DNA data storage algorithms and synchronization

Belaid Hamoum

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE BRETAGNE SUD

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Télécommunications*

Par

« Belaid HAMOUM »

« DNA data storage algorithms and synchronization »

« Algorithmes pour la synchronisation de données et leur stockage sur ADN »

Thèse présentée et soutenue à « Lorient », le « 15/12/2022 »
Unité de recherche : Lab-STICC
Thèse N° : «640»

Rapporteurs avant soutenance :

Marc ANTONINI	Directeur de recherche CNRS
Jossy SAYIR	Associate Teaching Professor University of Cambridge

Composition du Jury :

Président :	Aline ROUMY	Directrice de recherche INRIA
Examinateurs :	Iryna ANDRIYANOVA	Professeure CY Cergy Paris Université
		Maître de Conférence IMT ATLANTIQUE
Encadrante de thèse :	Elsa DUPRAZ	
Directeur de thèse :	Emmanuel BOUTILLON	Professeur Université Bretagne Sud

Dedication

To my parents, to my sisters and brothers.

Acknowledgement

Je souhaite tout d'abords remercier mes parents: Hadjila et Khelifa, ainsi que mes sœurs et frères: Lydia, Wardia, Sofiane, et Mokrane. Ils sont une source d'inspiration, de courage, de persévérance, et des océans de beaux principes et belles pensés. Vos encouragements, votre présence, et confiance infaillible ont grandement contribué à la réussite de cette thèse. Je tiens aussi à remercier mes grands-parents: Aldjia et Messaoud, ainsi que mes oncles et tantes: Idir, Ouali, Kamel, Mokrane, Mourad, Hassina et Houria qui m'ont toujours soutenu et contribué chacun à sa manière dans cette thèse.

Je remercie aussi Elsa Dupraz, maîtresse de conférences à l'IMT Atlantique, qui a dirigé et encadré les travaux de cette thèse d'une manière exceptionnelle. J'ai beaucoup apprécié travailler avec elle, particulièrement pour ses connaissances, sa supervision, et qualités relationnelles. Nos nombreuses réunions et discussions ont toujours débouchés sur de nombreuses idées et solutions pour avancer. Je tiens aussi à remercier Laura Conde-Canencia maîtresse de conférences à l'UBS, pour m'avoir offert l'opportunité de poursuivre en thèse, et d'avoir diriger la première année de ma thèse. Elle fait notamment partie des pionniers dans la recherche sur le stockage de données sur ADN en France, et j'ai bien apprécié travailler avec elle. Je voudrais aussi remercier Emmanuel Boutillon, qui a bien voulu reprendre la direction de cette thèse après le départ de Laura.

Je tiens également à remercier les membres de mon jury de thèse pour l'évaluation de mon travail. J'ai particulièrement apprécié les nombreux retours, questions et remarques liées à mon manuscrit et à ma soutenance de thèse. Un merci particulier à Marc Antonini et Jossy Sayir, rapporteurs de cette thèse, pour avoir consacré beaucoup de temps à la lecture de cette dernière, et pour leurs nombreux retours.

Je remercie aussi, chacun des membres du projet DnarXiv. J'ai apprécié chaque réunion de projet, particulièrement pour les nombreuses discussions, questions, et retours qui en découlent. Merci Olivier, Dominique, Emeline, Chloé, Julien, Jacques, Gouenou et Yann. Je remercie aussi Antonia Wachter-Zeh, professeure à l'Université Technique

de Munich, qui m'a accueilli dans son équipe pendant 3 mois. J'ai bien apprécié cette expérience grâce aux riches échanges et réunions sur le stockage de données sur ADN abordés avec Anisha, Lorenz et Andreas.

Un grand merci aussi à toutes les superbes personnes que j'ai rencontré dans le centre de recherche, et en dehors de celui-ci. Parmi eux : Noura, Ajinkya, Celia, Adel, Mohamed, Hani, Mohand, Béatrice, Cecilia, Thomas, et tant d'autres.

Table of Contents

Chapter 1

Introduction

Over the last decade, the amount of generated data has been growing up exponentially. In [1], it was predicted that data storage needs would grow from 45 zetabytes in 2019 to 175 zetabytes by 2025. But the growth in data storage capacity is below these forecasts [2]. In addition, a report published in 2020 by the European Commission [3] pointed out the increasing unbalance between offer and demand for raw materials needed to produce Classical Storage Medias (CSMs), where examples of CSMs include tapes, HDDs, or SDDs [4]. For these reasons, it is necessary to find alternatives to CSMs.

Among these alternatives, DNA data storage [2, 5, 6] appears as a promising solution that benefits from highly increased density and durability compared to existing storage solutions. Theoretically, DNA is 10^6 times more dense than tapes [2, 7, 8], which are themselves the densest CSM. In addition, DNA has a durability of over 500 years, while the durability of HDDs and Tapes is about 5 years and 30 years, respectively. Moreover, since CSMs used in data centers must be kept under a certain temperature, they are responsible of a high carbon footprint [9]. Oppositely, DNA can be stored at room temperature [10], and thus, DNA data storage naturally goes towards the green transition.

The idea of storing data on DNA goes back to the sixties. At this time, the idea of a future data storage media based on a genetic memory was discussed, and some of its advantages were already pointed out in [2, 11, 12]. However, the first two major experiments that showed the true potential of DNA data storage were both conducted much later, in 2012 [5, 6], thanks to the recent advances in DNA synthesis and sequencing [2]. Moreover, a fully automated end-to-end DNA data storage device demonstrator was reported by Microsoft and Washington University in 2019 [13].

1.1 Issues raised by DNA data storage in practice

Practical DNA data storage requires advanced technological processes from several scientific fields including biology, bioinformatics, data science, signal processing, channel

and source coding. In order to advance and mature this technology, several issues need to be addressed within these fields, or at the interface of these fields. This thesis will mostly focus on DNA data storage issues related to channel coding, with the objective of taking into account as much as possible constraints from other fields. We expect that this approach will allow to get closer to a practical implementation.

We now describe into details the issues addressed in this thesis and their connections with other scientific fields.

1.1.1 DNA sequencing and synthesis constraints

DNA sequencing is the operation by which the DNA is read, and DNA synthesis is the operation by which the DNA is written. New sequencing technologies allow to drastically reduce the cost and time of DNA sequencing [14, 15]. However, sequencing still takes from a few hours to a few days [8], depending on the amount of DNA to sequence. In addition, although major advances were observed recently, DNA synthesis still has a high cost [16]. And synthesis also takes from a few days to a few weeks, depending on the amount of DNA to synthesize.

Therefore, due to the synthesis and sequencing latencies, DNA data storage is currently dedicated to cold data storage [2, 17], where the term "cold" refers to data which must be stored for a long period of time (years, decades, centuries, ...). Furthermore, since cold data is rarely accessed, the storage devices are optimized for long-term preservation and low costs [8, 17]. For instance, the access latency in the Amazon Glacier storage service [18] can take up to 12 hours.

Of course, making progress in DNA synthesis and sequencing techniques is clearly out of the scope of this thesis. However, we will aim to identify and take into account the constraints of existing techniques during our studies. This is why in Chapter 2, we introduce into details the DNA data storage workflow and describe DNA synthesis and sequencing techniques.

1.1.2 Type and amount of errors introduced by DNA data storage

One of the main drawbacks of DNA data storage is the amount of errors introduced during DNA synthesis and sequencing [19, 20]. While DNA synthesis introduces a low amount of errors [19, 21], DNA sequencing is subject to a high amount of errors [19, 20].

These processes not only introduce substitutions, but also more unconventional deletion and insertion errors. Furthermore, the amount of errors can vary depending on the DNA sequencing technology and protocol [19, 20, 22].

Therefore, there is a need for accurate statistical models to represent the errors introduced by the DNA data storage channel. Indeed, a reliable statistical model would allow for *in silico* simulations of any numerical method (signal processing, coding, security, etc.) developed for DNA data storage, before developing costly *in vitro* experiments. It could also help better understand the different sources of errors. However, because of the biological process involved in DNA data storage, it is not straightforward to model the whole process. Hence, it is common to consider a simplified DNA data storage channel model, with incorrect assumptions on, *e.g.*, independent and identically distributed (i.i.d.) errors [23, 24].

Therefore, in Chapter 3, we propose a statistical channel model which accurately represents the DNA data storage channel. Our statistical model was trained on two different types of data: a set of experimental data which came through the full DNA data storage process, and a set of genomic data which were obtained from the sequencing of a bacteria.

1.1.3 Correction of errors introduced by the DNA data storage channel

In order to build reliable DNA data storage systems, it is necessary to implement efficient error-correction solutions. However, most conventional error-correction solutions [25, 26, 27] only correct substitution errors, and completely fail at correcting insertions and deletions. In this thesis, we will explore different solutions to correct insertion, deletion, and substitution errors. Especially, we will investigate two solutions which both resort on different techniques.

In Chapter 4, we introduce a first error-correction solution, which combines an approach from the bioinformatics field, and an approach from the coding field. This first solution was developed in collaboration with the GenScale team of INRIA Rennes. In Chapter 5, we introduce a second error-correction solution, which purely relies on channel coding so as to improve the error-correction process. This second solution was developed in collaboration with the Coding and Cryptography group of the Technical University of Munich (TUM).

1.1.4 Redundancy in the output data

When reading a DNA molecule, DNA sequencing outputs thousands of copies of the same sequence, with different noise realizations. Therefore, DNA sequencing generates a high amount of data, which contain a large amount of redundancy.

This is why in Chapter 6, we investigate efficient data deduplication algorithms [28] so as to remove the redundant data and reduce the storage space.

1.2 The DnarXiv project

This thesis was apart of the DnarXiv project [29], funded by the Labex Cominlabs. Note that the DnarXiv project started one year after the beginning of this thesis. This project aims to tackle DNA data storage at the interaction of several research fields: biotechnology, security, and coding:

— In the field of **biotechnology**, the project aims to investigate different techniques to improve DNA synthesis and sequencing. It also aims to identify the various constraints to consider so as to adapt and improve the security and coding parts. Finally, it will allow to test the developed coding and security techniques under *in vitro* experiments.

— In the field of **security**, the project aims to identify possible security threats in the DNA data storage workflow. It also aims to develop novel DNA-based security techniques, which would rely on the DNA structure. Until now, research on this axis was carried independently on our work on coding. However, our channel simulator was used to carry numerical experiments on the security aspects.

— In the field of **coding**, to which this thesis belongs, the project aims to explore different solutions to unconventional errors introduced by DNA data storage.

One of the objectives of the project is also to develop a **platform** to numerically simulate all together the different parts of the DnarXiv project. Our statistical channel model was integrated in this plateform, for use by the different parts of the DnarXiv project. Furthermore, to take into account the interactions between the different part of the project, the error-correction solutions proposed in this thesis were also tested on this plateform.

Chapter 2

DNA DATA STORAGE WORKFLOW

In this chapter, we describe the DNA data storage workflow. We first describe DNA and some of its biological principles. We then explain how DNA can be written through synthesis, and how DNA can be read through sequencing.

2.1 DNA

The DNA (DeoxyriboNucleic Acid) [30, 31, 32] is a long molecule which serves as support for the genetic information that is necessary for the development and functioning of all living organisms. It is therefore a natural dense memory [33].

A DNA molecule is made-up from two attached strands that form the famous double helix structure [34], see Figure 2.1. Each DNA strand contains a succession of nucleotides attached to the strand backbone. A nucleotide is composed of a deoxyribose sugars, a phosphate groups, and a single nitrogen which contains a base [35]. The order of the nucleotides on the DNA strand is critical since it allows to encode different functions.

The bases are represented with letters that form the genetic code. There exists two types of bases, which are the purine (two-carbone nitrogen ring) bases A (Adenine) and G (Guanine), and the pyrimidine (one-carbone nitrogen ring) bases C (Cytosine) and T (Thymine). The strands of the double helix are connected by hydrogen bonds between the bases. Furthermore, strands of double strand DNA (dsDNA) are complementary in the sense that a base A in one strand is always bonded to a base T in the other strand (and *vice versa*), and a base C in one strand is always bonded to a base G in the other strand (and *vice versa*).

Furthermore, the phosphate groups bond together the strand nucleotides between the 5´ (five prime) carbon and 3´ (three prime) carbon of adjacent deoxyribose sugar molecules [32]. The numbers 5 and 3 refer to the number of the carbon atom in a deoxyribose sugar, and where the carbons are clockwise numbered starting from the oxygen atom. Thanks to its 5´ end and 3´ end, the DNA molecule has by convention a reading

direction that goes from the 5´ end towards 3´ end.

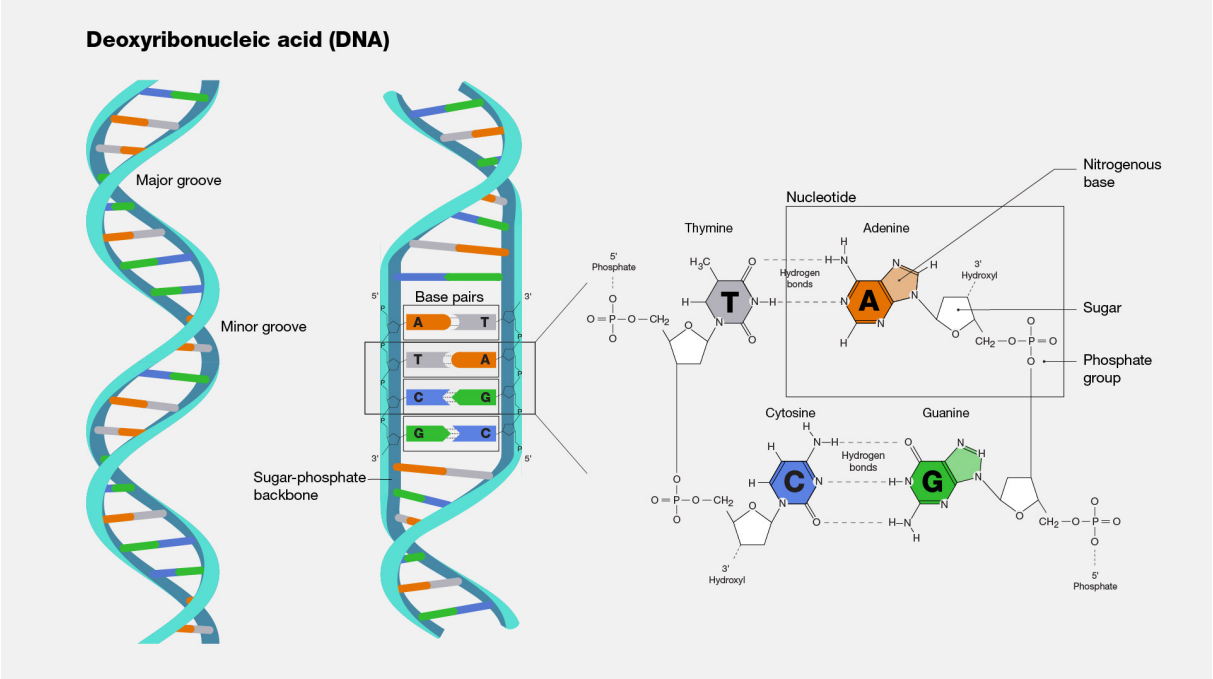


Figure 2.1: Deoxyribonucleic acid representation. The right part of the figure shows the 3 components that forms a nucleotide, how nucleotides A on its 5´ carbon and G on its 3´ carbon from same strand bond together thanks to the phosphate group; and the hydrogen bonds which is attaching nucleotides from different strands T with A and C with G together. The left part shows the backbone of a DNA helix (dsDNA) with its nucleotides attached together on and between the strands. Major and minor grooves represent the parts of the dsDNA where backbones are respectively far or close from each other.
Source: National Human Genome Research Institut (genome.gov).

As mentioned before, DNA exists in all living organisms. It is an important piece that explains how simple cells evolve into complex organisms such as human beings. For instance, understanding how DNA works can help to prevent, detect and fight against various diseases. Thus, we have invented sequencing techniques to read DNA, and synthesis techniques to build DNA. Since these two techniques are at the basis of DNA data storage, we now describe them into details.

2.2 DNA synthesis

DNA synthesis [36] is a process that builds single strands DNA (ssDNA) or dsDNA molecules by linking nucleotides. There exists several techniques for DNA synthesis, which

can be divided into two categories : natural synthesis (*in vivo*), and artificial synthesis (*in vitro*). For instance:

— **DNA replication** [37] in an *in vivo* DNA synthesis which relies on a polymerase enzyme to construct from one dsDNA molecule two new dsDNA molecules. This technique allows to replicate the genetic material from mother cells to daughter cells (cells division). A DNA template (ssDNA), which is obtained after separating the strands from dsDNA, is necessary to perform the synthesis.

— **Polymerase Chain Reaction** (PCR) [38] is an *in vitro* DNA synthesis that allows to amplify (replicate) one or several DNA molecules. This allows to make a focus on a particular DNA molecule for diagnosis purpose (for instance, look for COVID DNA material), or to implement a random memory access on a DNA data storage system [39], etc. PCR also relies on a DNA template to perform the synthesis.

— **Gene synthesis** [16] is an *in vitro de novo* DNA synthesis that aims to construct from scratch a DNA sequence and bypass the necessity to use a DNA template.

Gene synthesis is employed for DNA data storage [40] since it allows to construct almost any desired sequence. In this case, the synthesis constructs short synthetic ssDNA sequences called oligonucleotides, or oligos [16, 41]. Gene synthesis is subject to different constraints depending on the considering gene synthesis method:

— **Chemical synthesis** [36, 16, 42] is performed in parallel to synthesize from one hundred (column-based method) to ten thousands (microarray-based method) different oligos at a time [43]. The length of each oligo is in between 10 to 300 nucleotides. Both methods rely on the phosphoramidite approach, which is based on a four-step cyclic reaction [42] that adds a nucleotide by cycle to obtain the final oligos. Synthesizing oligos longer than 300 nucleotides tends to increase the amount of errors [21]. Thus, to obtain longer sequences, synthesis is usually performed in two steps. In the first step, the sequence is segmented, and each segment is synthesized as a unique oligo as described previously. In the second step, the oligos are assembled to form a longer sequence (the ordered sequence). In 2014, the cost for a column-based synthesis was between 0.05 and 0.15$ per base, and the cost for a microarray-based synthesis was between 0.00001 to 0.001$ per base [42]. Nowadays, the chemical synthesis remains the most common synthesis employed for DNA data storage [40].

— **Enzymatic-synthesis** relies on particular enzymes that are used to construct *de novo* oligos by adding nucleotides with the help of an enzyme, in a template independent manner (oppositely to PCR). This technique is still at its infancy, but it is expected to decrease synthesis costs and time, allow for longer oligos synthesis, while being free of hazardous chemicals [16, 40, 44].

Due to its costs and because it is time-consuming, the synthesis process remains the major bottleneck of DNA data storage. In the field of biology, the development of large-scale, low-cost, and highly-reliable synthesise techniques could catalyze rapid progress on this field [16].

2.3 DNA sequencing

DNA sequencing [45, 46] consists of determining the sequence of bases in a DNA molecule. The human genome (6.10^9 bases) was sequenced for the first time using the so-called Sanger sequencing, developed by Frederick Sanger in 1977 [45]. Although Sanger sequencing was considered as the gold standard of sequencing during many decades [46], the necessity for cheaper and faster sequencing led to the development of a new generation of sequencing techniques called Next-Generation Sequencing (NGS) [47, 14]. NGS offers a massively parallel sequencing with a high throughput, and drastically reduced costs, see Figure 2.2. Therefore, NGS opened the door for a genome sequencing boom [48]. As a matter of facts, to sequence the human genome assuming a seven times coverage (number of same sequences), a unique Sanger sequencer, would take 100 years when a unique NGS sequencer would take only two days.

