

Note that such aggregation could be realized at different levels of the decoding, but it was shown in [19] that it is more efficient to do it after applying the inner CC decoder. Note that [19] also proposed an alternative technique that is more efficient but also more complex to take into account

(Previous)	INS	DEL	SUB
Match	687	1579	1830
INS	12	12	4072
DEL	0	4096	0
SUB	42	3	4051

TABLE I

FOR THE MODEL TRAINED ON SETG, OVER 4096 k-MERS WITH $k = 6$, NUMBER OF TIMES THE LARGEST PROBABILITY IS FOR EVENT EITHER INS, DEL, OR SUB (COLUMNS) WITH RESPECT TO THE PREVIOUS ERROR EVENT (ROWS). FOR INSTANCE, WHEN THE PREVIOUS ERROR EVENT IS A MATCH (FIST ROW), FOR 687 OF THE k-MERS, THE PROBABILITY OF AN INSERTION IS LARGER THAN THE PROBABILITY OF A DELETION OR A SUBSTITUTION.

multiple sequences directly within the CC decoder. Given that Dec2 and Dec3 are already very complex, we leave for future works the investigation of this other technique.

VI. SIMULATION RESULTS

In this section, we first present numerical results for the channel models. We then investigate the performance of the proposed convolutional decoder.

A. Channel models

We first evaluate the proposed channel model against existing ones. In [11], we compared the edit maps of different models, and found that our model was visually the most similar to experimental data. Here, alternatively, we first provide statistics of error events in order to justify the assumptions of our model. We then use a more systematic approach which is based on the Kulback-Leibler (KL) divergence [37], [38] to evaluate the similarity between the simulated sequences for each model and the reference sequences.

1) Channel statistics: We now show some statistics of our model trained from the set of genomic data SetG. Indeed, in SetG, all k-mers appear a sufficient amount of time, which avoids any bias that could be introduced by unobserved k-mers. In this part, we consider $k = 6$, which is in accordance with the way the MinION sequence works.

First, Figure 7 shows the repartition of probabilities over k-mers for each event INS, DEL, SUB, depending on the previous event. For instance, we see that when the previous event is

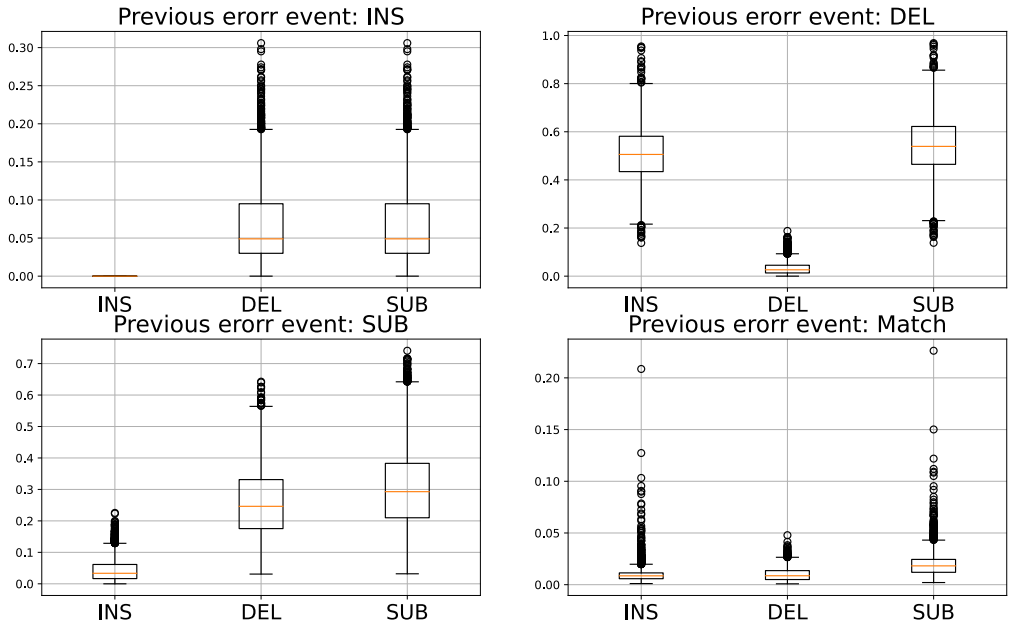

Fig. 7. Boxplots of error event probabilities depending on the previous event. This shows the probability repartition over k-mers for each event.

(Previous)	A	C	G	T
A	×	0.149	0.675	0.176
C	0.351	×	0.173	0.476
G	0.756	0.076	×	0.168
T	0.328	0.424	0.248	×

TABLE II

WHEN THERE IS A SUBSTITUTION ERROR, PROBABILITY THAT THE SYMBOL IN THE ROW IS REPLACED BY THE SYMBOL IN THE COLUMN. FOR INSTANCE, THE PROBABILITY THAT BASE C IS SUBSTITUTED BY BASE A IS 0.35. NOTE THAT PASSING FROM *e.g.*, BASE A TO A IS NOT AN ERROR, THIS IS WHY THERE ARE NO VALUES SPECIFIED IN THE DIAGONAL.

a substitution (bottom left figure), the average insertion probability is around 0.02, while the average deletion probability is around 0.25. This allows to conclude that event probabilities depend on both the current k-kmer, and the previous error event, as we consider in our model. Then, Table I shows the number of k-mers for which the probability of a given error event is the largest among all error events, depending on the previous event. For instance, when the previous event is a Match, 1830 k-mers have a larger substitution error probability, while 687 k-mers

have a larger insertion probability. This confirms the observations made from Figure 7. Finally, Table II shows the probability to replace a base by another one, under a substitution error. For instance, the probability to substitute a G by an A is 0.756, while the probability to substitute a G by a C is only 0.0755. This shows that substitution probabilities are non-uniform over bases, as we consider in our model.

Note that the channel simulator PBSIM2 [9] considers probabilities over quality scores, where probabilities depend on the current k-mer as in our model. However, PBSIM2 considers a fixed ratio between error event probabilities, and samples errors first from the quality scores, and second from the predefined ratio. This is in contradiction with the statistics of Table I, which instead show that there is no fixed ratio between error events (e.g. for some k-mers, insertion probability is larger than deletion probability, and for other k-mers, this is the opposite). In addition, PBSIM2 considers uniform substitution probabilities over bases, which enters in contradiction with the statistics of Table II.

We next investigate the accuracy of our channel model against other models, for which the underlying assumptions do not enter into direct contradiction with the statistics shown in this section.

2) KL divergence: We now compare our model against three different existing models: (i) the i.i.d. model, (ii) DeepSimulator, as a representative of DL-based methods, (iii) BadRead, for which event probabilities directly depend on the k-mers, as in our model. We here consider the two datasets, namely SetE and SetG, given that they have some different characteristics.

Starting with the set of experimental data SetE, Figure 8 shows the KL divergence with respect to the memory order k for the i.i.d. channel, DeepSimulator, BadRead, and our model. The channel models were all trained on SetE, except for DeepSimulator. Then, the KL divergence was averaged over the sequences of SetE. First note that only the curve for our model varies with k, because this is a parameter of our model only. Second, we observe that our model has the lowest KL divergence, while the i.i.d. model surprinsingly comes second. We see that for our model, $k = 1$ gives a better KL divergence than the i.i.d. model, because it still takes into account memory in successive events e_t, and dependency with the input symbol x_t. In addition, we see that for our model, when k is large enough, the KL divergence is close (but not equal) to 0, which means that our model trained on SetE represents accurately the dataset SetE, even for a value of k as small as 9 or 10.