Sanger sequencing allows to read long molecules with high accuracy [49] but suffers from its low-throughput which makes genome sequencing highly time-consuming. In the other hand, NGS sequencers offer raw read accuracy of 99.9% and has a high-throughput [50], but it can only sequence short reads. This makes genome assembly challenging, for instance because long repetitive regions of some genome sequences introduce confusion regarding where a sequenced fragment really belongs. To tackle this issue, a Third-Generation Sequencing (TGS) [51] was introduced. TGS can sequence long DNA molecules that could contain up to 2 Mbp (Mega base pair: 2 million of paired bases) at high-throughput [22, 52, 51]. Several TGS sequencers were developed by different companies such as (in an alphabetical order) Illumina, Oxford Nanopore Technologies (ONT), Pacific BioScience (PacBio), etc.

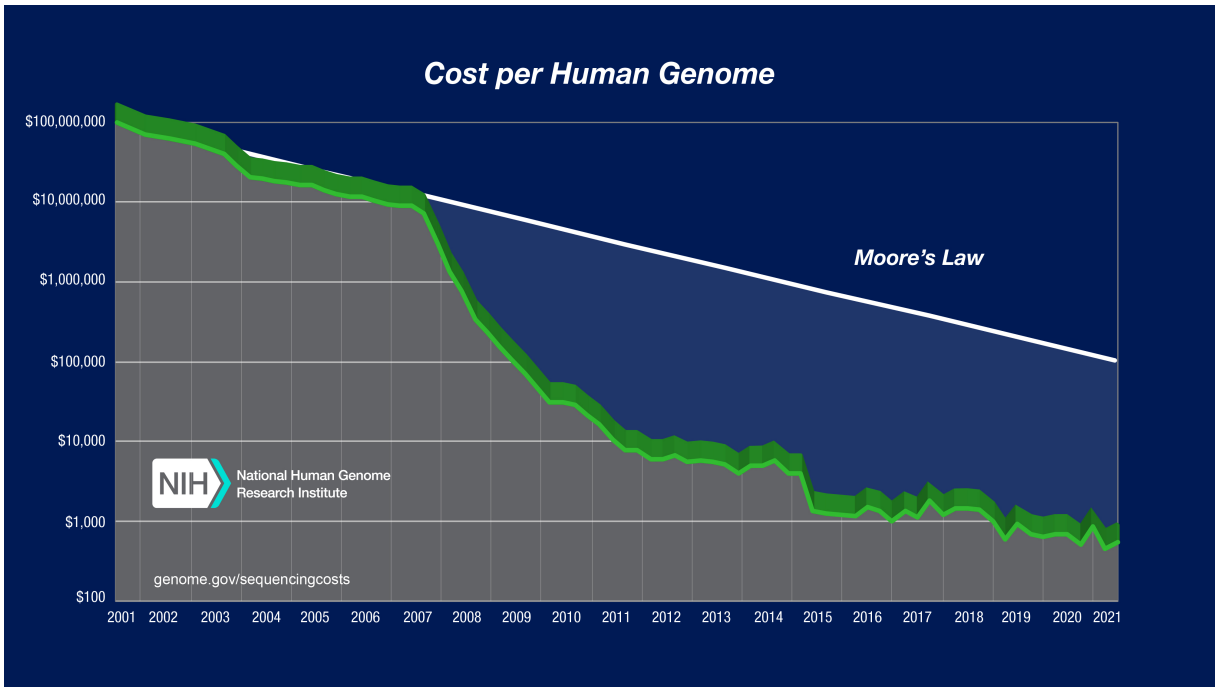


Figure 2.2: Sequencing costs of a human genome through the years and compared to Moore's Law. The plot shows how human genome sequencing costs drastically decreased from hundred millions to less than a thousands of dollars. Note that the impressive decrease that starts from 2007 is related to the transition from Sanger to next-generation sequencing.
Source: National Human Genome Research Institut (genome.gov).

In this work we mainly focus on the ONT MinION sequencer [15, 22]. The ONT MinION is a portable TGS sequencer which can sequence long DNA molecules at a high-throughput. In addition to having a small size (slightly bigger than a USB-key), it provides results in real-time in the sense that sequencing results are accessible before the whole sequencing is completed [15, 53, 22]. Furthermore, when NGS sequencers cost hundreds of thousands of dollars and a few thousands dollars for its reagents (substances to perform the sequencing) [54], the ONT MinION costs 1000$ with a few hundred dollars for its reagents. Thus ONT MinION is affordable even for small laboratories and individuals. These lasts features make it a good candidate for DNA data storage [55].

The ONT MinION works with nanoscale protein pores (nanopores) [22, 56, 57] which serve as biosensors. Nanopores are fixed on an electrically resistant polymer membrane which is immersed in an electrolytic solution. A constant voltage is applied to the solution to produce a ionic current through the nanopore. ssDNA molecules, being negatively charged, are naturally driven through the nanopore nucleotide by nucleotide, from the negatively charged part of the membrane to its positively charged side. A motor protein (called helicase) fixed on an adapter (oligo) added to a dsDNA molecules, separates strands and controls the speed of the ssDNA through the nanopore. The electrical current intensity inside the nanopore (reading region) changes depending on the 6-mer (group of 6 nucleotides) that is inside of it. Therefore the current intensity allows to identify 6-mers. The ONT MinIon sequencing is performed through 3 main steps [15, 57, 56]:

1. **Library preparation**: The DNA is prepared to meet some requirements and to eventually achieve some goals. The most rapid library preparation takes 5-10 minutes, where adapters are ligated to dsDNA and finally a motor protein is added to each dsDNA.
2. **Sequencing**: The dsDNA is driven through the nanopore nucleotide by nucleotide, and a current signal related to the group of 6-mer that is on the nanopore is recorded, see Figure 2.3.
3. **Data processing**: To decide which bases were read, a so-called basecalling step is first performed over the sequencing results using a basecaller software. The basecaller translates the electrical current signals of the successive 6-mers into a sequence of bases A, C, G, T. The resulted sequences can be assembled, aligned, or mapped with more efficiency than for the NGS because the fragments are longer. To illustrate this, we can imagine a situation where we have to put together pieces of a puzzle, and having bigger pieces (long fragments) certainly helps to assemble the

puzzle pieces quickly and to put them in the right locations.

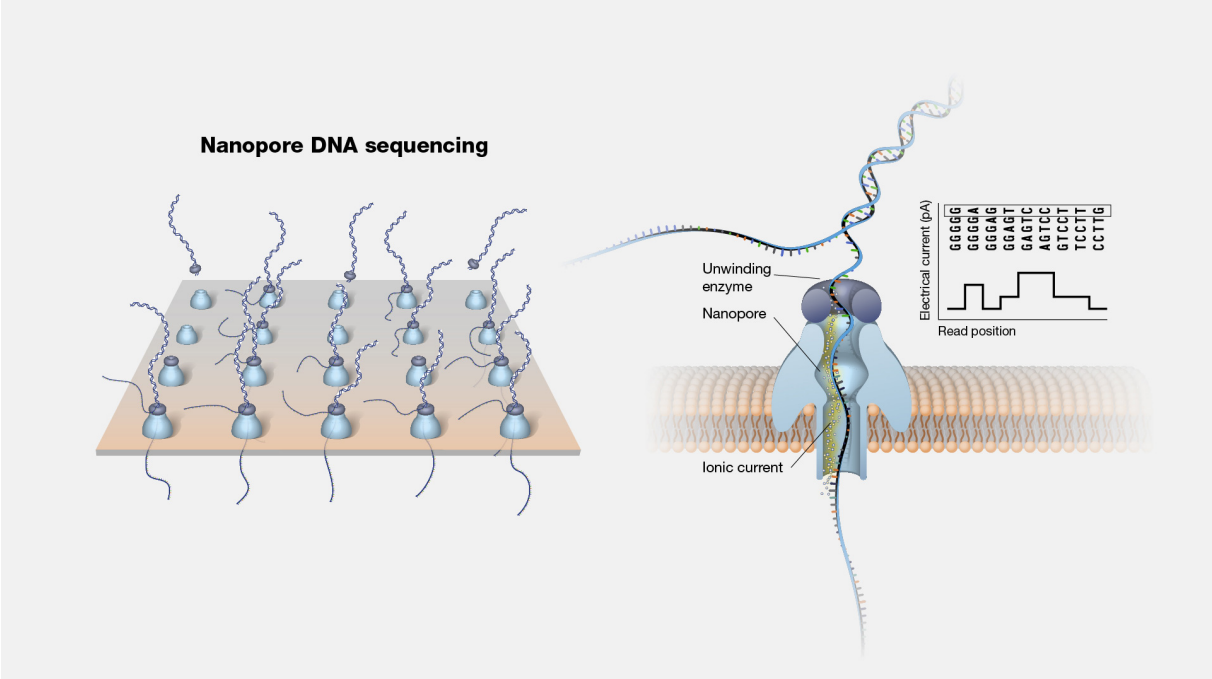


Figure 2.3: Nanopore DNA sequencing illustration. The left part represents dsDNA molecules and their motors protein (Unwinding enzyme) attracted and driven through nanopores that are embedded in a polymer membrane. The right part shows a dsDNA unwinded (separated) by the helicas of the unwinding enzyme. One strand goes through the nanopore when the other one will be captured by another nanopore. The right part also shows the basecalling step where the electrical signal is translated into 5-mers (only for illustration).
Source: National Human Genome Research Institut (genome.gov).

However, despite some successive improvements, the ONT MinION raw reads accuracy is about 94% [22]. Errors are due to different factors including the sequencing of long homopolymers which are long repetition of the same base (e.g.: AAAA is an homopolymer of length 4) [19]. Therefore, to build reliable DNA data storage systems, it is necessary to implement some mechanisms to retrieve the original data. One solution, which we will explore in this thesis, is to rely on error-correction codes so as to eliminate sequencing errors. However, one main issue for the design and evaluation of powerful error-correction solutions resides in the fact that synthesis and sequencing remain costly and time-consuming operations. Therefore, a first step would consist of numerically simulating the whole process, including synthesis and sequencing. This is why the next chapter introduces a statistical model to represent the DNA data storage channel, from the input of the synthesis to the output of the basecaller.

Chapter 3

DNA DATA STORAGE CHANNEL MODEL

In the previous chapter, we described the DNA data storage workflow. In particular, we described how DNA can be written using DNA synthesis, and read using DNA sequencing. We then emphasized the fact that errors are introduced during DNA synthesis and sequencing. Though, DNA sequencing with the MinION is the main contributors to these errors. This is why in this chapter we introduce a novel DNA data storage channel model that aims to model the whole DNA data storage process. As mentioned in Chapter 2, in this work we consider the ONT MinION sequencer. Thus, our channel model is inspired from how the MinION sequencer works, and takes into account the dependency between the read k-mers and the observed errors, and the dependency between successive errors. In addition, our channel model depends on some parameters which are set after a training phase. In our work, we used two different types of data to train our model, a set of experimental data which came through the full DNA data storage process, and a set of genomic data which come from the sequencing of a Streptococcus thermophilus bacteria. Each set has its advantages and drawbacks, which will be discussed in this chapter. In both cases, we evaluated the performance of our channel model through two different approaches: edit maps and Kullback Leibler (KL) divergence.

Thus, in this chapter, we introduce our DNA data storage channel model. We review existing DNA data storage channel models. We then introduce our novel statistical DNA data storage channel model. Finally, we compare channel models through numerical evaluation.

3.1 Channel modelling

We now describe all the physical processes involved in writing and reading data in DNA data storage [5, 6, 8, 58, 13], see Figure 3.1. The channel models the whole process from synthesis to sequencing, including basecalling. Preprocessing and postprocessing steps are not covered by the channel, and they will be described in more details in the

next chapters. In what follows, the notation $[\![1, V]\!]$ represents the set of integers between 1 and V.

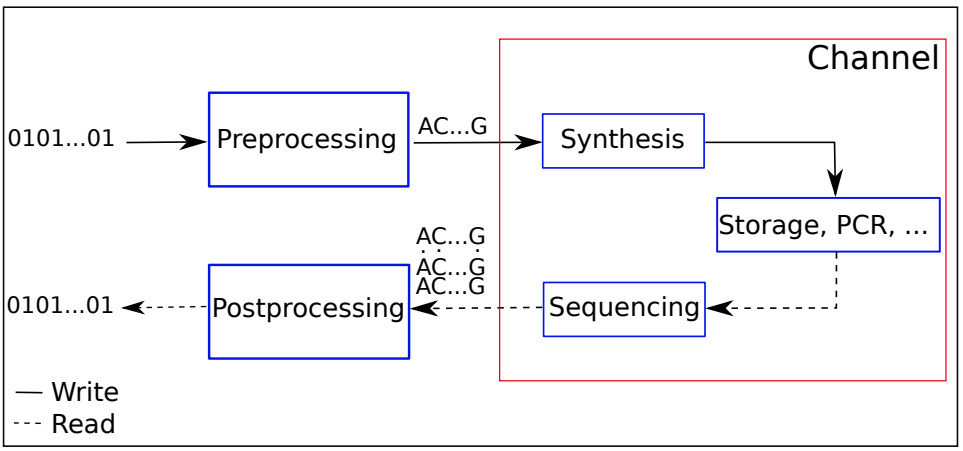


Figure 3.1: This figure represents the workflow to write and read data on a DNA data storage system. A preprocessing step is usually performed to prepare and convert a binary file into bases A, C, G, T. The channel models the whole process from synthesis to sequencing (including basecalling). The channel outputs several copies of the same sequence. The output sequences are then postprocessed to rerieve the original data.

3.1.1 Writing data (synthesis)

As described in Figure 3.1, the input binary sequence is first preprocessed and prepared for the DNA synthesis. The synthesis operation then outputs DNA strands, as described in Chapter 2. Each synthetic DNA strand is formed by successive nucleotides which take values A, C, G and T [16]. In this work, we consider chemical synthesis described in Chapter 2, in which oligonucleotides 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. It introduces a low amount of substitutions, insertions, and deletions [19, 21, 59].

Formally, in what follows, we use $\mathbf{x} = (x_1, \cdots, x_N)$ to denote the sequence of digits to be synthesized, where each digit x_t takes values in a quaternary alphabet, such that $x_t \in \{A, C, G, T\}$.

3.1.2 Reading data (sequencing)

For reading data, the stored DNA strands are sequenced. This produces digital signals which are postprocessed to retrieve the original data. In this work, we consider the ONT MinION sequencer described in Chapter 2.

In a formal way to describe how the nanopore sequencer works, assume that the sequence of nucleotides on a DNA strand can be represented by a sequence of bases $(\alpha_1, \alpha_2, \cdots, \alpha_t, \cdots)$. A k-mer consists of a group of k successive nucleotides, such as

$$k\text{-mer}_t = (\alpha_{t-k+1}, \alpha_{t-k+2}, \cdots, \alpha_t) \tag{3.1}$$

that pass through the nanopore, thus producing a current level c_t. The next k-mer defined as

$$k\text{-mer}_{t+1}(\alpha_{t-k+2}, \alpha_{t-k+3}, \cdots, \alpha_{t+1}) \tag{3.2}$$

produces the current level c_{t+1}. We emphasize that each current level c_t represents k successive nucleotides, and that two successive levels c_t and c_{t+1} come from two successive k-mers with $k-1$ bases in common.

Then, the basecaller transforms the current levels into sequences of bases with values A, C, G, T. In this work, we used the Guppy basecaller [61, 60], which is based on a Deep-Learning approaches and is maintained by ONT. The basecaller outputs a FastQ file which contains the read sequences and some metadata. Each sequence of this FastQ file is called a "read". We use V to denote the number of reads (*i.e.*, the number of output sequences), and we use $\mathbf{y}^{(v)}$ to denote the reads (*i.e.*, the output sequences), where $v \in [\![1, V]\!]$, and $y_t^{(v)}$ takes values in a quaternary alphabet $\{A, C, G, T\}$. The sequence $\mathbf{y}^{(v)}$ is of length $N^{(v)}$. The $N^{(v)}$ can all take different values, and in general, $N^{(v)} \neq N$, where N is the length of the input sequence $\mathbf{x}$. This comes from the fact that the sequencer introduces not only substitutions, but also insertions and deletions in the read sequences, and error realizations vary from sequence to sequence [19, 20].

Note that by convention, DNA molecules have a reading direction that goes from the 5´ end carbon towards 3´ end carbon, see Figure 2.1. One strand is arbitrarily considered as being the forward-strand when the second one is considered as being the reverse-strand. In a reverse-strand, the sequence is reversed in the sense that the last bases become the first, and each base is turned into its complement, *i.e.*, $A \to T$, $C \to G$, $G \to C$, and $T \to A$. Thus, forward and reverse strands are complementary and anti-parallel.

From an information-theoretic perspective, all the previous steps (synthesis, sequenc-

ing, basecalling) can be modeled as a channel. As mentioned before, it is very useful to have an accurate statistical description of the channel, for the design and performance prediction of error-correction solutions, before launching costly *in vitro* experiments. We now introduce our notation and describe state-of-the-art channel models for DNA storage. We then present our proposed channel model.

3.2 Notation

In our model, $\mathbf{x}$ is the channel input sequence of length N, and $x_t \in \{A, C, G, T\}$. In addition, $\boldsymbol{k}\mathbf{mer}_t = (x_{t-k+1}, \cdots, x_{t-1}, x_t)$ is the $k-$mer of length k at position t. In this section, although the sequencer outputs V sequences $\mathbf{y}^{(v)}$, we drop the notation v and assume that the channel only outputs one sequence $\mathbf{y}$, with $y_t \in \{A, C, G, T\}$. We consider this notation simplification since we assume that the error realizations are independent from sequence $\mathbf{y}^{(v)}$ to sequence $\mathbf{y}^{(v')}$ whenever $v \neq v'$.

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 introduce a 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. The interest of considering the sequence $\mathbf{e}$ is that it is aligned with the input $\mathbf{x}$ in the sense that $\mathbf{e}$ and $\mathbf{x}$ have the same length, and that error event e_t applies onto input symbol x_t. In this sense, $e_t = \text{Del}$ means that the symbol x_t is deleted from the sequence, and $e_t = \text{Sub}$ means that the base value of x_t is replaced by another base denoted $B_t \in \{A, C, G, T\}$. In addition, $e_t = \text{Ins}$ means that one or several symbols are inserted just after x_t. In this case, we denote by L_t the length of the insertion, that is the number of symbols added after position t.

We consider that a given DNA data storage channel model is defined by a set of probability distributions for the successive e_t. For instance, $\mathbb{P}(e_t|\boldsymbol{k}\mathbf{mer}_t, e_{t-1})$ is the conditional probability of e_t with respect to $\boldsymbol{k}\mathbf{mer}_t$ and the previous event e_{t-1}. In addition, $\mathbb{P}(e_t)$ is the marginal probability of e_t.

3.3 Existing DNA data storage models

We now review existing channel models for DNA data storage.

3.3.1 i.i.d. channel model

In the literature, the sequence $\mathbf{e}$ of events is often considered to be independent and identically distributed (i.i.d.). Events e_t are further assumed to be independent from the input symbols x_t, and the probabilities $\mathbb{P}(e_t)$ are either inferred from sets of experimental data, or fixed arbitrarily for simulation purposes [62, 23, 24, 63]. However, although it can simulate the correct amount of errors, this model cannot represent bursts of errors, see Figure 3.2, since it assumes that e_t is independent from e_{i-t}. In addition, it does not take into account the effect of $\boldsymbol{k}\mathbf{mer}_t$ onto error event e_t, while we know from several other works [61] that a statistical relation between those exists.

3.3.2 Deepsimulator

Deepsimulator is a popular tool to simulate the DNA data storage channel [64, 65]. Deepsimulator relies on a Deep-Learning approach combined with a basecaller. In a first step, Deepsimulator takes as input a sequence of bases, and simulates electrical current levels which would be obtained after the sequencing, by relying on a Deep Neural Network. In a second step, the current levels are sent to a basecaller which transcripts the current levels into bases. Event sequences $\mathbf{e}$ generated by Deepsimulator contain some memory and are statistically dependent from the input sequences $\mathbf{x}$. However, after having performed a significant amount of simulations, we could observe that Deepsimulator does not reflect well the randomness of the DNA storage channel. For instance, as shown in Figure 3.2, the same type of error appears in most of the simulated sequences in the same particular position, which does not correspond to what can be experimentally observed after synthesis and sequencing. In addition, we could also observe an inaccurate predominance of substitutions and insertions over deletions. Furthermore, in Deepsimulator channel, we can observe some event positions $\mathbf{e_t}$ where there is no observed errors (there are only matches) over all the event sequences $\mathbf{e}$, even though this is never occuring on the experimental data.