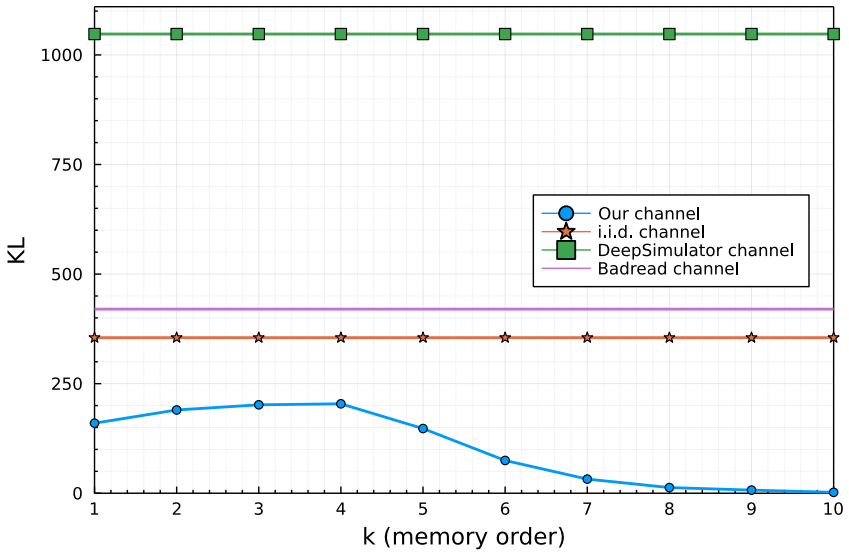

Fig. 8. KL divergence for different channel models compared with *SetE*. Except for DeepSimulator, all the other channel models were trained on *SetE*.

Then, in Figure 9, BadRead and DeepSimulator were taken from their own knowledge (from the online code), while our model as well as the i.i.d. model were trained on SetG. The KL divergence values were then calculated over SetE. One first important remark is that in this case, the KL divergence values are much higher than in Figure 8, because the models were trained on a dataset and evaluated on another one. Interestingly, our model again shows the lowest KL divergence over all the models, and the i.i.d. model still comes second. It is worth noting that this time, the value $k = 6$ is the one that provides the lowest KL divergence, while the KL values start increasing again from $k = 7$. This suggests that $k = 6$ leads to a better generalization capability for our model, which is consistent with the way the MiniON sequencer works (with $k = 6$). Therefore, in what follows, we only consider our model with $k = 6$.

B. CC decoders performance

We now evaluate the performance of the three CC decoders, by running Monte-Carlo simulations over our memory channel model. We consider the two versions of our memory channel model, *i.e.*, the one trained on the experimental data SetE, and the one trained on the genomic data SetG. Both versions are considered because the overall error rates differ between the two models, as explained in Section III. Each simulation run generates a random binary sequence $\mathbf{w}$,

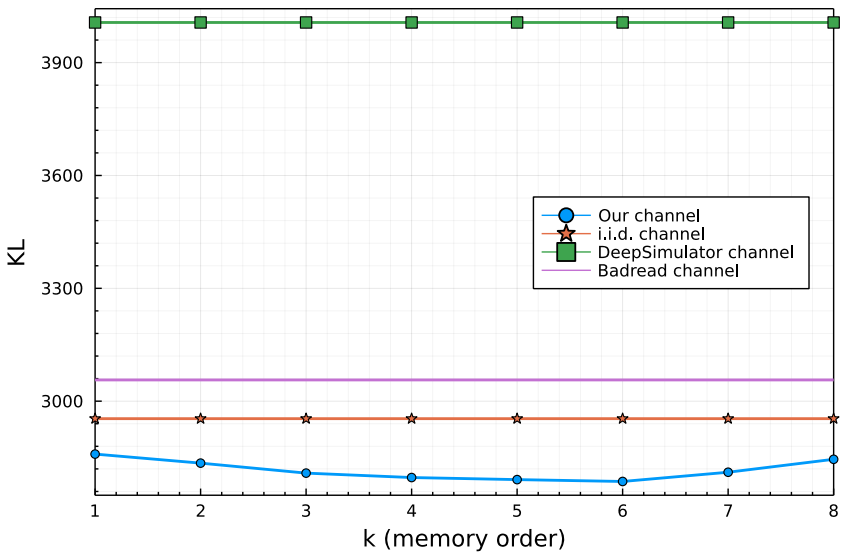

Fig. 9. Kullback-Leibler divergence between channel models simulation results and *SetE*. BadRead and DeepSimulator use their own knowledge. Our channel model and the i.i.d channel model were trained on *SetG*.

uniformly at random. Then $\mathbf{w}$ is encoded with a $(k_c = 1, n_c = 2, K_c = 3)$ CC of rate $1/2$, which uses the generator polynomial $\delta_{poly} = [\delta^2 + 1, \delta^2 + \delta + 1]$, and outputs a binary encoded sequence $\mathbf{x}$ of a certain length N. We consider values $N \in \{54, 204\}$, which falls in the range of length of DNA molecules produced by current synthesis techniques [39]. The sequence $\mathbf{x}$ is then passed through our memory channel model, and the CC decoder takes as input J sequences output by the channel. For each considered value J, we perform a maximum of 50000 simulation runs to evaluate the FER and BER of each decoder, and stop the evaluation after 100 frames in error.

Figure 10 shows the FER and BER with respect to J, of the three decoders over the memory channel model trained onto *SetE*, for $N = 54$. We observe that Dec3 has the best performance both in terms of FER and BER, with a clear performance gain compared to Dec1 of [19]. This can be explained by the fact that Dec3 fully takes into account the channel model. The performance gain is even more significant when the number of sequences J increases. More surprisingly, we also observe that Dec2 performs worst, most probably because this decoder only partially takes into account our channel model. Note that we also observed that the decoder that considers partial channel model through the state $\boldsymbol{\sigma}_t = (s_t, d_t, \boldsymbol{\eta}_t)$ (*e.g.*, e_t is replaced by $\boldsymbol{\eta}_t$) also performs worst than Dec1.

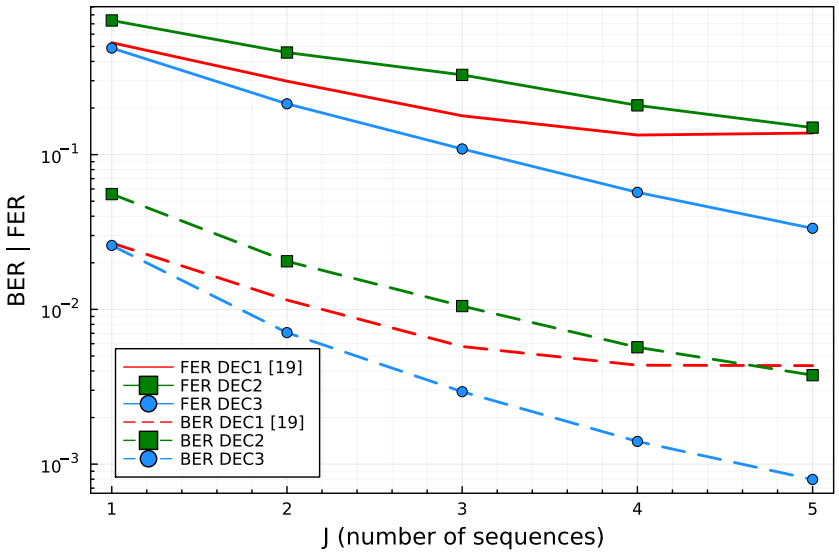

Fig. 10. FER and BER with respect to the number of sequences J, over the memory channel model trained onto *SetE*, for N=54. Dashed lines represent the BER and plain lines represent the FER.