3.3.3 Badread

Badread [66] is another simulator that relies on estimated transition probabilities (provides by the simulator), which are the probabilities to replace a certain k-mer by another specific sequence of length $k' \neq k$. The simulator picks one k-mers after each other at random in the input sequence, toss a coin to decide whether introducing an

error on the current k-mer, and if this is the case, uses the corresponding transition probabilities to replace the k-mer by another sequence. However, in Badread, the user has to specify many parameters such as the average reads-length, the total amount of errors, the probability of error, etc. When we have access to examples of input and output channel sequences, we can manually adjust the parameters of BadRead so as to produce simulated outputs that fit with the examples, see Figure 3.2. However, incorrect choices of parameters can lead to unsatisfying results, shown in the numerical results section.

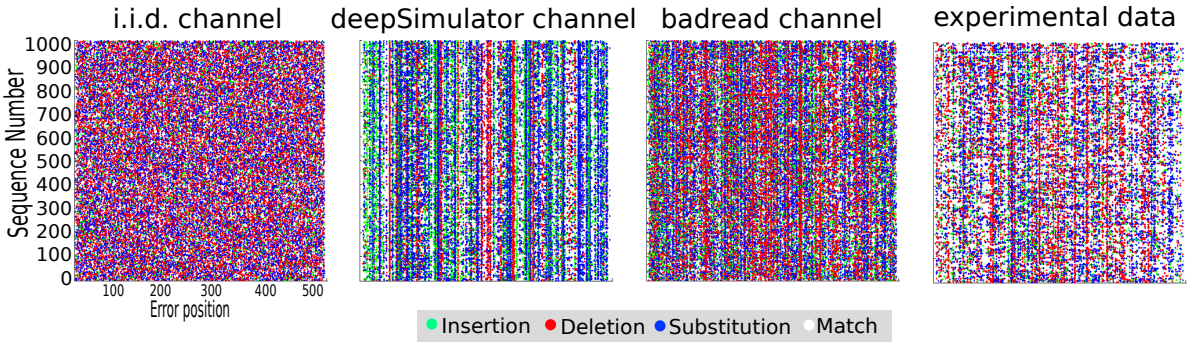


Figure 3.2: Simulations and experimental results for a given sequence. The figures represents the observed errors through the simulated and the experimental data. Each line represents one sequence $\mathbf{e}$, and each column represents the error eventse_t at position t ($t \in [\![1, 500]\!]$) in each sequence. These figures called edit maps allow to visually compare the channel model outputs to the experimental data to see if they are similar.

Depending on the application, it is usually necessary to have a channel representation which is as accurate as possible. Therefore, there is a need to develop more accurate and ready-to-use models (without parameters).

3.4 Proposed channel model with memory

We now introduce our statistical channel model for DNA data storage [67] [1], that aims to solve the drawbacks of existing models. The proposed model takes into account the statistical dependencies between the event sequence $\mathbf{e}$ and the input sequence $\mathbf{x}$, and can be seen as a Markov chain [68] with a memory of order k. Our model also considers that event e_t depends on $\boldsymbol{k}\mathbf{mer}_t$, which allows to model editions due to successive k-mer reads, according to the way the MinION sequencer works. Our model also assumes some internal memory in $\mathbf{e}$, by considering that previous event e_{t-1} can affect current event e_t.

This allows to take burst errors into account.

Our model is then described by the following conditional probability distribution:

— $\mathbb{P}(e_t|\boldsymbol{k}\mathbf{mer}_t, e_{t-1})$ characterizes the dependency between the current event $e_t \in \{\text{Ins}, \text{Del}, \text{Sub}, \text{Match}\}$, the read $\boldsymbol{k}\mathbf{mer}_t$, and the previous event e_{t-1}. This captures the error dependency with the k-mer, and allows to consider bursts of errors through the dependency to e_{t-1}.

— $\mathbb{P}(L_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Ins})$ characterizes the insertion length L_t depending on the read $\boldsymbol{k}\mathbf{mer}_t$. Note that deletions are always of length 1.

— $\mathbb{P}(B_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Sub})$ characterizes the probability to substitute the last base x_t of $\boldsymbol{k}\mathbf{mer}_t$ by the base B_t, where $x_t \neq B_t$.

We assume that the probabilities $\mathbb{P}(e_t|\boldsymbol{k}\mathbf{mer}_t, e_{t-1})$ and $\mathbb{P}(L_t|\boldsymbol{k}\mathbf{mer}_t, e_t = \text{Ins})$ do not vary with t for $k \leq t < N$. In addition, we observed from experimental data that the probability to get an error is higher at the first position $t = 1$ and at the last one $t = N$, compared with middle positions $1 < t < N$. Therefore, we allow for different probability distributions $\mathbb{P}(e_1)$, $\mathbb{P}(L_1|e_1 = \text{Ins})$, and $\mathbb{P}(e_N)$, $\mathbb{P}(L_N|e_N = \text{Ins})$, to be used when $t = 1$ and $t = N$. Finally, for $1 < t < k$, since no complete k-mer was observed already, we consider probabilities $P(e_t|x_t)$, $P(B|x_t, e_t = \text{Sub})$ and $P(L|x_t, e_t = \text{Ins})$ that only depend on the input value x_t.

The next step consists of learning all the probability terms from some sets of data, so as to build the simulator. In this work, we trained the model over two different sets of data with different characteristics: one set of experimental data, and one set of genomic data. Each of the two sets of data has its own advantages and drawbacks, as will be described in the following.

3.4.1 Training on a set of experimental data

Experimental data

At the beginning of the DnarXiv project, in order to obtain experimental data, we performed the synthesis of nine sequences. In this set, 8 over 9 of the sequences were obtained from text files extracted from the "Little Red Riding Hood" story, and the "Universal Declaration of Human Rights". The last sequence was constructed by introducing plenty of long-homopolymers so as to track the errors that they introduce during the

1. The Memory Channel Model for DNA Data Storage "DNArSim" software is available on GitHub: https://github.com/BHam-1/DNArSim

sequencing [19, 20]. In addition, 6 sequences had length 500 nucleotides, and 3 sequences had length 1000 nucleotides. The ordered sequences were built by ourselves to meet different requirements and synthesis constraints, and the DNA synthesis was ordered from *Thermo Fisher* company, see Figure 3.3. Thousands of copies of each of the nine ordered sequences have been synthesized. The resulting DNA strands, were then sequenced by Emeline Roux at INRA STLO Rennes institute, see Figure 3.3. We then analyzed and postprocessed the results. Thus, we can consider that the obtained experimental data went through the whole DNA data storage channel.

Figure 3.3: The right part of the figure represents a test tube which contains thousands of copies of an ordered synthetic DNA sequence. The left part represents a flowcell which is a consumable that is used on the MinION sequencer to sequence DNA. The experiments were performed at INRAE STLO Rennes.

Training

To train our model, *i.e.*, to estimate all the probability terms, we first used the set of experimental data described previously.

This set was generated from $U = 9$ input sequences $\mathbf{x}^{(u)}$ ($u \in [\![1, U]\!]$), called the reference sequences. We had access to FAST5 files (current levels) obtained after sequencing of the data, and then applied the Guppy basecaller. We used the "Super Accuracy" mode of Guppy, which is three times slower than the "High Accuracy" mode, but offers a more

reliable basecalling. We also used the default "Q10" quality score (QScore) of Guppy, which is based on the basecalling quality score that is provided on the FastQ files. The Q10 parameter allows to filter the output sequences, so as to throw those which have more than 10% of errors.

The basecalling of each reference sequence $\mathbf{x}^{(u)}$ ($u \in [\![1, U]\!]$), provided V_u output sequences $\mathbf{y}^{(u,v)}$ ($u \in [\![1, U]\!]$, $v \in [\![1, V_u]\!]$), for a total of $V = \sum_{u=1}^{9} V_u = 34604$ output sequences obtained after sequencing. The output sequences $\mathbf{y}^{(u,v)}$ correspond to either forward or reverse strands of $\mathbf{x}^{(u)}$. For clarity, we denote the reverse strands by $\bar{\mathbf{y}}^{(u,v)}$ instead of $\mathbf{y}^{(u,v)}$, and we denote by $\bar{\mathbf{x}}^{(u)}$ the reversed version of $\mathbf{x}^{(u)}$. We refer to the dataset $(\mathbf{x}^{(u)}, \bar{\mathbf{x}}^{(u)}, \mathbf{y}^{(u,v)}, \bar{\mathbf{y}}^{(u,v)})$ as *SetE*.

In order to do the training, as a first step, each read sequence $\mathbf{y}^{(u,v)}$ was aligned with its reference sequence $\mathbf{x}^{(u)}$. The alignments were done using the ggsearch36 tool from the FASTA software [69]. This tool allows to do global-to-global alignments, that is to say that the whole read (from its first to its last base) is aligned against the whole reference. For each pair $(\mathbf{x}^{(u)}, \mathbf{y}^{(u,v)})$, this first step provided one of the following cases:

— *Aligned-read*: the read was aligned against the reference.

— *Unaligned-read*: either the read cannot be aligned, or its alignment score is lower than the minimum threshold.

— *Reverse-read alignment*: the read cannot be aligned against the reference itself but can be aligned against $\bar{\mathbf{x}}^{(u)}$. This behavior is due to the fact that the read represents the reverse-strand of the DNA helix.

Note that after the alignment, some sequence can have more than 10% of errors, since the Q10 parameter of Guppy is based on the basecalling quality and not on the alignment quality.

In a second step, we used the aligned-reads to compute conditional probabilities involving each k-mer contained in the sequences $\mathbf{x}^{(u)}$, and the reverse-reads to compute conditional probabilities involving each k-mer contained in the sequences $\bar{\mathbf{x}}^{(u)}$. For instance, we estimated the conditional probabilities $\mathbb{P}(e_t = E | \boldsymbol{k}\mathbf{mer}_t, e_{t-1} = E')$ by counting over all the aligned-read pairs $(\mathbf{x}^{(u)}, \mathbf{y}^{(u,v)})$ and all the reverse-read pairs $(\bar{\mathbf{x}}^{(u)}, \bar{\mathbf{y}}^{(u,v)})$ the number of outcomes of each event $E \in \{\text{Ins}, \text{Del}, \text{Sub}, \text{Match}\}$, divided by the number of outcomes of the considered $(\boldsymbol{k}\mathbf{mer}_t, e_{t-1} = E')$. The other probability terms were estimated following the same way.

Advantages and drawbacks

Training on the set of experimental data has two advantages. The first advantage is that the reference sequences $\mathbf{x}^{(u)}$ are perfectly known, which makes the comparisons between the reads and the reference fully reliable. The second advantage is that since each read corresponds to the whole reference sequence, it makes global-to-global alignments possible. On the other hand, the main drawback of this set is that it contains only a small amount of data, due to high DNA synthesis costs. This insufficiency of data led to plenty of unobserved pairs $(\boldsymbol{k}\mathbf{mer}_t, e_{t-1} = E)$, since there are $4^k \times 4$ different such combinations. 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. In addition, the dataset was biased since one of the 9 reference sequences contained a lot of long-homopolymers, which are known to introduce a lot of synthesis and sequencing errors [19]. As a result, the overall error probability over this dataset is high, about 10%. However, it was useful to learn the model over this first dataset, both to validate the approach (since the reference sequences are perfectly known), and also to have an example of DNA storage channel with a large amount of errors, which will be useful in our simulations.

3.4.2 Training on a set of genomic data

Unfortunately, open access experimental datasets are rare and insufficient in terms of size. This is why in the previous section, we only trained our model onto our own, but small, set of experimental data. Oppositely, datasets of genomic data are way more accessible through different genomic databases such as ENA Browser [70], GenBank [71], etc.

Genomic data

A genome [72] is a DNA sequence that contains the whole DNA material related to a particular living organism. It allows to define and identify living organisms, make comparisons with other genomes from same species to identify variations that could be responsible of diseases, etc. The complete genome of a living organism is usually built by assembling fragments coming from parallel (or separate) sequencing, using *de novo* genome assembly algorithms [73]. Assembly is computationally complex, since it relies on the short overlaps that exist between the read fragments to construct a contigous

sequence. For instance, NGS sequencing of the human genome can produce two to three billion reads, with hundred copies of each. In addition, some reads are similar but belong to different parts of the genome.

Depending on the considered specie, genome length is different and can contains from a hundred up to 10^{12} nucleotides [74]. Thanks to NGS and TGS, hundred of thousands genomes were sequenced with different sequencers (illumina, ONT MinION, ...) and are publicly available in online genomic databases.

Training

In this work [75], we considered a set of genomic data that contains $U = 7$ strains (subtypes) [76] of the *Streptococcus thermophilus bacteria* [77], which provides $U = 7$ reference sequences $\mathbf{x}^{(u)}$. We had access to FAST5 files (current levels) obtained after sequencing of the data, and then applied the Guppy basecaller with the same parameters and mode than for the experimental data training. The basecalling provided V_u read sequences $\mathbf{y}^{(u,v)}$, for each $u \in [\![1, U]\!]$. These read sequences contain forward and reverse strands, as in the first dataset described in section 3.4.1. We refer to the dataset $(\mathbf{x}^{(u)}, \bar{\mathbf{x}}^{(u)}, \mathbf{y}^{(u,v)}, \bar{\mathbf{y}}^{(u,v)})$ as *SetG*.

However, training the model using genomic data is different from training over *SetE*. Especially, when in Section 3.4.1, both references and read sequences barely reach 10^3 bases, in the genomics case, the reference sequences contain about 10^6 bases, and the read sequences vary between 10^3 and 10^5 bases. Both the amount of bases and the difference in length between reference and reads make it impossible to use global-to-global alignments. Therefore, for this dataset, we consider global-to-local alignments. Global-to-local alignments aims to align a whole read (from its first to its last base) against a particular part of the reference. To perform this type of alignment we use minimap2 aligner [79, 78], which is very efficient at performing global-to-local alignments of very long sequences in a reasonable time.

When using minimap2, we obtain different type of alignments:

— *Primary alignment*: the read is aligned against only one location of the genome and can have few unaligned nucleotides on its extremities.

— *Multiple possible alignment*: The read is aligned against multiple parts of the genome. It then has several secondary alignments and a unique primary alignment which has the best alignment score.

— *Chimeric alignment*: the read is aligned against one or multiple parts of the genome, but some of its contents cannot be aligned. The read itself is called chimeric read.

— *Unaligned read*: none of the read can be aligned against the genome.

To obtain an accurate model after the training, we did not consider secondary alignments, nor chimeric alignment. We then selected only primary-alignments, including those obtained inside multiple possible alignments. In addition, we considered only reads that have a length larger than 5000 nucleotides. The length constraint helps to avoid confusing alignments, where a short read can be aligned to similar but different parts of the reference. Also, long homopolymer regions, where the homopolymer length is more than seven nucleotides are ignored during the training. The long homopolymers constraint helps to prevent the case where after the alignment, an error can be put anywhere on a long homopolymer, and the alignment score remains the same. Thus, using the long homopolymer constraint, errors are affected to the correct k-mers.

We then used the same process as in section 3.4.1 in order to estimate all the probability terms.

Advantages and drawbacks

As a main advantage, genomic data provide a much larger amount of data. All possible combinations ($\boldsymbol{k}\mathbf{mer}_t, e_{t-1} = E$) were observed, and because of the large amount and diversity in the data, there is a good balance between all k-mers. However, with genomic data, the effect of DNA synthesis is not taken into account by the channel model. We assume that this does not affect the structure of our channel model, because unlike MinION sequencing, chemical synthesis introduces a very small amount of errors, although this will certainly affect the amount of errors learned by the model. Furthermore, since we are considering only primary-alignments to perform the training, the channel model may introduce another type of bias. In fact, the training discards chimeric reads and low-scoring alignments, *i.e.*, sequences that have a high amount of errors. As a results, the overall error probability over this dataset is about 3%. The amount of error is significantly smaller than on *SetE* training because of two main factors. The first factor is related to the read selection as mentioned previously. The second factor is due to the large amount of data that is on *SetG*, which allows to remove the biases that have been introduced by the homopolymers as on the *SetE* training.

3.4.3 Channel simulator

Once we get all the probabilities from training over one or the other set, we can build a simulator that takes as input a given sequence and generates random output sequences according to the channel model. This model has three main advantages:

1. In case of technology evolution (synthesis, sequencing, basecalling,..), the model can be retrained with new sets of experimental or genomic data.
2. Our model also takes into account the basecaller, thus, it is faster than Deepsimulator which requires to run the basecaller during simulations.
3. As opposed to a black-box approach, all our probabilities terms are explicit. The values of these probabilities can then be used, either to better understand how errors are introduced during the DNA storage process, or to incorporate them into channel code construction and decoders, as will be done in the next chapters.

We now compare the memory channel model trained over *SetE* and *SetG* to the existing channel models.

3.5 Performance evaluation

In this section, we compare our channel model with existing ones, onto our available experimental data. We use two approaches for performance comparison. The first one is based on a qualitative comparison using edit maps. The second one is based on a quantitative comparison and relies on the Kullback-Leibler (KL) divergence. Furthermore, we consider two scenarios.

— **Scenario 1**: The channel models were all trained on *SetE*, and they where simulated by taking as inputs some sequences of *SetE*. Thus, the channel models have also learned from the sequences to simulate.

— **Scenario 2**: The channel models were not trained on *SetE* but they use their own knowledge (for instance, Badread and Deepsimulator were taken as they are from the online code). In this second scenario, our memory channel was trained on *SetG*. Channel models simulations take as input some sequences of *SetE*.

3.5.1 Edit maps

Edit maps are scatter plots which represent 1000 event sequences $\mathbf{e}$ for different runs of channel simulation. Each line represents one sequence $\mathbf{e}$, and each column represents the error events e_t at position t ($t \in [\![1, 500]\!]$) in each sequence. We generated edit maps for input sequences $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(3)}$ (from *SetE*). A channel model is considered approaching the experimental data when their edit maps are visually close. Visually close means that they have close error patterns, close amount of errors, close type of errors.

We now present the obtained edit maps for four different setups. The first two setups are both taking sequence $\mathbf{x}^{(2)}$ as input, and then respectively run the scenario 1 for the first setup, and the scenario 2 for the second setup. Since it contains a lot of long homopolymers, the sequence $\mathbf{x}^{(2)}$ is somehow a worst case approach. The last two setups take as input $\mathbf{x}^{(3)}$, and then respectively run the scenario 1 for the first setup, and the scenario 2 for the second one. In each setup, edit maps are obtained for different models, by generating 1000 sequences at random from the model. Edit maps for experimental data are also represented for comparison purpose, by taking the sequences as they are. Regarding Badread, we set the fragment length parameter as equal to the input sequence $\mathbf{x}^{(u)}$ length.

Figure 3.4 shows edit maps for the first setup, where the input sequence is $\mathbf{x}^{(2)}$ and the first scenario is considered. We observe that for the i.i.d. model, errors are incorrectly uniformly distributed over all the sequence, while for Badread channel, the type and amount of errors is different from the experimental data, in the sense that compared to the experimental channel, Badread outputs a very small amount of deletions and a high amount of insertions and substitutions. Finally, the memory channel model seems to be the closest one to the experimental data. The edit map of Deepsimulator channel is not represented because its basecalling step did not output sequences. The basecalling step do not output a sequence when its quality score (Q10 by default) is not sufficient, *i.e.* it contains more than 10% of errors.