In addition, Figure 11 shows the FER and BER with respect to J, of the three decoders over the memory channel model trained onto *SetG*, for $N = 204$. In this setup as well, we observe that Dec3 has a better performance than Dec1, and that Dec2 still performs poorly. Note that the FER and BER values are lower in Figure 11 than in Figure 10, because the overall error probability on *SetG* is lower than on *SetE*.

C. Concatenated codes performance

In order to evaluate the concatenated code construction, we now consider the dynamic channel model introduced in Section III-D. Starting with the set of probabilities obtained from the genomic data, we consider various parameters α which provide different channel error rates. The channel error rate for a given α was numerically evaluated from Monte Carlo simulations. Table III provides the measured error rates for each considered value of α.

We now consider the concatenated code construction introduced in Section IV, with the same CC considered in the previous simulation results, used together with a regular non-binary LDPC code of rate $R = 4/5$ in GF(4), and with $N = 50$. In the decoding part, we consider Dec3 followed by a standard BP decoder in GF(4). Figure 12 shows the FER with respect to the channel error probability, for various values of J for the CC alone, and for $J = 1$ for the full

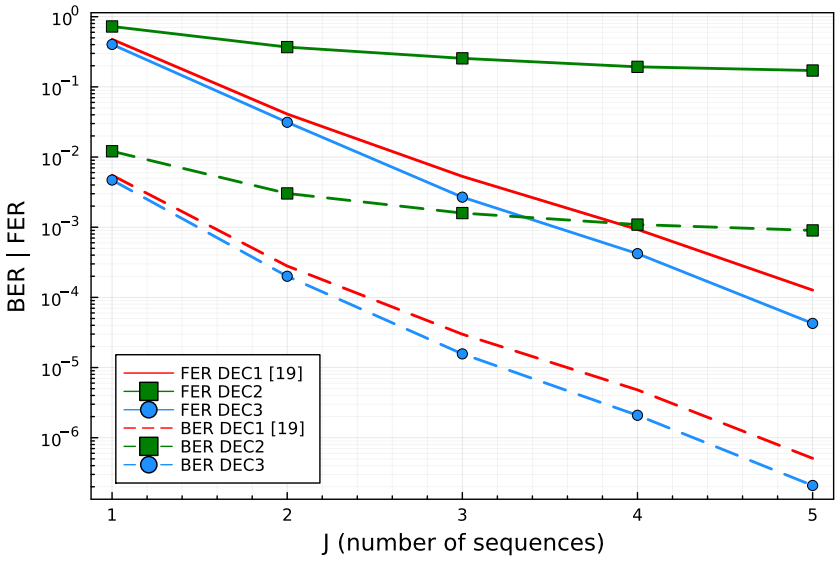

Fig. 11. FER and BER with respect to the number of sequences J, over the memory channel model trained onto *SetG*, for N=204. Dashed lines represent the BER and plain lines represent the FER.

α	0.13	0.28	0.52	0.8	1.05	1.25	1.55	1.85	2.12	2.4	2.7
Error probability	0.005	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1

TABLE III

ERROR PROBABILITY WITH RESPECT TO THE VALUE OF α FOR THE DYNAMIC CHANNEL MODEL STARTING FROM THE SET OF PROBABILITIES ESTIMATED FROM SETG. THE ERROR PROBABILITIES WERE ESTIMATED FROM MONTE-CARLO SIMULATIONS.

concatenated code construction. Figure 13 shows the same curves but in terms of BER. In both cases, we observe a significant gain of the concatenated construction with $J = 1$ compared to the CC alone with $J = 1$. In addition, in terms of FER, the concatenated constructions with $J = 1$ and $J = 2$ outperform the CC decoder alone with $J = 3$. For the BER, while the concatenated construction with $J = 2$ is best, the same construction with $J = 1$ is not as good as the CC decoder alone with $J = 3$. One explanation is that the CC decoder alone is known to perform poorly in terms of FER, while it already has a fairly good BER performance. This also illustrates the tradeoff between the coding rate and the number of sequences used for decoding.

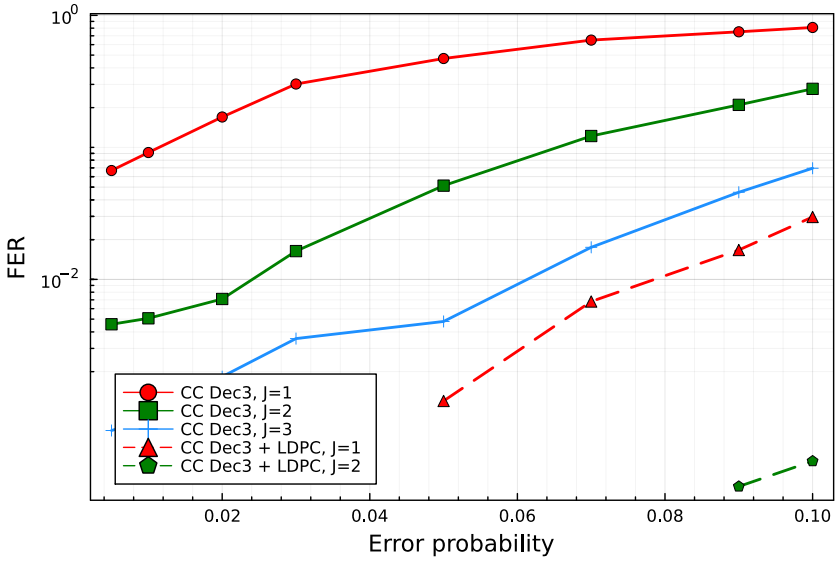

Fig. 12. FER with respect to the overall error rate for the *dynamic* channel model, for the CC Dec3, and for the concatenated code construction with Dec3 and a NB-LDPC decoder, with $N = 50$.

VII. Conclusion

This paper introduced a new channel model for DNA data storage, which can be seen as a Markov model of order k and was trained on experimental and genomic datasets. The proposed model was shown to be more accurate than existing ones through numerical evaluation of the KL divergence. Additionally, this work improved the CC decoder of [19] by incorporating the knowledge of the channel model, resulting in better BER and FER performance. Future works will focus on reducing the complexity of the proposed decoder.

Acknowledgement

We acknowledge Emeline Roux for kindly providing the experimental and genomic datasets used in this work.

References

[1] C. Yang, J. Chu, R. L. Warren, and I. Birol, "NanoSim: nanopore sequence read simulator based on statistical characterization," *GigaScience*, vol. 6, no. 4, p. gix010, Apr. 2017.

[2] E. A. G. Baker, S. Goodwin, W. R. McCombie, and O. Mendivil Ramos, "Silico: A simulator of long read sequencing in pacbio and oxford nanopore," *bioRxiv*, 2016. [Online]. Available: https://www.biorxiv.org/content/early/2016/09/22/076901