Figure 3.5 shows the edit maps for the second setup, where the input sequence is $\mathbf{x}^{(2)}$ and the second scenario is considered. The memory channel model was trained on *SetG*. Although they reproduce some experimental error patterns, the amount of errors introduced by both channels is different from the experimental data. Badread seems to introduce too much errors, when our memory channel introduces too few errors. In both case, relying on edit maps alone, it is hard to say which one is closer to the experimental data. Furthermore, the i.i.d. edit map is not represented because the errors remain

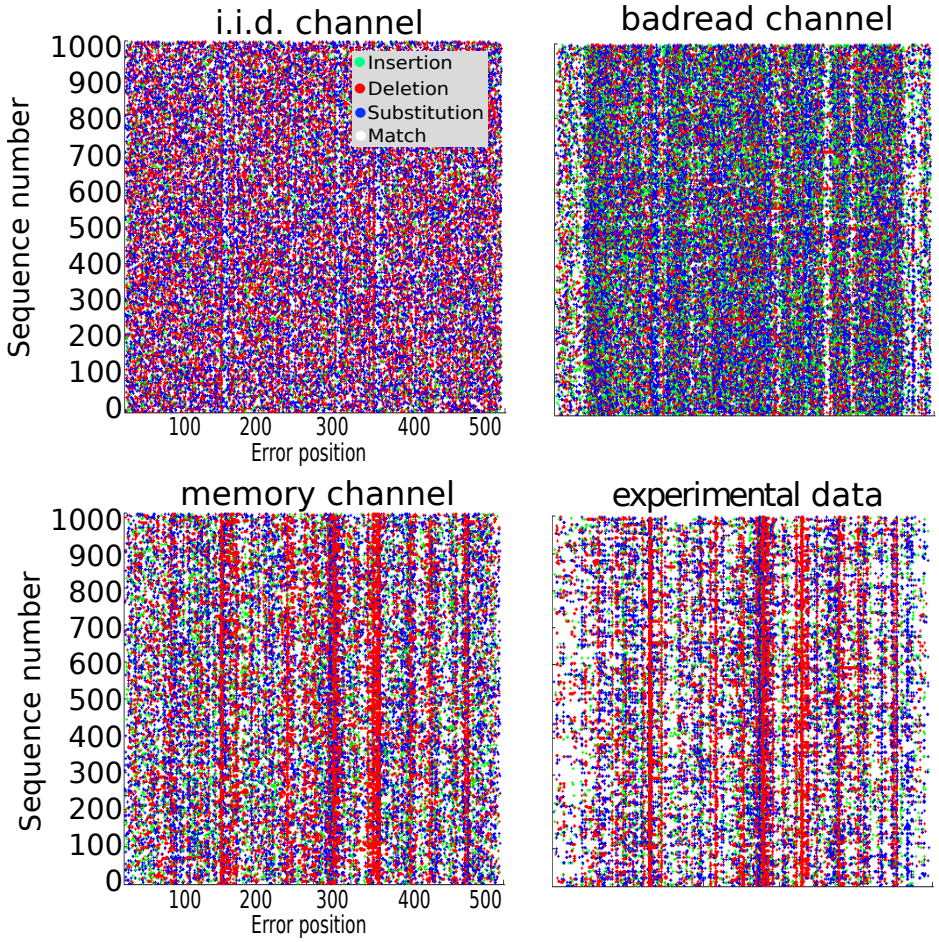


Figure 3.4: Edit maps of simulations and experimental results for $\mathbf{x}^{(2)}$. The channel models were all trained on *SetE*. The edit map of deepsimulator channel is not represented because its basecalling step did not output sequences. The basecalling step do not output a sequence when its quality score (Q10 by default) is not sufficient, *i.e.*, it contains more than 10% of errors.

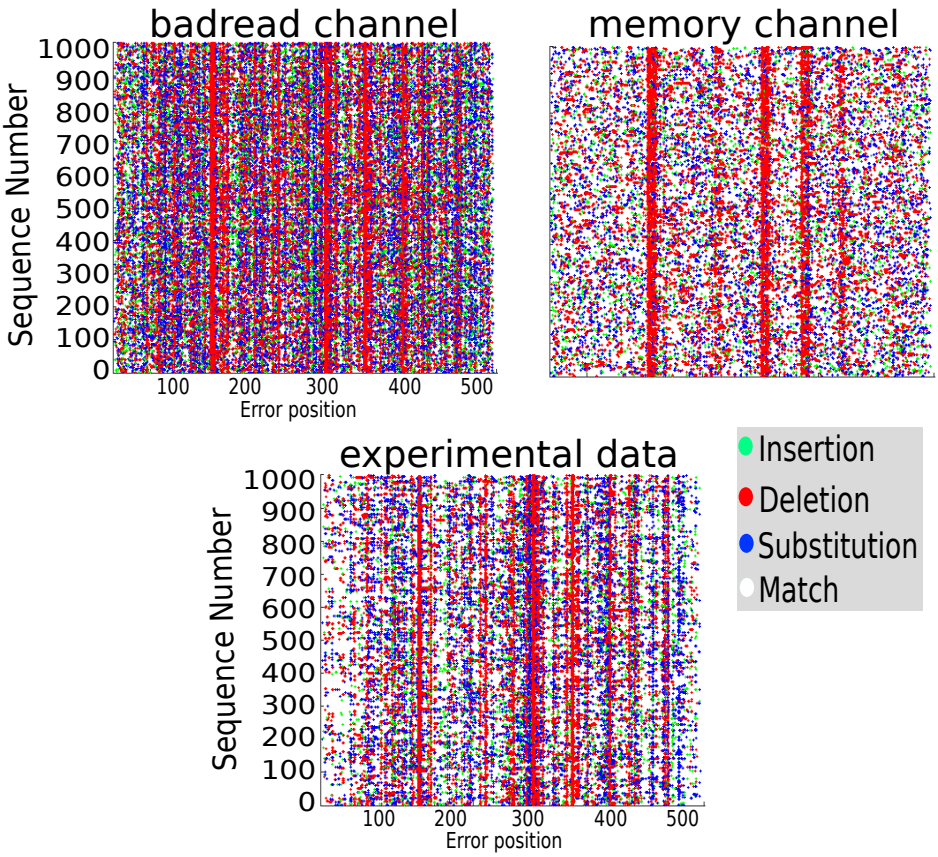


Figure 3.5: Edit maps of simulations and experimental results for $\mathbf{x}^{(2)}$. The channel models are using their default training, while the memory channel model was trained on *SetG*. deepsimulator has the same issue *i.e.*, no output, thus, it is not represented. The i.i.d. is not represented because the error rate are the same than previously, thus, its edit map will remain practically the same.

uniformly distributed, thus, the same previous conclusion can be drawn. In the case of Deepsimulator channel, the same issue related to no output during the basecalling persists.

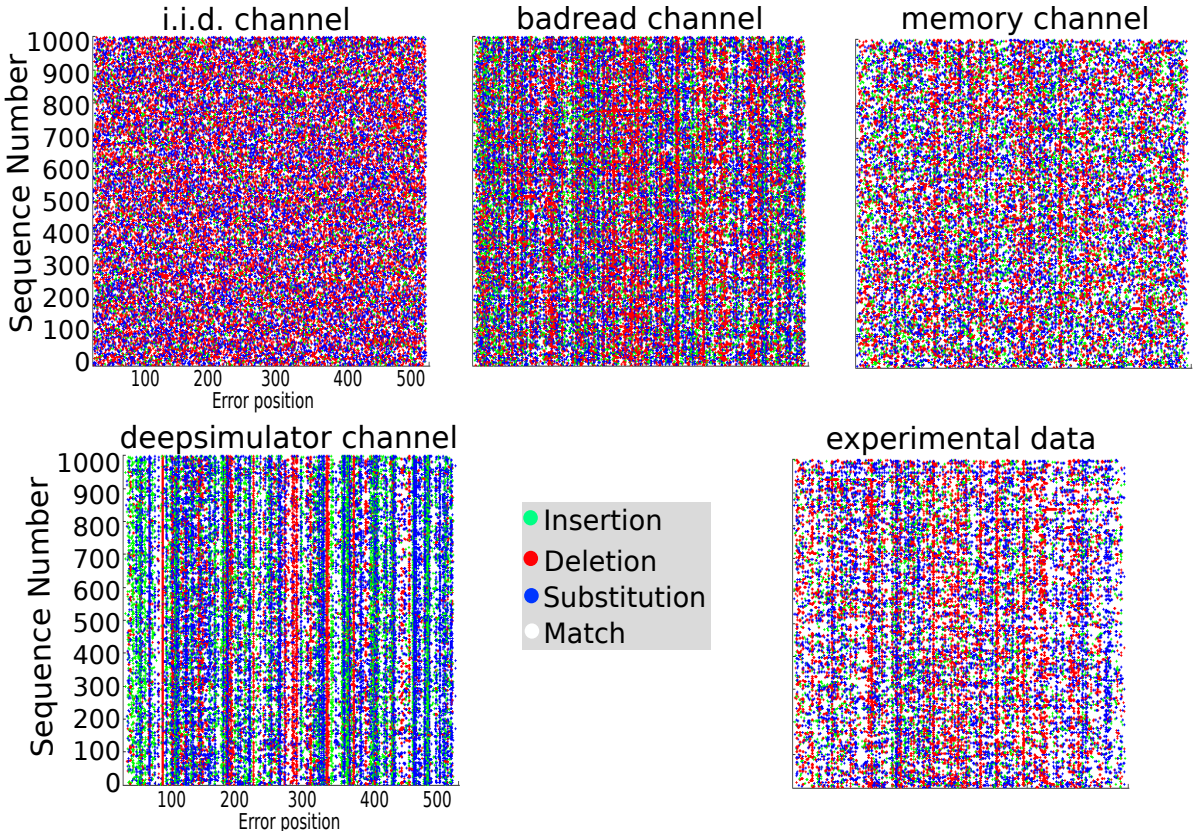


Figure 3.6: Edit maps of simulations and experimental results for $x^{(3)}$. Except for deepsimulator which have some issue to be trained, all the other channel models were trained on the *SetE*.

Figure 3.6, shows the edit maps for the third setup, where the input sequence is $\mathbf{x}^{(3)}$, and the first scenario is considered. We observe that in terms of amount, type, and patterns of errors, badread and our memory channel model approach the most the experimental data. Deepsimulator channel seems to introduce a larger amount of errors, and the amount of insertions is more important than in the experimental data. In addition, due to software issues, deepsimulator was not trained on *SetE* but uses it own training. Furthermore, as in Figure 3.4, the i.i.d. channel model does not represent well the experimental data.

Figure 3.7, shows the edit maps for the fourth setup, where the input sequence is $\mathbf{x}^{(3)}$, and the second scenario is considered. We observe that although the type of errors seems to be close between each channel model and the experimental data, the amount of errors is

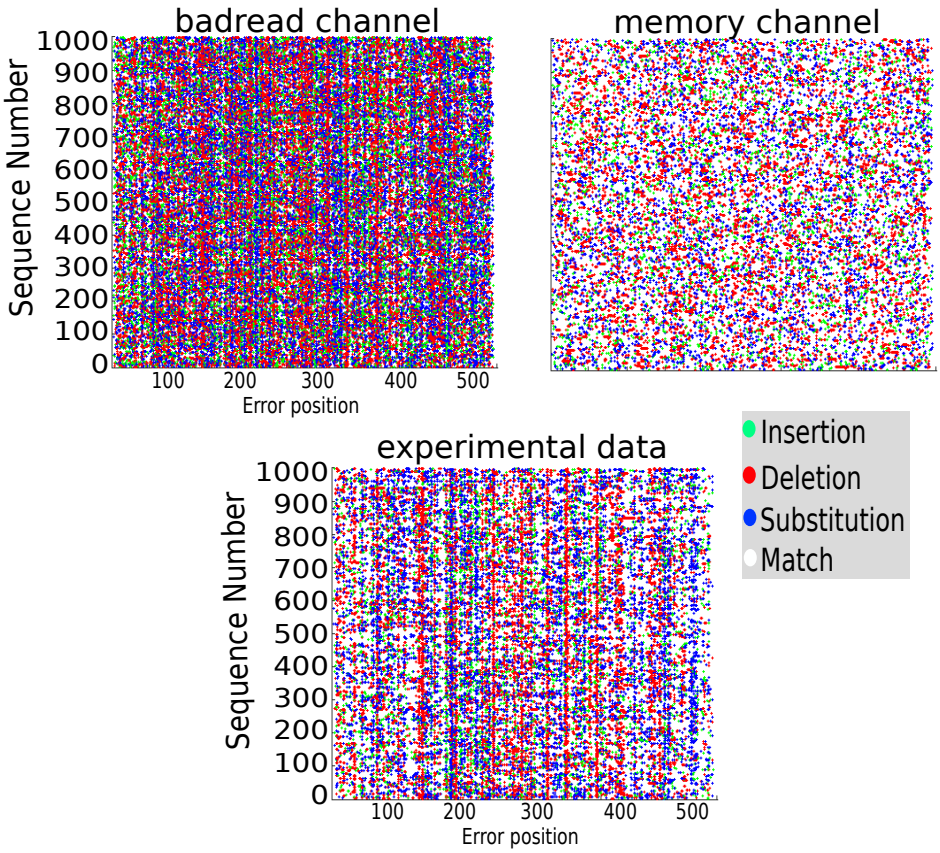


Figure 3.7: Edit maps of simulations and experimental results for $\mathbf{x}^{(3)}$. The channel models are using their default training, while the memory channel model was trained on *SetG*.

different. Badread channel introduces too much errors, when our memory channel model seems to introduce too few errors. In term of error patterns, it is difficult to tell which channel model is better. Thus, based on edit maps alone, we cannot tell which channel model is closer to the experimental data.

The edit maps are helpful to observe that some models are clearly not fitting the experimental data, and to observe error patterns. However, as we have seen previously, it can be difficult to accurately compare channel models which have edit maps visually close to the experimental data. This is why we now introduce the Kullback-Leibler divergence as a criterion for formal comparison of the models.

3.5.2 Kullback-Leibler divegence

Kullback-Leibler (KL) divergence [81, 80] is a measure from the information theory field. It is a non-symetric measure that is used to compare two probability distributions over the same variable z. It aims to measure the amount of information that is lost when approximating a probability distribution $p(z)$ by another probability distribution $q(z)$. KL divergence between distributions p and q is defined as

$$D_{KL}(p||q) = \sum_{t=1}^{N} p(z_t) \times ln\frac{p(z_t)}{q(z_t)} \tag{3.3}$$

Although it measures the difference between two distributions, KL divergence is not a distance because it is a non-symetric measure in the sense that

$$D_{KL}(p||q) \neq D_{KL}(q||p) \tag{3.4}$$

KL divergence is also widely used to perform comparison between statistical models [80, 82]. Therefore, we use KL divergence to compare channel models to experimental data. Moreover, by varying the parameter k, the KL divergence will allow us to compute different KL divergences for our memory channel model, in order to select the best memory order k.

Method

For each event $E \in \{\text{Match, Sub, Del, Ins}\}$, we estimate the corresponding distributions $\mathbb{P}_u(E)$ and $\widetilde{\mathbb{P}}_u(E)$, respectively over the experimental data sequences $\mathbf{y}^{(u,v)}$ from *SetE*

and the channel models simulated sequences $\widetilde{\mathbf{y}}^{(u,v)}$, where $u \in [\![1, 9]\!]$ and $v \in [\![1, 1000]\!]$. Thus, the marginal probability $\mathbb{P}_u(E_t)$ represents the probability to observe the event E at position t ($t \in [\![1, 500]\!]$) on the input sequence $\mathbf{x}^{(u)}$. Furthermore, to perform a fair comparison between the channel models we consider the Scenario 2, where the channel models were not trained on the *SetE*.

For each set $\mathbf{y}^{(u,v)}$ and $\widetilde{\mathbf{y}}^{(u,v)}$ related to the sequence $\mathbf{x}^{(u)}$, we compute four different KL divergences. Each KL divergence represents the KL divergence between distributions $\mathbb{P}_u(E)$ and $\widetilde{\mathbb{P}}_u(E)$ for a particular event E, using equation (3.5):

$$D_{KL}(\mathbb{P}_u(E)||\widetilde{\mathbb{P}}_u(E)) = \sum_{t=1}^{N} \mathbb{P}_u(E_t)) \times ln \frac{\mathbb{P}_u(E_t)}{\widetilde{\mathbb{P}}_u(E_t)}, \forall E \in \{\text{Match}, \text{Sub}, \text{Del}, \text{Ins}\} \tag{3.5}$$

Then for each sequence $\mathbf{x}^{(u)}$, the four KL divergences (one for each event E) are summed to obtain the KL divergence between $\mathbf{y}^{(u,v)}$ and $\widetilde{\mathbf{y}}^{(u,v)}$ as shown in equation (3.6):

$$D_{KL}(\mathbb{P}_u||\widetilde{\mathbb{P}}_u) = \sum_{E} D_{KL}(\mathbb{P}_u(E)||\widetilde{\mathbb{P}}_u(E)) \tag{3.6}$$

Thus, the overall KL divergence is the sum of the KL divergences related to each sequence $\mathbf{x}^{(u)}$, and we compute it as follows:

$$D_{KL}(B||\widetilde{B}_u) = \sum_{u=1}^{U} D_{KL}(B_u||\widetilde{B}_u) \tag{3.7}$$

This overall KL divergence is calculated for each considered channel model simulated data (Deepsimulator, Badread, i.i.d., our memory channel).

We also use equation (3.7) to evaluate the influence of the parameter k for our channel model with memory. .

Numerical results

Figure 3.8 shows the KL divergence calculated between each channel model and the experimental data which serves as reference. To choose the best memory order for our memory channel model, we compute the KL divergence for multiple values of k. Thus, except for our memory channel, all the KL divergences are constant because they are independent from the memory order. Note that a low KL divergence means a better model.

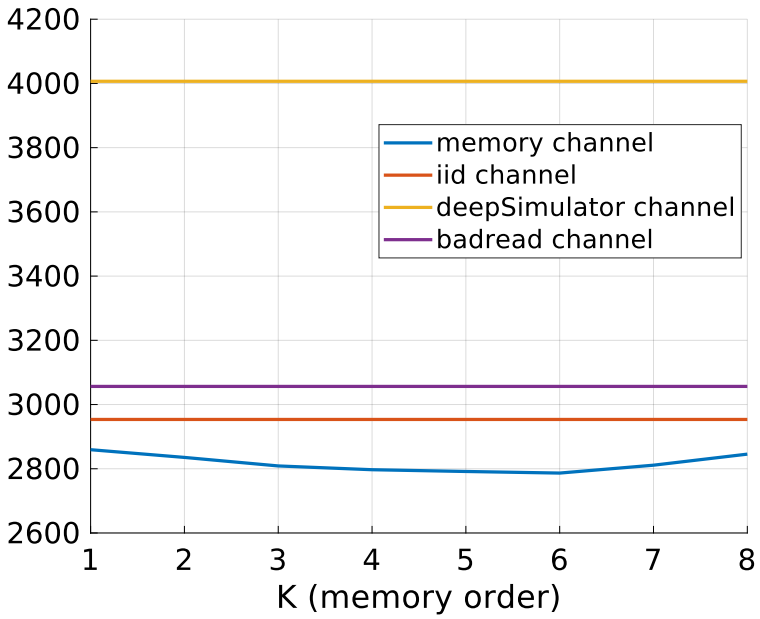


Figure 3.8: KL divergence between the channel models and the experimental data.

We also observe some interesting results in Figure 3.8:

— Deepsimulator is the channel which is approaching the least the experimental data. This result comes from the simulated event sequences $\mathbf{e}$, which has many positions E_t where there are a few or no errors (100% of matches), even though we observe errors at the same positions E_t on the experimental data.

— Badread performed better than deepsimulator, because its simulated event sequences $\mathbf{e}$ contain errors on all the positions E_t. Thus, unlike Deepsimulator, it captures the randomness of the experimental data. However, surprisingly, Badread performed worst than the i.i.d. channel. Its is due to the fact that badread usually introduces a high amount of errors compared to the experimental data, as observe in the edit maps.

— As mentioned in the previous point, the i.i.d. channel model surprisingly performed better than badread and deepsimulator. There are main reasons for that. The first one is its errors probabilities parameters $\mathbb{P}(E)$, $E \in \{\text{Match, Sub, Del, Ins}\}$, which were learned from the experimental data, and represent the average amount of errors over all the experimental data. Therefore, the amount of errors introduced by the i.i.d. channel on each position E_t is usually close to the experimental data. Thus, the

KL divergence penalties remain reasonable for the i.i.d. model. The second reason is how the KL divergence is computed and its non-symmetric property. Indeed, whenever the i.i.d. channel introduces a high amount of errors (unlike badread, never above 10%) than the experimental data (*i.e.*, $\mathbb{P}_u(E_t) < \widetilde{P}_u(E_t)$), the KL divergence penalty is smaller than if it was the inverse (*i.e.*, $\mathbb{P}_u(E_t) > \widetilde{P}_u(E_t)$), because the multiplication factor in equation (3.5) remains $\mathbb{P}_u(E_t)$ in both cases.

— Among the considered channel models, based on the KL divergence, the memory channel model is the one which approaches the most the experimental data. This is probably due to the fact that our model takes into account the dependency between the simulated sequences and how the errors are introduced. Thus, it introduces amount of errors that are statistically close to the experimental data.

— In terms of memory order for our memory channel, we notice that a memory order of $k = 6$ leads to the best KL divergence. When $k > 6$ the efficiency of the memory channel model starts to decrease due to over-training issues. Therefore, in the next chapters we set the memory order of the memory channel to $k = 6$, which is consistent with how the nanopore sequencing works, over k-mers of length 6.

3.6 Conclusion

In this chapter, we proposed a channel model with memory for DNA data storage. Through comparisons with edit maps and KL divergence, we concluded that the proposed channel model represents experimental data more accurately than other existing models. The memory channel will then allow for efficient source/channel codes design, and for code performance evaluation before developing costy in-vitro experiments. The memory channel model was already integrated in the simulator developed in the DnarXiv project.

The next two chapters introduce error-correction codes solutions that aim to correct errors introduced by the DNA data storage channel. In the first solution, the memory channel model is only used for simulation purpose. In the second solution, the knowledge of the model is taken into account to improve the decoding.

Chapter 4

First error-correction solution: Consensus and LDPC codes

In the previous chapter, we introduced our memory channel model, and discussed errors introduced during DNA synthesis and sequencing. Now, there is a need to develop error-correction codes to correct errors introduced by the channel. Our memory channel model will allow us to perform numerical simulations so as to design and evaluate the proposed error-correction codes solutions. Due to insertion and deletion errors, standard error-correction solutions completely fail to correct DNA data storage errors. Hence, there is a need to develop specific error-correction solutions that can correct these errors.

In this chapter, we introduce a first error-correction code solution for DNA data storage. This solution combines a consensus algorithm which comes from the field of bioinformatics, with Non-Binary Low Density Parity Check (NB-LDPC) codes which come from the coding field. Since after the consensus a few insertions and deletions remain, we introduce a synchronization algorithm to be used before standard NB-LDPC decoding. The synchronization algorithm relies on the NB-LDPC code structure, and helps to correct the remaining deletion errors after the consensus algorithm. In the field of bioinformatics, the consensus algorithm is a standard solution to correct errors introduced by DNA data storage. Therefore, we wanted to see if its combination with a coding solution could constitute a competitive solution for error-correction after DNA data storage.

In this chapter, we first introduce channel coding, and explain the main differences between a conventional channel and the DNA data storage channel. We also review existing DNA data storage Error-Correction solutions. We then introduce our decoding scheme which combines a consensus algorithm, a novel synchronization algorithm, and a standard NB-LDPC decoder. Finally, we show some numerical results to evaluate the proposed decoding solution.

4.1 Channel coding

In the field of telecommunications, the information transmitted from a source to a destination is often subject to noise introduced by the transmission channel. Errors introduced by the noisy channel can be corrected by relying on Error-Correction Codes (ECCs). In order to retrieve the correct information, ECCs add redundancy (extra bits) to the initial information before transmitting it.

Let $\mathbf{x}$ be the sequence transmitted over the channel, and let $\mathbf{y}$ be the received sequence. The channel introduces errors, so that in general $\mathbf{y} \neq \mathbf{x}$, as shown in Figure 4.1. To recover the sequence $\mathbf{x}$, it is necessary to use an ECC that encodes the sequence $\mathbf{x}$ of length K, into a new sequence $\mathbf{c}$ of length N ($N > K$), called a codeword. The sequence $\mathbf{c}$ is transmitted through the channel instead of $\mathbf{x}$, as shown in Figure 4.2. The channel decoder outputs a sequence $\hat{\mathbf{x}}$, which is expected to be equal to the original sequence $\mathbf{x}$. Given that $N > K$, the redundancy in $\mathbf{c}$ helps the decoder to recover $\mathbf{x}$ from the received sequence $\mathbf{y}$.

Currently, the ECC solutions that offer the best trade-offs between complexity and decoding performance are Turbo codes [25], LDPC codes [26] and Polar codes [83]. In this work, we investigate the use of LDPC codes because there exists Non-Binary LDPC (NB-LDPC) codes [84] for a long time. In addition, to be consistent with the DNA alphabet, we use NB-LDPC codes that are defined over Galois Fields GF(4), *i.e.*, four symbols are allowed. Furthermore, a lot of LDPC code design tools are available in our research team.

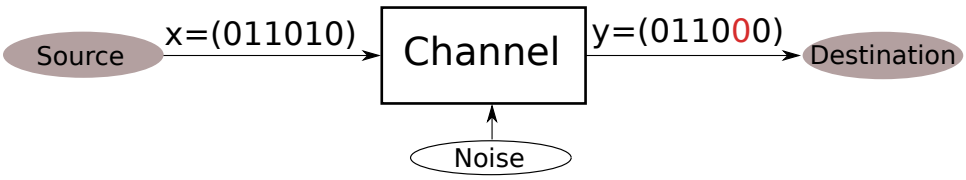


Figure 4.1: Transmission without channel coding. The bit highlighted in red on the received sequence $\mathbf{y}$, is not equal to the emitted bit on sequence $\mathbf{x}$.

4.2 DNA data storage channel VS classical channels

DNA data storage channel is different from conventional telecommunication channels [85, 86, 87] in several aspects.

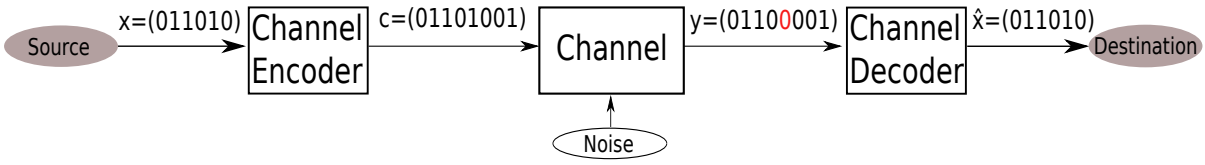
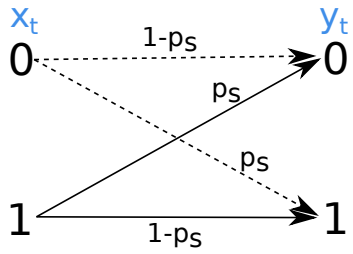


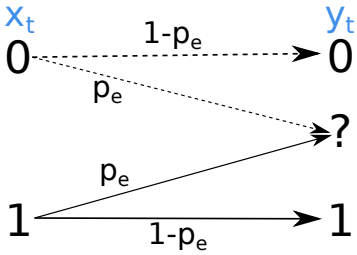
Figure 4.2: Transmission with channel coding. Although the bit highlighted in red on the received sequence $\mathbf{y}$ is not equal to the emitted bit on sequence $\mathbf{c}$, the channel decoder is able to guess the correct sequence $\hat{\mathbf{x}}$.

4.2.1 Nature of errors

Most standard telecommunication channels are subject to additive noise, substitution errors, or erasures. For instance, in the Binary Symetric Channel (BSC), a binary symbol x_t is flipped with a certain probability p_s, see Figure 4.3a. As another example, the Binary Erasure Channel (BEC) introduces erasures with a certain probability p_e, so that either $y_t = x_t$ (no erasure) or $y_t =?$ (erasure), see Figure 4.3b. Oppositely, the DNA data storage channel is subject not only to substitutions, but also to insertion and deletion errors, see Figure 4.4. Insertions and deletions introduce synchronization errors. In particular, a deletion is different from an erasure in the sense that, the position of an erasure is known, while the position of a deletion is unknown. Substitution errors in DNA data storage are the same as in conventional channels.



(a) Binary Symetric Channel.



(b) Binary Erasure Channel.

Figure 4.3: Examples of standard telecommunication channels

4.2.2 Symbol dependency

Most conventional channels are memoryless in the sense that the probability of the output symbol y_t depends only on the input symbol x_t at the same position t. On the opposite, due to synchronization errors and how the MinION sequencer works, the DNA

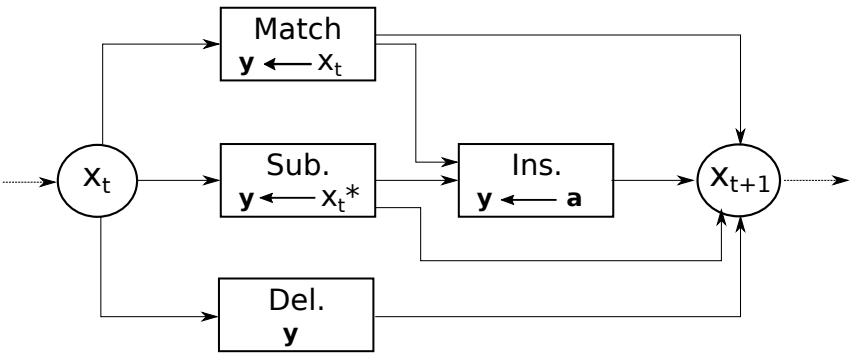


Figure 4.4: DNA data storage error channel. In this figure $x_t \neq x_t*$, and $\mathbf{a}$ is a vector of one or several symbols.

data storage channel has memory. Thus, the probability of an output symbol y_t, might also depend on input symbols $x_{t'}$ at positions $t' \neq t$.

4.2.3 Received data

A sequence $\mathbf{x}$ transmitted over a conventional channel produces a unique received sequence $\mathbf{y}$, of the same length N as $\mathbf{x}$. In DNA data storage channel, a transmitted sequence $\mathbf{x}$ produces V received sequences $\mathbf{y}^{(v)}$, and due to the synchronization errors, the length $N^{(v)}$ of $\mathbf{y}^{(v)}$ can differ from N.

While coding for conventional channels has a rich history and is well understood, only few is known about coding for insertion and deletion channels [87, 88, 89]. In the next section, we review existing ECCs solutions for synchronization errors, and for DNA data storage in general.

4.3 Existing ECC solutions for synchronization errors

There exist a wide range of ECC solutions to correct substitution and erasure errors, and some of them achieve near-capacity performance, such as Turbo-codes [25], LDPC codes [26], etc.. However, because of the loss in synchronization, the conventional ECC solutions cannot be applied to correct insertion and deletion errors [87, 88, 89]. Existing ECCs for synchronization errors can correct insertions and/or deletions only under some restrictive assumptions. The first introduced ECC solutions for synchronization errors were only able to correct one unique insertion or deletion [90]. Some more recent ECC

solutions allow to correct a few insertions or a few deletions per codeword, but burst of errors cannot be corrected [90, 91, 92]. Other ECCs are able to correct a few errors or a unique burst of errors on a codeword of length N, but only on a specific window of length N_{window} ($N_{window} < N$), *i.e.*, if the distance (number of bits) between two corrupted bits is larger than N_{window}, then the codeword cannot be corrected [93, 94].

In the context of DNA data storage, two main approaches are often considered. The first approach consists of combining a consensus algorithm, with a standard channel decoder. The consensus algorithm is supposed to correct the vast majority of errors, while the channel decoder handles remaining residual errors [97, 95, 96]. The second approach relies on pure channel coding solutions, without resorting to a consensus algorithm [100, 99, 101, 98]. We explore the first approach in this chapter, while the second approach will be explored in the next chapter.

In this chapter, we consider the consensus algorithm proposed in [102] together with NB-LDPC codes [84] that fit the quaternary DNA alphabet. As an alternative to [95] in order to handle the few remaining insertions and deletions, we propose a novel intermediate step which we insert between the consensus algorithm and the channel decoding. This intermediate step consists in re-synchronizing the sequence at the output of the consensus, by relying on the LDPC code structure rather than on additional markers. This represents an interesting gain in terms of coding redundancy, which may allow to reduce the expensive synthesis costs. In what follows we call this error-correction scheme the CSL (Consensus, Synchronization, LDPC codes) solution.

4.4 Proposed CSL solution

We now describe our first proposed error-correction scheme for DNA data storage. This scheme relies on three components: a consensus algorithm, a resynchronization step, and a NB-LDPC decoder. The first component aims to provide a sequence of good quality by relying only on the redundancy introduced by sequencing (e.g. the fact that sequencing outputs not 1, but V sequences), while the last two components aim to correct residual errors. The consensus algorithm relies on the fact that known primers (short sequence of about 40 nucleotides) are added at the beginning and at the end of the sequence $\mathbf{x}^{(u)}$, ($u \in [\![1, U]\!]$). These primers consist of sequences of known bases which are mainly used to select sequences of interest through biotechnology manipulations.

The CSL reconstruction solution is shown on Figure 4.5. The first step consists of

applying the CCSA consensus algorithm [102] over M sequences selected at random among the V sequences that have length greater than N and correct primers p_{start} and p_{end}. The consensus algorithm then outputs one or several sequences $\overline{\mathbf{y}_{cons}(g)}$ of length N_g. If the consensus outputs a sequence of length $N_g = N$, we set this sequence as $\overline{\overline{\mathbf{y}_{cons}}}$. Otherwise, we pick a sequence of length $N_g < N$ as close as possible to N, and we apply a synchronization method in order to get a sequence $\overline{\overline{\mathbf{y}_{cons}}}'$ of the correct length N. In both cases, the sequence $\overline{\overline{\mathbf{y}_{cons}}}'$ (or $\overline{\overline{\mathbf{y}_{cons}}}$ if there is no need for synchronization) is passed through the NB-LDPC decoder [84] to correct residual substitution errors. If the sequence $\hat{\mathbf{c}}$ is incorrect in the sense that $\hat{\mathbf{c}}.\mathbf{H}^T \neq \mathbf{0}$, we restart the full reconstruct process (consensus + synchronization + NB-LDPC decoder), from another set of M random sequences provided to the consensus. This process allows to reconstruct the original sequence $\mathbf{c}$ with a small number of restarts.

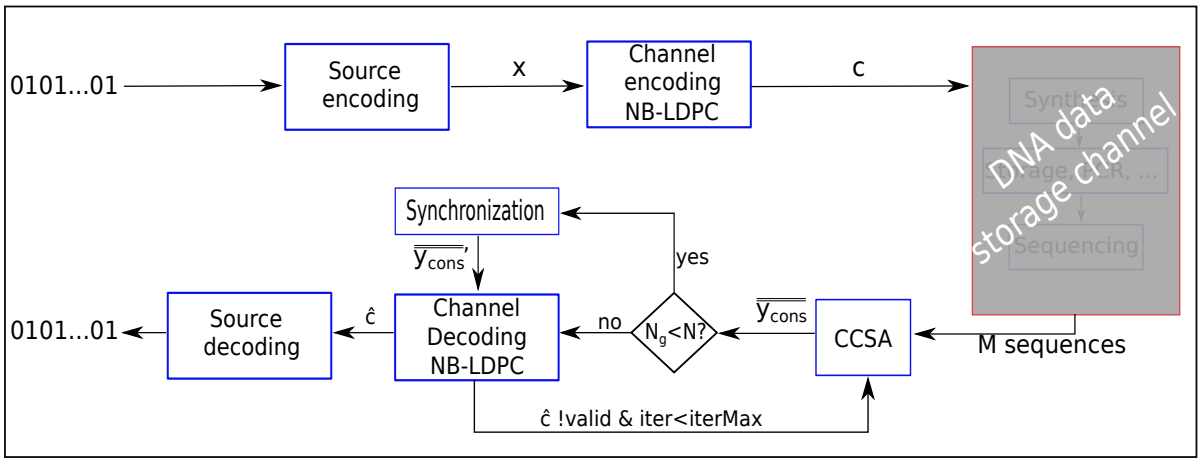


Figure 4.5: This figure represents the full reconstruction solution based on the consensus, synchronization, and NB-LDPC decoder.

We now describe into details each of the components of the CSL solution.

4.5 Consensus algorithm

Consensus algorithms are widely used techniques in the field of bioinformatics. They aim to reconstruct a sequence $\overline{\overline{\mathbf{y}_{cons}}}$ called "consensus sequence", from a set of M sequences $\mathbf{y}^{(m)}$ called "edited replicas". It is expected that the consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$ is error-free or at least contains less errors than the edited replicas $\mathbf{y}^{(m)}$. There exists various consen-

sus algorithms, but most of them rely on alignments and majority votes to reconstruct $\overline{\overline{\mathbf{y}_{cons}}}$ [103, 104].

The consensus algorithm we consider, called CCSA (Constrained Consensus Sequence Algorithm) [102], has been specifically designed for DNA data storage by Dominique Lavenier [102] in the framework of the DnarXiv project. CCSA takes as input M sequences $(\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(M)})$ selected randomly from the set of V sequences output by the basecaller. Then, for given parameters T and l, the algorithm forms a directed graph whose nodes are given by the subsequences of length l that appear at least T times among the M input sequences. There is an edge between two nodes of the graph if the suffix subsequence of the first node overlaps by at least d bases the prefix subsequence of the second node. Finally, a Viterbi-like algorithm is applied over the directed graph in order to select the path with the highest score between the start primer (p_{start}) and the end primer (p_{end}). Note that CCSA outputs either one or G sequences $\overline{\overline{\mathbf{y}_{cons}(g)}}$ of equal (highest) score but of different lengths N_g close to N. The score of a path is calculated by taking into account the nodes weights (number of occurrence of the subsequence over the M sequences) and the edges weights (l minus overlap length between the two subsequences) over the path. A detailed description of the CCSA algorithm can be found in [102]. CCSA results reported in [102], as well as our own simulations, show that this algorithm is able to correct most of the errors introduced by the DNA storage process, although a few residual insertions, deletions, and substitutions remain. This is why additional correction steps based on NB-LDPC decoders are needed.

4.6 Low Density Parity Check Codes (LDPC)

LDPC codes [105, 26, 107, 106] are linear capacity-approaching block codes commonly used in communication systems to correct channel substitution errors. In the binary case, a [N,K] LDPC code is defined by a sparse binary parity check matrix $\mathbf{H}$ (see Figure 4.6) of size $(N-K) \times N$. A sequence $\mathbf{c}$ is a codeword if it satisfies the linear equation

$$\mathbf{c}.\mathbf{H}^T = \mathbf{0} \tag{4.1}$$

Each row of $\mathbf{H}$ corresponds to a parity check equation where all the arithmetic operations are modulo 2 (division remainder), and each column of $\mathbf{H}$ corresponds to a bit from the codeword $\mathbf{c}$. Furthermore, to generate a codeword $\mathbf{c}$ from the sequence $\mathbf{x}$, a generator

matrix $\mathcal{G}$ is derived from $\mathbf{H}$ for instance through a Gaussian-Jordan elimination [108], so that $\mathcal{G}.\mathbf{H}^T = \mathbf{0}$ and $\mathbf{c} = \mathbf{x}.\mathcal{G}$.

The LDPC parity-check matrix $\mathbf{H}$ can also be represented by a bipartite graph called Tanner graph [109], see Figure 4.7. The Tanner graph contains two types of nodes: the variable-nodes and the check-nodes. The variable-nodes correspond to the codeword bits and the check-nodes correspond to the parity check equations. Therefore, the Tanner graph contains N variable-nodes which are denoted VN_i ($i \in [\![1, N]\!]$) and are represented with circles, and N-K check-nodes denoted CN_j ($j \in [\![1, N-K]\!]$) and represented with squares. There is an edge between a variable-node VN_i and a check-node CN_j if VN_i is involved in the parity-check equation j, *i.e.*, $H_{i,j} \neq 0$.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4.6: An example of LDPC parity-check matrix $\mathbf{H}$. Each row represents a parity-check equation involving some of the codeword bits.