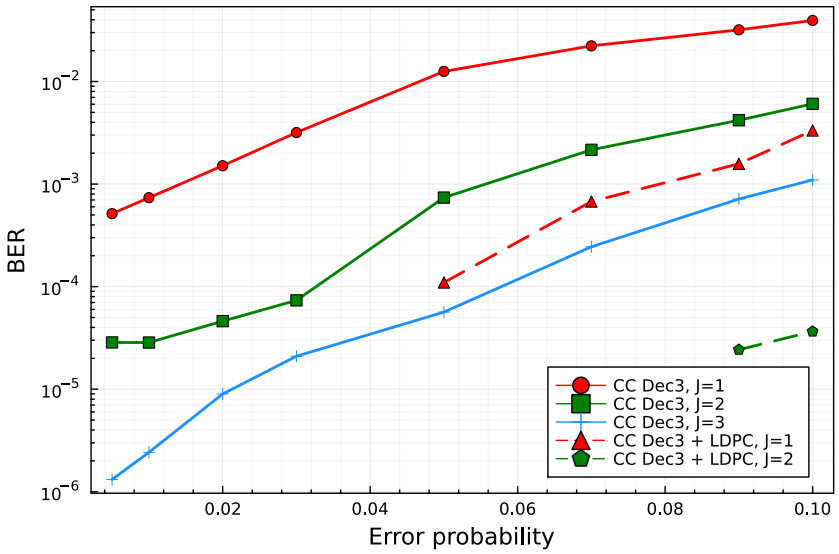

Fig. 13. BER with respect to the overall error rate for the *dynamic* channel model, for the CC Dec3, and for the concatenated code construction with Dec3 and a NB-LDPC decoder, with $N = 50$.

[3] L. Conde-Canencia and L. Dolecek, "Nanopore DNA sequencing channel modeling," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2018, pp. 258–262.

[4] E. G. San Antonio, T. Heinis, L. Carteron, M. Dimopoulou, and M. Antonini, "Nanopore sequencing simulator for dna data storage," in *International Conference on Visual Communications and Image Processing (VCIP)*, 2021, pp. 1–5.

[5] R. Heckel, G. Mikutis, and R. N. Grass, "A Characterization of the DNA data storage channel," *Scientific Reports*, vol. 9, no. 1, p. 9663, 2019.

[6] Y. Li, R. Han, C. Bi, M. Li, S. Wang, and X. Gao, "DeepSimulator: a deep simulator for Nanopore sequencing," *Bioinformatics*, vol. 34, no. 17, pp. 2899–2908, 2018.

[7] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao, "DeepSimulator1. 5: a more powerful, quicker and lighter simulator for nanopore sequencing," *Bioinformatics*, vol. 36, no. 8, pp. 2578–2580, 2020.

[8] S. Kang, Y. Gao, J. Jeong, S.-J. Park, J.-W. Kim, J.-S. No, H. Jeon, J. W. Lee, S. Kim, H. Park, and A. No, "Generative adversarial networks for DNA storage channel simulator," *IEEE Access*, vol. 11, pp. 3781–3793, 2023.

[9] Y. Ono, K. Asai, and M. Hamada, "PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores," *Bioinformatics*, vol. 37, no. 5, pp. 589–595, 2021.

[10] R. R. Wick, "Badread: simulation of error-prone long reads," *Journal of Open Source Software*, vol. 4, no. 36, p. 1316, 2019. [Online]. Available: https://doi.org/10.21105/joss.01316

[11] B. Hamoum, E. Dupraz, L. Conde-Canencia, and D. Lavenier, "Channel model with memory for DNA data storage with nanopore sequencing," in *11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1–5.

[12] R. R. Varshamov and G. M. Tenengol'ts, "A code that corrects single unsymmetric errors," *Avtomatika Telemekhanika*, vol. 26, no. 2, pp. 288–292, 1965.

[13] V. Levenshtein, "Asymptotically optimum binary code with correction for losses of one or two adjacent bits," *Problemy*

Kibernetiki, vol. 19, pp. 293–298, 1967.

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

[15] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[16] A. Helberg and H. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 305–308, 2002.

[17] X. Lu and S. Kim, "Design of nonbinary error correction codes with a maximum run-length constraint to correct a single insertion or deletion error for DNA storage," *IEEE Access*, vol. 9, pp. 135 354–135 363, 2021.

[18] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman, and H. Ji, "Overcoming high nanopore basecaller error rates for DNA storage via basecaller-decoder integration and convolutional codes," in *ICASSP*, 2020, pp. 8822–8826.

[19] A. Lenz, I. Maarouf, L. Welter, A. Wachter-Zeh, E. Rosnes, and A. G. i Amat, "Concatenated codes for recovery from multiple reads of DNA sequences," in *IEEE Information Theory Workshop (ITW)*, 2021, pp. 1–5.