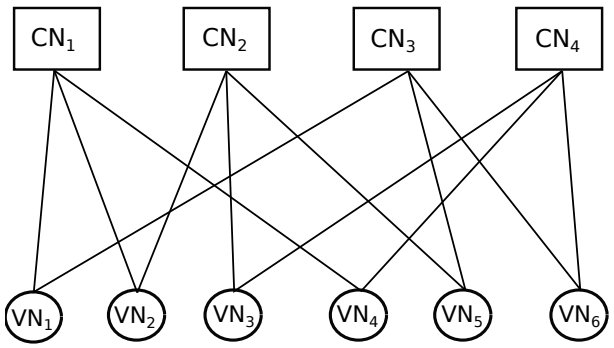


Figure 4.7: Tanner graph representation of the parity check matrix $\mathbf{H}$. The check-nodes CN_j are represented by squares, and The variable-nodes VN_i are represented by circles.

As an example, let us assume a received sequence $\mathbf{y} = [0\ 0\ 1\ 0\ 1\ 1]$. When considering the matrix $\mathbf{H}$ given in Figure 4.6, in order to verify if equation (4.1) is satisfied, the

following parity check equations are calculated:

$$\begin{aligned} CN_1: &\quad VN_1 \oplus VN_2 \oplus VN_4 = 0 \oplus 0 \oplus 0 = 0 \\ CN_2: &\quad VN_2 \oplus VN_3 \oplus VN_5 = 0 \oplus 1 \oplus 1 = 0 \\ CN_3: &\quad VN_1 \oplus VN_5 \oplus VN_6 = 0 \oplus 1 \oplus 1 = 0 \\ CN_4: &\quad VN_3 \oplus VN_4 \oplus VN_6 = 1 \oplus 0 \oplus 1 = 0 \end{aligned} \tag{4.2}$$

In this example, we see that all the parity check equations are satisfied. Thus, $\mathbf{y}$ is a codeword.

Various LDPC decoders exist [108], and the standard Belief-Propagation (BP) decoder is the most efficient in terms of decoding and complexity. The BP decoder is an iterative decoder, which passes messages through the Tanner graph edges, from variable-nodes to check-nodes and *vice versa*.

The LDPC codes performance improves with large values of N [108]. However, as mentioned in Chapter 2, the current synthesis technologies limits the length of the sequences to a few hundred nucleotides. Thus, to improve the decoding performance, we use a NB-LDPC code which outperforms the binary LDPC codes on short sequences [84]. NB-LDPC codes are defined over Galois fields GF(q) of order $q \geq 2$. Furthermore, for consistency with the DNA alphabet, we consider $q = 4$.

4.6.1 Galois Fields

Before describing NB-LDPC codes, we briefly introduce Galois fields. A Galois field [110, 111] is a field which contains a finite number q of elements, that is GF(q) = $\{0, 1, ..., q\}$. All GF(q) arithmetic operations, namely addition, subtraction, multiplication and division, can be performed under field axioms constraints. Since GF(q) is a field, the result of any arithmetic operation over GF(q) is an element of GF(q). GF(q) is either a prime field if q is a prime number, or an extension field if the order q is a power of a prime number *i.e.* $q = \mathcal{P}^{\lambda}$, where $\mathcal{P}$ is a prime number and $\lambda \in \mathbb{N}^{+}$ is a positive integer. Arithmetic operations over a prime field can be done using integer arithmetic, followed by a *modulo* q operation. However, in order to perform arithmetic operations over an extension field, the elements of GF(q) are represented with polynomials of degree at most equal to λ:

$$poly(X) = a_{\lambda-1}X^{\lambda-1} + ... + a_1X^1 + a_0 \in GF(q = \mathcal{P}^{\lambda}) \tag{4.3}$$

Where a_i are coefficients that take values in GF($\mathcal{P}$). Thus, to represent an element from GF($\mathcal{P}^\lambda$) based on GF($\mathcal{P}$) elements, we need a vector of length λ. Furthermore, extension fields arithmetic operations are followed by a *modulo* $Ir(X)$ operation, where $Ir(X)$ is an irreducible polynomial which cannot be factored over GF($\mathcal{P}$).

Let us consider a Galois field of 4 elements: GF(4) = $\{0, 1, 2, 3\}$. Since GF(4) = GF(2^λ), then, to perform arithmetic operations, we first need to extend GF(2) so as to represent the elements of GF(4) with a polynomials of the form

$$poly(X) = a_1.X^1 + a_0 \in GF(4) \tag{4.4}$$

Table 4.1, shows the polynomial representation of GF(4) elements.

Integer	Vector	Polynomial
0	$[0, 0]$	$0.X^1 + 0 = 0$
1	$[0, 1]$	$0.X^1 + 1 = 1$
2	$[1, 0]$	$1.X^1 + 0 = X$
3	$[1, 1]$	$1.X^1 + 1 = 1 + X$

Table 4.1: Polynomial representation of GF(4) elements

To define arithmetic operations over the field, we first define an irreducible polynomial $Ir(X) = X^2 + X + 1$, which cannot be factorized over GF(2). Accordingly, Tables 4.2 and 4.3, show addition and multiplication operations, respectively.

$\oplus$	0	1	X	$1 + X$
0	0	1	X	$1 + X$
1	1	0	$1 + X$	X
X	X	$X + 1$	0	1
$1 + X$	$1 + X$	X	1	0

Table 4.2: Addition over GF(4)

In GF(2^λ), a subtraction is equivalent to an addition. Furthermore, the division can be transformed into a multiplication with the inverse element. For instance, $\frac{a}{b} = a \otimes b^{-1}$, where $a, b, b^{-1} \in$ GF(q), and b^{-1} is the inverse element of b. Since $b \otimes b^{-1} = 1$. Thus, inverse elements can be deduced from the multiplication table. Accordingly, Table 4.4 shows the results of the division operation over GF(4).

In order to perform the arithmetic operations on the GF(4) NB-LDPC encoder and decoder, we use Tables 4.2, 4.3 and 4.4.

$\otimes$	0	1	X	$1+X$
0	0	0	0	0
1	0	1	X	$1+X$
X	0	X	$1+X$	1
$1+X$	0	$1+X$	1	$1+X$

Table 4.3: Multiplication over GF(4)

$\oslash$	1	X	$1+X$
0	0	0	0
1	1	$1+X$	X
X	X	1	$1+X$
$1+X$	$1+X$	X	1

Table 4.4: Division over GF(4)

4.6.2 NB-LDPC codes

In this work, we use NB-LDPC codes [112, 84, 113] in GF(4) for consistency with the quaternary bases alphabet $\{A,\ C,\ G,\ T\}$. In this case, the non-zero elements of the parity check matrix $\mathbf{H}$ take values in GF(4). Then, any codeword $\mathbf{c}$ with elements of GF(4) verifies $\mathbf{c}.\mathbf{H}^T = \mathbf{0}$, where the arithmetic operations are evaluated over GF(4). We consider a standard BP decoder initialized with probabilities obtained after applying the CCSA algorithm over the data Set$E$ described in Chapter 3. The NB-LDPC BP decoder takes as input the consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$ and seeks to output a sequence $\hat{\mathbf{c}}$ close to $\mathbf{c}$ and that verifies the condition $\hat{\mathbf{c}}.\mathbf{H}^T = \mathbf{0}$. The decoding process is almost the same as for binary LDPC, though the message formats and equations change slightly, and some steps are added. To simplify the notation, we denote elements of GF(4) as integers instead of polynomials, although as mentionned previously, the polynomial representation is required to perform the arithmetic operations. In what follows, we use $\mathcal{M}_{ij}$ to denote the message that goes from the variable-node i to the check-node j, and we use $\mathcal{M}_{ji}$ to denote the message that goes from the check-node j to the variable-node i. We now describe the decoding process for the NB-LDPC code.

1. **Initialize variable nodes** with *a priori* probabilities. Unlike the binary LDPC decoder, a vector of likelihoods is considered rather than a unique likelihood:

$$\mathcal{L}(c_i) = \left[\frac{\mathbb{P}(c_i=0|y_i)}{\mathbb{P}(c_i=0|y_i)}, \frac{\mathbb{P}(c_i=1|y_i)}{\mathbb{P}(c_i=0|y_i)}, \frac{\mathbb{P}(c_i=2|y_i)}{\mathbb{P}(c_i=0|y_i)}, \frac{\mathbb{P}(c_i=3|y_i)}{\mathbb{P}(c_i=0|y_i)}\right] \tag{4.5}$$

The *a priori* probabilities $\mathbb{P}(c_i = b|y_i)$ were obtained from our experiments with the CCSA algorithm.

2. **Compute the variable-nodes to check-nodes messages $\mathcal{M}_{ij}$** as

$$\forall\ \tau \in GF(4),\ \mathcal{M}_{ij}[\tau] = \begin{cases} \mathcal{L}(c_i)[\tau] & \text{, if } 1^{st} \text{ iteration} \\ \mathcal{L}(c_i)[\tau] \prod\limits_{j' \in \mathrm{SCN}_i \backslash \{j\}} \mathcal{M}_{j'i}[\tau] & \text{, otherwise} \end{cases} \tag{4.6}$$

where SCN_i is the set of check-nodes that are connected to the variable-node VN_i. The notation $j' \in \mathrm{SCN}_i \backslash \{j\}$ means that we consider all the nodes in SCN_i except the j^{th} node.

3. **Permutation of the variable-nodes to check-nodes messages**. In the binary version, the non-zero elements $h_{i,j}$ ($i \in [\![1, N]\!]$, $j \in [\![1, N-K]\!]$) of $\mathbf{H}$ are equal to one. Thus, it is not necessary to perform a multiplication $\mathcal{M}_{ij}.h_{i,j}$ of the message and the non-zero element. However, the NB-LDPC code has non-zero elements $h_{i,j} \in \mathrm{GF}(q)$, which makes a multiplication necessary. To do so, note that the multiplication over $\mathrm{GF}(q)$ can be simplified using left shifts, see Table 4.3. Therefore, at the end of this operation, elements of the message $\mathcal{M}_{ij}$ are shifted to the left by $h_{i,j}$ positions. For instance, if $h_{i,j} = 2$, the message $\mathcal{M}_{ij} = [\ M_{ij}[0],\ M_{ij}[1],\ M_{ij}[2],\ M_{ij}[3]\ ]$ becomes $\mathcal{M}_{ij} = [\ M_{ij}[2],\ M_{ij}[3],\ M_{ij}[0],\ M_{ij}[1]\ ]$.

4. **Compute the check-nodes to variable-nodes messages $\mathcal{M}_{ji}$** as

$$\forall\ \tau, \tau' \in GF(4),\ \mathcal{M}_{ji}[\tau] = \sum_{\mathrm{CN}_j = 0, \mathrm{VN}_i = \tau} \prod_{i' \in \mathrm{SVN}_j \backslash \{i\}} \mathcal{M}_{i'j} \tag{4.7}$$

where SVN_j is the set of variable-nodes that are connected to the check-node CN_j. The sum $\sum\limits_{\mathrm{CN}_j = 0, \mathrm{VN}_i = \tau}$ allows to consider all the configurations where the parity-check equation CN_j is satisfied given that $\mathrm{VN}_i = \tau$. Therefore, $\mathcal{M}_{ji}[\tau]$ corresponds to the probability that $\mathrm{CN}_j = 0$ is satisfied given that $\mathrm{VN}_i = \tau$.

5. **Permutation of check-nodes to the variable-nodes messages**. To inverse the permutation introduced previously by $h_{i,j}$, a division by $h_{i,j}$ is necessary. To do so, note that the division over $\mathrm{GF}(q)$ can be simplified using right shifts, see Table 4.4. At the end of this operation, elements of the message $\mathcal{M}_{ji}$ are shifted to the right by $h_{i,j}$ positions. For instance, if $h_{i,j} = 1$, the message $\mathcal{M}_{ji} = [\ M_{ji}[0],\ M_{ji}[1],\ M_{ji}[2],\ M_{ji}[3]\ ]$, becomes $\mathcal{M}_{ji} = [\ M_{ji}[3],\ M_{ji}[0],\ M_{ji}[1],\ M_{ji}[2]\ ]$.

6. **Estimate** $\hat{c}_i$ with the *a posteriori* probability $APP(c_i)$, calculated as

$$\forall\ s \in GF(4),\ APP(c_i)[\tau] = \mathcal{L}(c_i)[\tau] \prod_{j \in SCN_i} \mathcal{M}_{ji}[\tau] \tag{4.8}$$

The decoded symbol $\hat{c}_i$ is set to be the maximum argument (arg max) of $APP(c_i)$, *i.e.* the element with the highest likelihood over $APP(c_i)$. After computing the temporary estimate of $\hat{\mathbf{c}}$, if $\hat{\mathbf{c}}.\mathbf{H}^t = \mathbf{0}$, then $\hat{\mathbf{c}}$ is a valid codeword, and we stop the decoding. Otherwise, we repeat steps two to six until $\hat{\mathbf{c}}.\mathbf{H}^t = \mathbf{0}$ have been satisfied or a maximum number of iterations has been reached.

However, the standard BP decoder as well as most existing LDPC decoders can only correct substitution errors. This is why we now propose a synchronization method to also correct a few amount of deletions. We only apply this synchronization step if the consensus does not output a sequence of length $N_g = N$, and by assuming that output sequences of length $N_g < N$ only result from deletions.

4.7 NB-LDPC codes synchronization

In this section, we first show some results obtained when using the CCSA algorithm. We then introduce the synchronization method.

4.7.1 Residual errors after the CCSA algorithm

In order to evaluate which kind of errors remain after the CCSA algorithm, we apply this algorithm onto all the sequences contained in set$E$. More into details, we run the CCSA algorithm 1000 times for each reference sequence $\mathbf{x}^{(u)}$, where $u \in [\![1, U]\!] \backslash \{2\}$. For a given reference $\mathbf{x}^{(u)}$, for each run of the algorithm, we select M sequences $\mathbf{y}^{(u,v)}$ at random, where $v \in [\![1, V]\!]$ is the number of output sequences corresponding to $\mathbf{x}^{(u)}$. For a given run, the M selected sequences are then used as inputs to the CCSA algorithm. We also vary the parameter M, which corresponds to the number of sequences $\mathbf{y}^{(u,v)}$ taken into account by the CCSA. Note that, we do not consider the sequence $\mathbf{x}^{(2)}$ because of its high amount of long homopolymers, which makes it impossible to reconstruct a consensus sequence using the CCSA.

Figure 4.8 shows rates and types of errors obtained after the CCSA algorithm with respect to the number M of selected sequences. We observe that the amount of errors is

relatively small on the output consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$. However, although we consider different values M of input sequences for the CCSA, the majority of observed errors are synchronization errors (insertions and deletions). Furthermore, the value of M has only a slight effect on the error rate, because except for the sequence $\mathbf{x}^{(2)}$, the sequences from Set$E$ contain none or short homopolymers (≤ 3). This is why to use the NB-LDPC decoder, it is first necessary to re-synchronize the consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$ to handle synchronization errors.

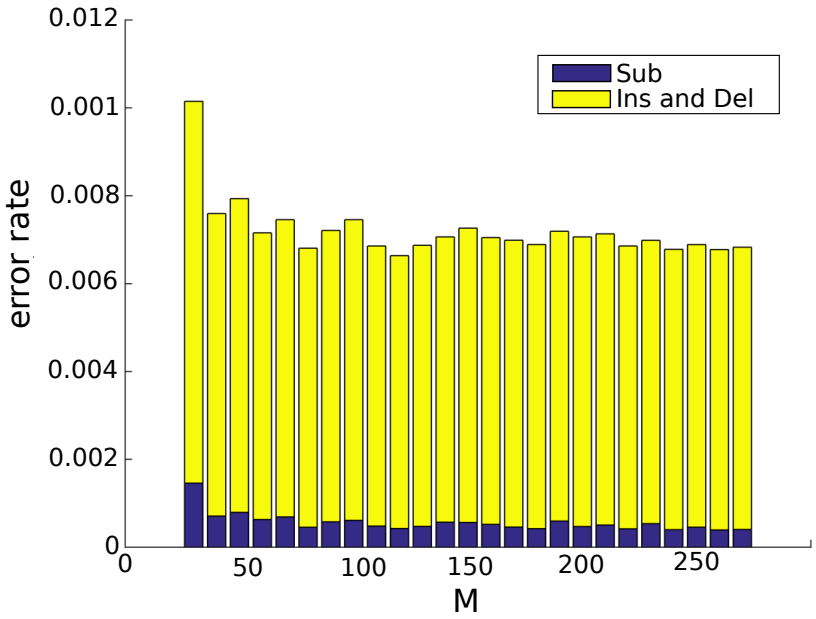


Figure 4.8: This figure represents the type and amount of observed errors in the consensus sequences $\overline{\overline{\mathbf{y}_{cons}}}$. We observe that increasing M does not decrease the amount of synchronization errors (insertions and deletions).

4.7.2 Synchronization algorithm

We now introduce a novel synchronization method, which allows to transform synchronization errors that may remain in the consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$, into substitution errors. This method relies on the structure of the considered NB-LDPC code, and more precisely on its parity check equations.

We use $\mathcal{S}(\mathbf{y})$ to denote the score of a given sequence $\mathbf{y}$, which corresponds to the number of unsatisfied parity check equations, that is the number of non-zero components

in the vector $\mathbf{y}.\mathbf{H}^T$. For instance, let us assume an LDPC code using the parity-check matrix $\mathbf{H}$ given in Figure 4.6, and a received sequence $\mathbf{y} = [0\ 0\ 1\ 1\ 1\ 1]$. In order to compute $\mathcal{S}(\mathbf{y})$, the check-nodes equations are evaluated :

$$\begin{aligned} CN_1: &\quad VN_1 \oplus VN_2 \oplus VN_4 = 0 \oplus 0 \oplus 1 = 1 \\ CN_2: &\quad VN_2 \oplus VN_3 \oplus VN_5 = 0 \oplus 1 \oplus 1 = 0 \\ CN_3: &\quad VN_1 \oplus VN_5 \oplus VN_6 = 0 \oplus 1 \oplus 1 = 0 \\ CN_4: &\quad VN_3 \oplus VN_4 \oplus VN_6 = 1 \oplus 1 \oplus 1 = 1 \end{aligned} \tag{4.9}$$

In this case, $\mathcal{S}(\mathbf{y}) = 2$, since there are two unsatisfied parity-check equations.

We now describe the synchronization algorithm when the consensus outputs a sequence $\overline{\overline{\mathbf{y}_{cons}}}$ of length $N-1$. This algorithm has 3 main steps:

1. *Segmentation*: the consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$ is segmented into multiple segments of length l_s. the segment length l_s plays a crucial role as it addresses a tradeoff between complexity and amount of substitution errors in the resulting sequence.
2. *Insertion attempts*: we then try to insert a base with arbitrary value "A" at position 1, then at position $l_s + 1$, then $2.l_s + 1$, and so on. For each considered position, we compute the corresponding sequence score $\mathcal{S}(\overline{\overline{\mathbf{y}_{cons}}})$. This step aims to detect the segment where the deletion occured.
3. *Definitive insertion*: at the end, we permanently add a base "A" at the position $(i.l_s)+1$ ($i \in [\![0, \frac{N}{l_s} - 1]\!]$) that gives the lowest score. This gives a new vector $\overline{\overline{\mathbf{y}_{cons}}}'$. Thus, on the sequence $\overline{\overline{\mathbf{y}_{cons}}}'$ the deletion error has been transformed into one or several substitutions depending on the position of the deletion in the segment.