[20] S. R. Srinivasavaradhan, S. Gopi, H. D. Pfister, and S. Yekhanin, "Trellis BMA: Coded trace reconstruction on IDS channels for DNA storage," in *IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2453–2458.

[21] S. Chandak, K. Tatwawadi, B. Lau, J. Mardia, M. Kubit, J. Neu, P. Griffin, M. Wootters, T. Weissman, and H. Ji, "Improved read/write cost tradeoff in DNA-based data storage using LDPC codes," in *Allerton*, 2019, pp. 147–156.

[22] F. Wang, D. Fertonani, and T. M. Duman, "Symbol-level synchronization and LDPC code design for insertion/deletion channels," *IEEE transactions on communications*, vol. 59, no. 5, pp. 1287–1297, 2011.

[23] R. Shibata, G. Hosoya, and H. Yashima, "Design of irregular LDPC codes without markers for insertion/deletion channels," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[24] L. Deng, Y. Wang, M. Noor-A-Rahim, Y. L. Guan, Z. Shi, E. Gunawan, and C. L. Poh, "Optimized code design for constrained DNA data storage with asymmetric errors," *IEEE Access*, vol. 7, pp. 84 107–84 121, 2019.

[25] H. D. Pfister and I. Tal, "Polar codes for channels with insertions, deletions, and substitutions," in *IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2554–2559.

[26] I. Maarouf, E. Rosnes *et al.*, "Achievable information rates and concatenated codes for the dna nanopore sequencing channel," *arXiv preprint arXiv:2212.07287*, December 2022.

[27] M. Hao, J. Qiao, and H. Qi, "Current and Emerging Methods for the Synthesis of Single-Stranded DNA," *Genes*, vol. 11, no. 2, p. 116, 2020.

[28] R. A. Hughes and A. D. Ellington, "Synthetic DNA synthesis and assembly: putting the synthetic in synthetic biology," *Cold Spring Harbor perspectives in biology*, vol. 9, no. 1, p. a023812, 2017.

[29] Y. Wang, Y. Zhao, A. Bollas, Y. Wang, and K. F. Au, "Nanopore sequencing technology, bioinformatics and applications," *Nature biotechnology*, vol. 39, no. 11, pp. 1348–1365, 2021.

[30] W. R. Pearson, "Finding protein and nucleotide similarities with fasta," *Current Protocols in Bioinformatics*, vol. 53, no. 1, pp. 3.9.1–3.9.25, 2016. [Online]. Available: https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/0471250953.bi0309s53

[31] W. Li, D. Raoult, and P.-E. Fournier, "Bacterial strain typing in the genomic era," *FEMS Microbiology Reviews*, vol. 33, no. 5, pp. 892–916, 09 2009. [Online]. Available: https://doi.org/10.1111/j.1574-6976.2009.00182.x

[32] E. Roux, A. Nicolas, F. Valence, G. Siekaniec, V. Chuat, J. Nicolas, Y. Le Loir, and E. Guédon, "The genomic basis of the streptococcus thermophilus health-promoting properties," *BMC Genomics*, vol. 23, no. 1, p. 210, Mar. 2022.

[33] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 05 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty191

[34] V. Buttigieg and N. Farrugia, "Improved bit error rate performance of convolutional codes with synchronization errors," in *IEEE International Conference on Communications (ICC)*, 2015, pp. 4077–4082.

[35] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on information theory*, vol. 20, no. 2, pp. 284–287, 1974.

[36] L. Bahl and F. Jelinek, "Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 4, pp. 404–411, 1975.

[37] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[38] J. M. Joyce, *Kullback-Leibler Divergence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_327

[39] E. M. LeProust, B. J. Peck, K. Spirin, H. B. McCuen, B. Moore, E. Namsaraev, and M. H. Caruthers, "Synthesis of high-quality libraries of long (150mer) oligonucleotides by a novel depurination controlled process," *Nucleic Acids Res.*, vol. 38, no. 8, pp. 2522–2540, May 2010.