Figure 4.9 illustrates the synchronization algorithm with an example. In this example, the codeword $\mathbf{c}$ =[ACGTCACGGA] after the CCSA consensus step is affected by the deletion of the "C" base highlighted in red. To simplify the example, the score is here calculated as the number of unsynchronized bases instead of the number of unsatisfied parity-check equations. As the first step, the consensus sequence $\overline{\overline{\mathbf{y}_{cons}}}$ is splitted with $l_s = 2$. At the second step, an arbitrary base "A" is inserted at the beginning of each segment, and all resulting scores are computed. Finally, the base "A" is definitively inserted into the segment with the lowest score, transforming the deletion into a substitution highlighted in blue.

Even if the position and value of the inserted base are not entirely correct, adding this

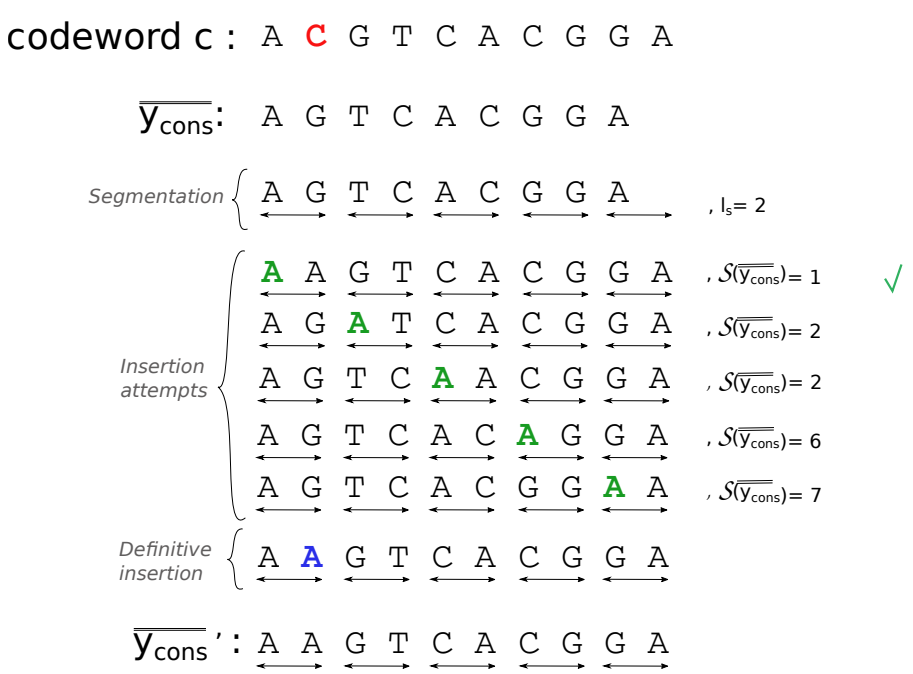


Figure 4.9: This figure illustrates the proposed synchronization algorithm over the sequence example $\mathbf{c}$ =[ACGTCACGGA]. The "C" base highlighted in red was deleted during the CCSA step.

base close to the correct position should greatly reduce the score by re-synchronizing the output sequence $\overline{\overline{\mathbf{y}_{cons}}}'$ with the original codeword $\mathbf{c}$.

Now, if the sequence $\overline{\overline{\mathbf{y}_{cons}}}'$ is of lower length $N - \tau$ ($\tau \in \mathbb{N}^+$), then τ bases are inserted one after each other in a greedy maneer: the first base is inserted at the position which gives the lowest score, then the second an third step are repeated so as to insert the second base, and so on. This process allows to replace deletion errors by substitution errors which can then be corrected by the NB-LDPC BP decoder. If after synchronization, the decoder fails (unsatisfied parity check equations remain), a new consensus will be restarted. Note that we only consider deletions, because the results of our experiments showed that the proportion of deletion is more important than insertions in both sequencing and consensus steps. However, it would be possible to tackle insertions by slightly adapting the synchronization algorithm.

This synchronization technique only relies on the LDPC code structure, and does not require any additional redundancy, unlike in the solution with periodical markers proposed in [62].

Algorithm complexity

The introduced synchronization algorithm checks $\tau.(\frac{N}{l_s}).(N-K)$ parity check equations, where τ corresponds to the number of deletions, $\frac{N}{l_s}$ corresponds to the number of segments, and $N-K$ corresponds to the number of parity-check equations. Thus, the algorithm complexity is linear with the code length N.

We now evaluate the CSL reconstruction solution through numerical simulations.

4.8 Numerical results

In this section, we evaluate the performance of our full CSL reconstruction method, by using the channel model with memory described in Chapter 3.

4.8.1 Performance of synchronization algorithm and NB-LDPC decoder

We first evaluated the performance of the synchronization algorithm followed by NB-LDPC decoding, for various values of M. The synchronization algorithm being a novel method, the main purpose here is to check if it helps to improve the NB-LDPC decoder performance when there are deletions on the received sequence.

Each simulation run considers a randomly generated input sequences $\mathbf{X}$ of length K in GF(4), where each element X_i takes values in alphabet $\{A, C, G, T\}$ at random. Each sequence $\mathbf{X}$ is then encoded with a NB-LDPC code, which outputs sequences $\mathbf{C}$ of length N in GF(4). Then, for each sequence $\mathbf{C}$, a fixed number τ of deletions are introduced at random positions onto $\mathbf{C}$, where $\tau \in [\![1, 4]\!]$. In this set of simulation, we consider a NB-LDPC code of length $N = 200$ in GF(4), and different code rate values: $R = \frac{1}{4}$ (*i.e.*, $K = 50$), $R = \frac{1}{2}$ (*i.e.*, $K = 100$), $R = \frac{3}{4}$ (*i.e.*, $K = 150$), and $R = \frac{9}{10}$ (*i.e.*, $K = 180$). The segments length during the synchronization step is fixed to $l_s = 5$. For each couple (τ,R) we consider 1000 simulation runs to evaluate the Symbol Error Rate (SER) and FER of the synchronization algorithm and NB-LDPC decoding solution.

Figures 4.10 and 4.11 show the obtained SER and FER, respectively, obtained after synchronization and NB-LDPC decoding. We observe that the code rates $R = \frac{1}{4}$ and $R = \frac{1}{2}$ always allow to correct one deletion error, while for higher code rates, the probability of error becomes important. Furthermore, when there are two deletions, the synchronization and NB-LDPC decoding error rate is important no matter the code rate. This high error

rate obtained under several deletions is related to two factors. The first one is related to the process of transforming deletions into one or several substitutions. In this case, multiple deletions introduce a high amount of substitutions that cannot be handled by such short NB-LDPC code ($N = 200$). The second factor is related to the position of the deletions when there are several. Indeed, if the deletions are not on the same segment of length l_s, the first factor makes several parity-check equations incorrect because of the substitutions. Thus, when executing the synchronization algorithm in a greedy manner, the arbitrary base A is not introduced as close as possible of the deletion error.

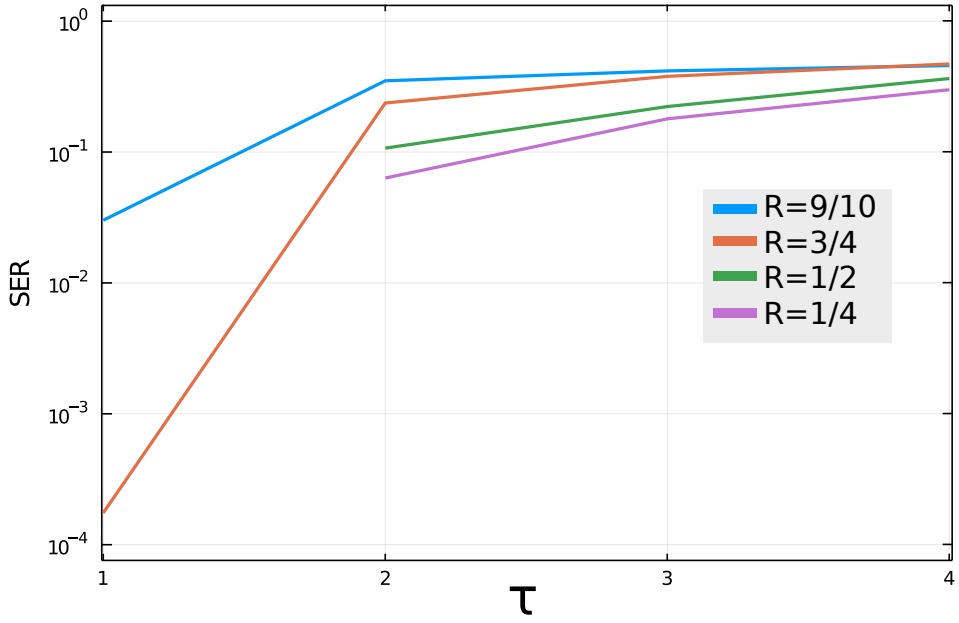


Figure 4.10: This figure shows the synchronization and NB-LDPC decoding SER given a number of deletion.

4.8.2 CSL solution evaluation

We considered sequences $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(3)}$ among the nine sequences from Set$E$, and encoded the corresponding sequences with a regular $(3, 6)$ NB-LDPC code in GF(4) of size $(K = 500, N = 1000)$ and code rate $R = 1/2$, constructed from a PEG algorithm [114]. We further set the segment length to $l_s = 50$ for our synchronization method. To evaluate the proposed reconstruction method, we considered three setups: (i) consensus alone, (ii) consensus + NB-LDPC decoder without re-synchronization, (iii) consensus + re-

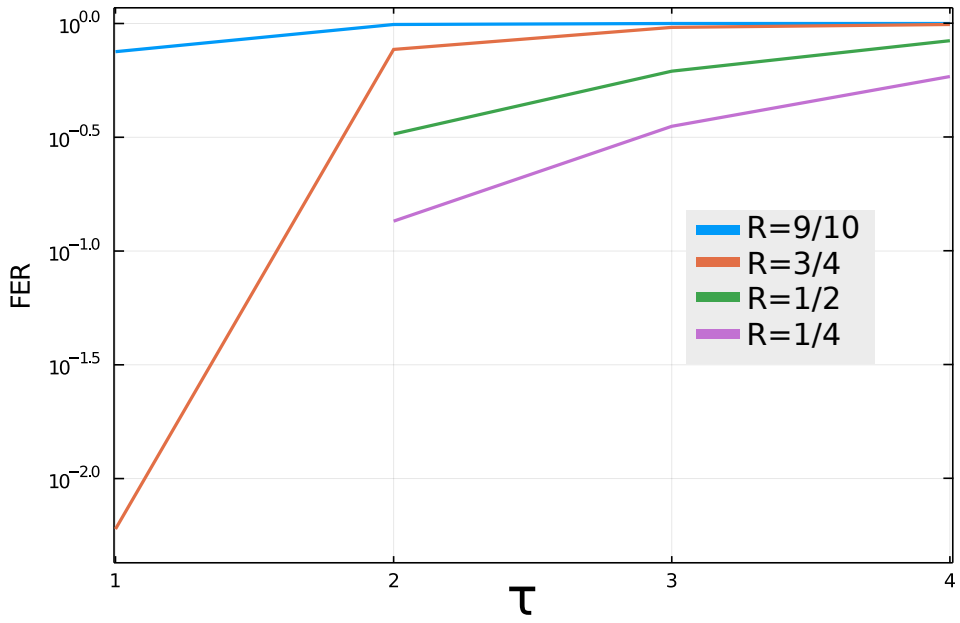


Figure 4.11: This figure shows the synchronization and NB-LDPC decoding FER given a number of deletion.

synchronization + NB-LDPC decoder. For each of these setups, we applied the successive reconstruction steps 900 times, and evaluated the proportion of perfectly recovered sequences, called "success probability". This metric is of interest in our setup, since the condition $\mathbf{c}.\mathbf{H}^T = \mathbf{0}$ allows to check whether the original sequence was correctly retrieved, and to stop the reconstruction accordingly.

Figure 4.12 shows the success probability for the two considered sequences $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(3)}$, with respect to the number M of input sequences to the consensus. For both sequences, as expected, the success probability increases with M, before reaching a peak at around $M = 100$. The decrease in performance after this peak probably comes from the fact that the consensus has difficulties to handle too many sequences, due to the initial majority voting operation. Then, we observe two different configurations, depending on the considered sequence. For $\mathbf{x}^{(1)}$, it seems that the synchronization algorithm and the NB-LDPC decoder are not useful, in the sense that the sequences at the output of the consensus are either correct, or very far from the original data. On the opposite, for $\mathbf{x}^{(3)}$, we observe that the synchronization algorithm greatly improves the success probability, while NB-LDPC decoder alone after consensus does not help. This is probably due to the fact that for $\mathbf{x}^{(3)}$, most sequences at the output of the consensus contain a few amount of deletions

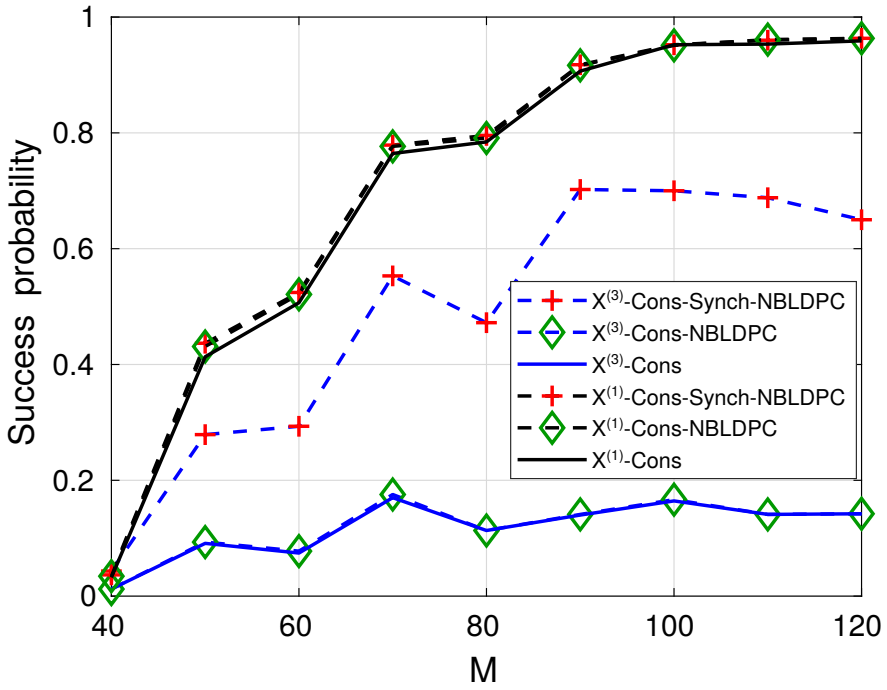


Figure 4.12: This figure represents the proportion of correctly retrieved sequences with respect to the number M of input sequences to the consensus.

introduced by an homopolymer of length 6, which are transformed into substitutions by the synchronization method.

4.8.3 The CSL solution in the full DnarXiv pipeline

The CSL solution was then fully integrated in the DnarXiv pipeline as shown in Figure 4.13. Olivier Boullé and Dominique Lavenier are in charge of the plateform which aims to numerically simulate all together the different part of the DnarXiv project: Biotechnology, Bioinformatics, Security and Coding. They performed some simulations to test the pipeline. To do so, they encoded an image equivalent to 150k bases with three different methods:

— Method Cons100: The consensus was used without channel coding, and the 150k bases were fragmented into $I = 1700$ fragments $f^{(i)}_{\text{Cons100}}$ ($i \in [\![1, I]\!]$) of size $N_{\text{Cons100}} = 100$ bases.

— Method Cons200: The consensus was used without channel coding, and the 150k bases were fragmented into $I = 800$ fragments $f^{(i)}_{\text{Cons200}}$ ($i \in [\![1, I]\!]$) of size $N_{\text{Cons200}} = 200$ bases.

— Method CSL200: The consensus was used with channel coding. This method used the CSL solution with a code rate $R = \frac{1}{2}$ for the NB-LDPC code. Due to the code rate R, 300k bases were obtained after the encoding. These 300k bases were then fragmented into $I = 1700$ fragments $f^{(i)}_{\text{CSL100}}$ ($i \in [\![1, I]\!]$) of size $N_{\text{CSL200}} = 200$ bases.

For each of these methods, the fragments contain an index part of 11 bases, and a payload part which corresponds to 89 bases and 189 bases when $N = 100$ and $N = 200$ respectively. Then the resulting fragments of each method were randomly assembled into DNA molecules ξ_{Cons100}, ξ_{Cons200}, and ξ_{CSL200} of length equal to ten times ($\times 10$) the fragment length of the considered method. For instance,

$$\xi^{(j)}_{\text{CSL100}} = [\underbrace{f^{(64)}_{\text{CSL200}}}_{1}, \underbrace{f^{(15)}_{\text{CSL200}}}_{2}, ..., \underbrace{f^{(124)}_{\text{CSL200}}}_{9}, \underbrace{f^{(15)}_{\text{CSL200}}}_{10}],\ j \in [\![1, 50000]\!]$$

is the j^{th} DNA molecule, which contains 10 fragments selected at random, and the fragment $f^{(15)}_{\text{CSL200}}$ was selected two times. The molecules are then extracted (selected) before going through the memory channel model. The resulting simulated molecules are then fragmented again into fragments $\tilde{f}^{(i)}_{\text{Cons100}}$, $\tilde{f}^{(i)}_{\text{Cons200}}$, $\tilde{f}^{(i)}_{\text{CSL100}}$, which are edited versions of the original fragments. These fragments are then clustered into clusters $O(i)$, where $i \in [\![1, I']\!]$

and $I' \leq I$. Then, the decoding is applied to each cluster $O(i)$. In methods Cons200 and Cons100, the decoding is done with the CCSA algorithm only. In CSL200, the decoding is performed with the full proposed CSL solution.

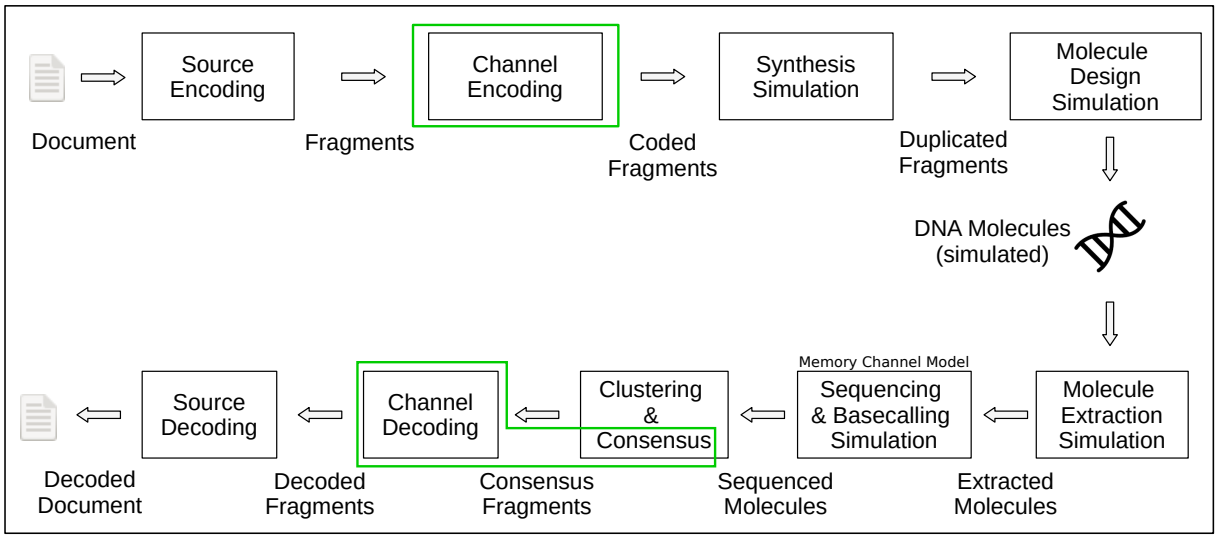


Figure 4.13: This figure represents the complete pipeline of storage and extraction developped in the project DnarXiv. The parts specific to the CSL coding scheme are highlighted in green. Note that our memory channel model is used to perform the sequencing and basecalling simulations.
Credit: Figure provided by Olivier Boullé and Dominique Lavenier.

Figure 4.14 shows a comparison between the three methods. The x-axis corresponds to the number of DNA molecule reads, and the y-axis corresponds to the precision of the sequences. The precision means the percentage of correctly aligned and equal bases between the reconstructed fragment and the original one. The best method is the one which has a high precision with fewer reads.

Unfortunately, it is disappointing to see that Cons200 and Cons100 methods have better results than CSL200 method which includes channel coding. However, these results have several explanations:

— **The amount of information (without redundancy) contained on each DNA molecule**. In Cons200 method, there is no channel coding. Hence, each DNA molecule contains two times more information than CSL200 method, which has a code rate $R = \frac{1}{2}$. Therefore, in Cons200 the majority of clusters $O(i)$ are retrieved faster and with enough fragments $\tilde{f}^{(i)}_{\text{Cons200}}$ to reconstruct a consensus sequence $\overline{\overline{f^{(i)}_{\text{Cons200}}}}$ with the CCSA. In CSL200, some clusters $O(i)$ are never retrieved,

or they have not enough fragments $\tilde{f}^{(i)}_{\text{CSL200}}$ to reconstruct a consensus sequence $\overline{\overline{f^{(i)}_{\text{CSL200}}}}$ before launching the synchronization and NB-LDPC decoding.

— **The consensus sequence reconstruction**. In CSL200 method, the consensus algorithm is a bottleneck, since if no consensus sequence can be reconstructed, it is not possible to launch the synchronization and NB-LDPC decoding. Thus, for DNA molecules ξ of the same length, to maximize the chances to retrieve all the $O(i)$ clusters, and thus, to reconstruct a consensus sequence $\overline{\overline{f^{(i)}}}$ for each fragment $f^{(i)}$, it is better to increase the number I of fragments per DNA molecule than to decrease the code rate R.

— **The fragments length**. When considering the same number M of input sequences, the performance of the consensus algorithm is better on short sequences than on long ones. In this case, although Cons100 and CSL200 methods contains the same amount of information, because of their redundancy, we have that $N_{\text{CSL200}} = 2 \times N_{\text{Cons100}}$. Therefore, the CCSA algorithm outputs a consensus sequence $\overline{\overline{f^{(i)}}}$ for much more clusters $O(i)$ in Cons100 method than in CSL200 method.

Therefore, in the case where the number M of read DNA molecules with same length N is fixed, it is better to increase the number I of fragments per DNA molecule than to decrease the code rate R. Hence, the CSL solution is not well adapted for this case. However, in some cases, the CSL solution can be useful as shown on Figure 4.12. In this case, the number M of read fragments $\tilde{f}^{(i)}$ in each cluster $O(i)$ is the same for all the methods.

4.9 Conclusion

In this chapter, we proposed a DNA data storage coding scheme, based on combining a consensus algorithm with a NB-LDPC code. We also proposed a synchronization algorithm, which allows to re-synchronize the consensus sequences by transforming its deletion errors into substitution errors. Through numerical simulations, we observed that the synchronization algorithm allows to improve the decoding performance without adding periodical markers. However, when there are multiple deletions the performance of the synchronization algorithm starts to decrease. Furthermore, this coding scheme is highly dependent of the consensus step, which needs a hundred of sequences to reconstruct a consensus sequence. Thus, when it is not possible to reconstruct a consensus sequence,

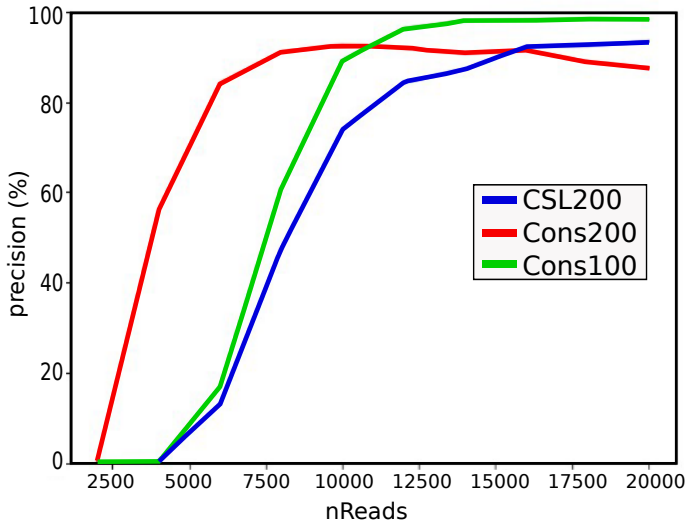


Figure 4.14: This figure shows a comparison between the methods Cons200, Cons100, and CSL200. The x-axis corresponds to the number of DNA molecules reads, and the y-axis corresponds to the precision of the outputs. Credit: Figure provided by Olivier Boullé and Dominique Lavenier.

the synchronization and NB-LDPC decoding cannot be used, which clearly penalizes this solution. This is why in the next chapter, we introduce another coding scheme to replace the consensus step. As a main feature, this second scheme only needs one or a few sequences to correctly decode the input sequence. Furthermore, unlike the CSL solution, it can offer a tradeoff between the code rate R and the number of sequence M that need to be used for decoding.

Chapter 5

Second error-correction solution: Convolutional Codes with decoder aware of the channel memory

In the previous chapter, we presented a first ECC solution called CSL, which combines a consensus algorithm, a synchronization algorithm, and a NB-LDPC decoder. Although the synchronization algorithm allows to improve the decoding performance, the numerical results showed that the performance is driven by the consensus algorithm, which does not exploit coding. Therefore, we now introduce another error-correction solution based on Convolutional Codes (CCs), which replaces the consensus step. We consider a particular convolutional decoder initially proposed in [115] which allows to correct synchronization and substitution errors. We then propose to augment the considered convolutional decoder in order to take into account the memory channel model introduced in Chapter 3 so as to improve the decoding performance.

In this chapter, we first describe standard CCs, and then review existing convolutional decoding solutions which can handle synchronization errors. We then introduce our augmented convolutional decoder solution, which takes into account our memory channel model. Finally, we compare existing solutions to our solution through Monte-Carlo simulations.

5.1 Standard Convolutional Codes

Unlike block codes (LDPC codes, Polar codes,...), CCs [117, 116, 27] can encode blocks of arbitrary length. Furthermore, the CC sends the parity-check bits, rather than the parity-check bits and the information bits, as in block codes.

5.1.1 Notation

In what follows we use:

— k_c to denote the number of input bits.

— n_c to denote the number of output bits.

— K_c to denote the constraint length, such that to encode k_c bits into n_c bits, up to K_c bits can be involved in the parity-check equations. The constraint length can be seen as a sliding window, which slides by k_c bits after each encoding.

— δ_{poly} to denote the generator polynomial, which provides the equations to generate the n_c output bits from the k_c input bits. Thus, the number of polynomials on the generator polynomial, is equal to n_c. Since we consider a binary CC, all the arithmetic operations are modulo 2.

— $\mathbf{y}_{\mathcal{B}}^{\mathcal{E}}$ to denote the substring of $\mathbf{y}$ that starts at position $\mathcal{B}$ and ends at position $\mathcal{E}$. If $\mathcal{E} < \mathcal{B}$, the substring $\mathbf{y}_{\mathcal{B}}^{\mathcal{E}}$ is empty, *i.e.*, $\mathbf{y}_{\mathcal{B}}^{\mathcal{E}} = \emptyset$.

5.1.2 CC representation

A (k_c, n_c, K_c) CC code has a code rate $R = \frac{k_c}{n_c}$, and can be represented either with a diagram, or with a state machine. Let us assume a $(1, 2, 3)$ CC with polynomial generator $\delta_{poly} = [\delta^2 + 1, \delta^2 + \delta + 1]$. The diagram representation for this CC is shown on Figure 5.1. In this figure, the value of the two memory registers in grey define the current state S_i of the encoder, with $i \in [\![0, 2^{K_c - 1} - 1]\!]$.

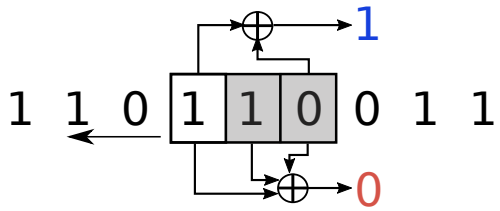


Figure 5.1: $(1, 2, 3)$ CC diagram representation. The generator polynomial is $\delta_{poly} = [\delta^2 + 1, \delta^2 + \delta + 1]$. The first output bit in blue is obtained from the first polynomial $\delta^2 + 1$, and the second output bit in red is obtained from the second polynomial $\delta^2 + \delta + 1$. The leftmost bit is the most recent input bit. The grey boxes are memory registers, which retain older input bits.

Although the diagram representation is useful to understand the CC structure, a CC can be represented in a more convenient way by a state machine, which will be used

during the CC decoding. In the state machine, the number of states is equal to 2^{K_c-1}, and the states are denoted by S_i. In order to avoid confusion later in this chapter, we also refer to these states as internal CC states. The state machine of the $(1, 2, 3)$ CC is shown in Figure 5.2. The initial state is always set to $S_0 = 00$ *i.e.*, the memory registers are initialized to zero. The transition from a state to another state is possible if there is an edge which connects the two states. Each edge is labeled with a $w_t/c_1..c_{n_c}$ notation, where w_t is the t^{th} input bit that allows the transition from state S_i to state $S_{i'}$, and this transition produces an output sequence $c_1..c_{n_c}$.

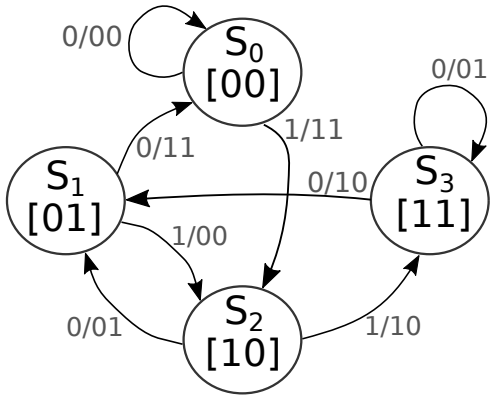


Figure 5.2: $(1, 2, 3)$ CC state machine representation. The polynomial generator is $\delta_{poly} = [\delta^2 + 1, \delta^2 + \delta + 1]$.

5.1.3 Convolutional decoder

The decoder only observes received sequence $\mathbf{y}$ which may contain corrupted bits. Its task is to infer the most likely state sequence $\mathbf{S}$ that produced the received sequence $\mathbf{y}$. Furthermore, due to the convolutional encoding, each bit y_t only depends on the current state s_t and the next state s_{t+1}. Therefore, the inference process is based on Hidden Markov Models [68]. The decoding process also relies on a trellis structure, as shown in Figure 5.3. The trellis structure is derived from the state machine, and it shows the evolution of the state machine through time. The rows of the trellis correspond to the whole possible states of the state machine, and each column represents a specific instant t, with $t \in [\![0, \frac{N}{n_c}]\!]$, N being the received sequence length. Decoding over the trellis aims to estimate s_t as the most likely state at instant t. The bit value $\hat{w}_t$ is then deduced from s_t and s_{t+1}. Figure 5.3 illustrates an example where the received sequence $\mathbf{y}$ contains a substitution error highlighted in blue, and the decoder infers the most likely sequence $\mathbf{S}$

that produced $\mathbf{y}$. As a convention, the trellis path always starts and ends at the state S_0. Thus, $K_c - 1$ zeros are added to the encoded sequence $\mathbf{c}$ in order to flush the memory registers and put the internal CC state back to S_0. The added zeros are ignored after the decoding. The example in the figure assumes a Viterbi decoding algorithm [118], which selects the best path at each instant t to find a path with maximum likelihood, *e.g.*, which maximizes $\mathbb{P}(\mathbf{S}|\mathbf{y})$. Figure 5.3 also shows the estimated sequence $\hat{\mathbf{w}}$ corresponding to the path of maximum likelihood. However, in this work, rather than the Viterbi algorithm, we consider the BCJR decoding algorithm [27, 119] named after its authors Bahl, Cocke, Jelinek and Raviv. Unlike the Viterbi algorithm, the BCJR maximizes the *A posteriori* Probability (MAP) $\mathbb{P}(s_t|\mathbf{y})$. Although it is more complex than the Viterbi algorithm, the BCJR algorithm has better performance [120]. We explain later in this chapter how the BCJR algorithm works.

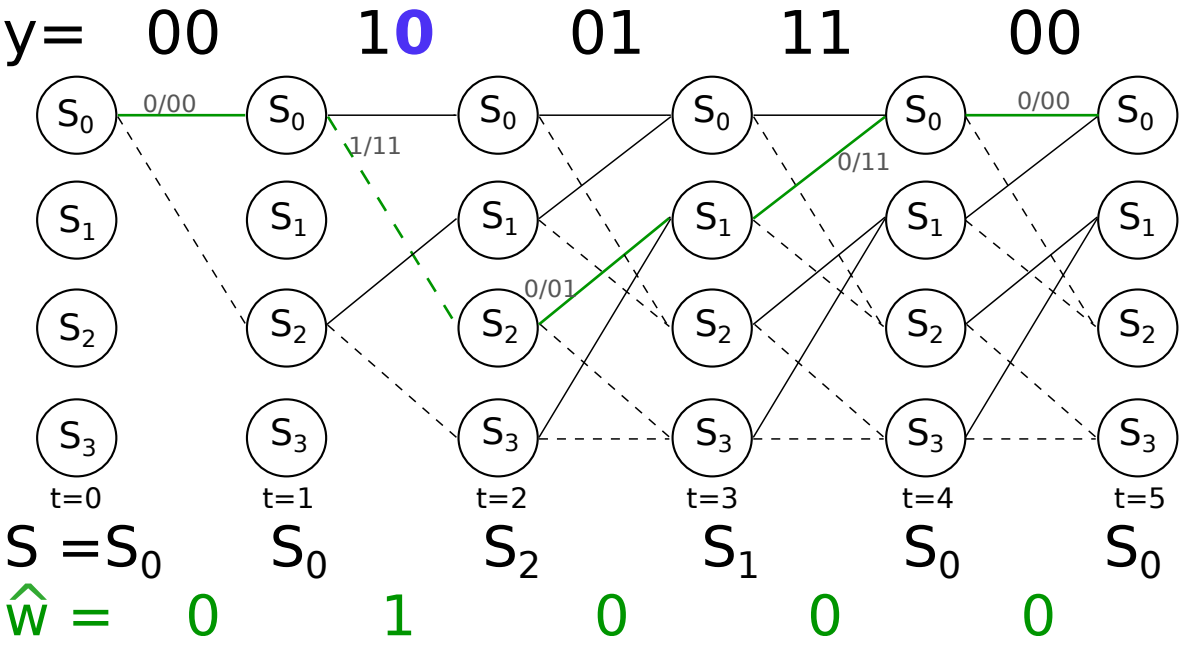


Figure 5.3: This figure shows a trellis on which the decoding process is performed for a $(1, 2, 3)$ CC. The polynomial generator is $\delta_{poly} = [\delta^2 + 1, \delta^2 + \delta + 1]$. The bit highlighted in blue in $\mathbf{y}$ is corrupted because of a substitution. The decoding process uses a Viterbi decoding, where at each instant t the best path is selected and highlighted in green.

We now introduce the CC decoding scheme which tackles synchronization and substitution errors.

5.2 CC coding scheme for synchronization errors

In this work, we consider the CC construction which was first introduced in [115] and later considered in [101, 121, 122], to correct insertion, deletion, and substitution errors. The performance of this scheme can be further improved by considering a concatenated code construction. The concatenated code construction considers a CC as inner code, so as to correct synchronization errors and most substitution errors, and a LDPC [101] as outer code, to correct the remaining substitution errors. In [121] to further improve the bit error rate of the CC decoder, it was proposed to sum a random sequence, called offset, with the encoded sequence, so as to create a dependency between the decoded sequence and the states s_t, $t \in [\![0, \frac{N}{n_c}]\!]$.

However, all these existing CC constructions target i.i.d. channels. Since the DNA data storage channel is not an i.i.d. channel, to improve the decoding performance in terms of BER and FER, we propose to take into account our memory channel model. In [101], the inner CC allows to correct most of the insertion, deletion, and substitution errors, while the outer LDPC code only corrects residual substitution errors. Therefore, it is more critical to first improve the performance of the inner CC, which has a stronger influence on the final reconstruction performance. This is why in this work we only work on improving the performance of the inner CC, and do not consider the concatenated code construction of [101]. We aim to propose an augmented version of the convolutional decoder, which takes into account our memory channel model so as to improve the decoding performance. In this section, we describe the main steps of the considered CC scheme. We first introduce our notation for the encoding part and the decoding part of the coding scheme, which is described in Figure 5.4. For simplicity we consider that all the sequences are in GF(4).

5.2.1 Encoder part

As input to the convolutional encoder, we consider a sequence $\mathbf{w}$ of length N_0. From $\mathbf{w}$, the convolutional encoder outputs an encoded sequence $\mathbf{c}$ of length N. An offset sequence $\mathbf{r}$ is then added to $\mathbf{c}$. The sequence $\mathbf{r}$ is a random sequence of length N. This provides a sequence $\mathbf{x} = \mathbf{c} \oplus \mathbf{r}$, where the sum is in GF(4). It was shown in [121] that in the case of a channel with synchronization errors, and under some assumptions, the offset sequence improves the decoder performance by reducing the BER by a factor of 10^{-2}. The sequence $\mathbf{x}$ is then stored as a DNA molecule after synthesis. In this work we consider a binary convolutional code. Thus, the input and output sequences of the

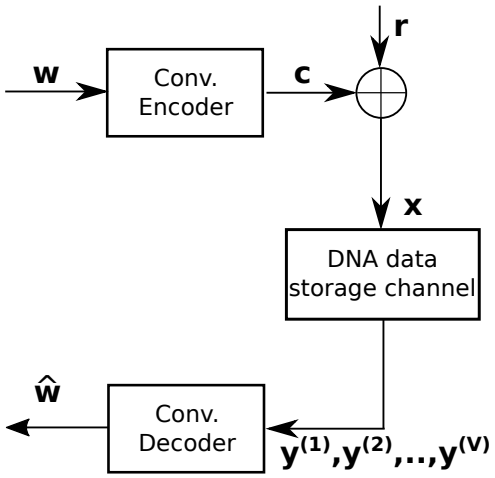


Figure 5.4: CC coding scheme for synchronization errors.

decoder are respectively converted into binary and into GF(4) elements. The conversion process transform each couple of bits into an element of GF(4) (and *vice versa*), with the convention "00" $\rightarrow$ 0, "01" $\rightarrow$ 1, "10" $\rightarrow$ 2 and "11" $\rightarrow$ 3.

5.2.2 Decoder part

After sequencing, the DNA storage channel outputs V sequences $(\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(V)})$, where $\mathbf{y}^{(v)}$ is of length $N^{(v)}$, $v \in [\![1, V]\!]$. We first describe how to apply the CC decoder to only one sequence $\mathbf{y} = \mathbf{y}^{(v)}$. At the end of the next section, we also discuss how to perform the decoding over V' $(V' < V)$ sequences $\mathbf{y}^{(v')}$. Note that the offset sequence is taken into account directly inside the convolutional decoder. Although $\mathbf{w}$, $\mathbf{x}$ and $\mathbf{y}^{(v)}$ are in GF(4), for simplicity, in the convolutional decoder we consider $\mathbf{w}$, $\mathbf{x}$ and $\mathbf{y}^{(v)}$ as binary sequences. We also use N to denote the length of the binary sequence $\mathbf{x}$.

We now introduce the standard BCJR algorithm [27, 119], since an extended version of this algorithm will be considered in our decoder.

5.2.3 BCJR algorithm

The BCJR decoding algorithm [27, 119] allows to estimate the input symbols w_t. In a binary case, this estimation is performed with the *a posteriori* likelihood ratios

$$\mathcal{L}(w_t|\mathbf{y}) = \log\left(\frac{\mathbb{P}(w_t = 1|\mathbf{y})}{\mathbb{P}(w_t = 0|\mathbf{y})}\right) \tag{5.1}$$

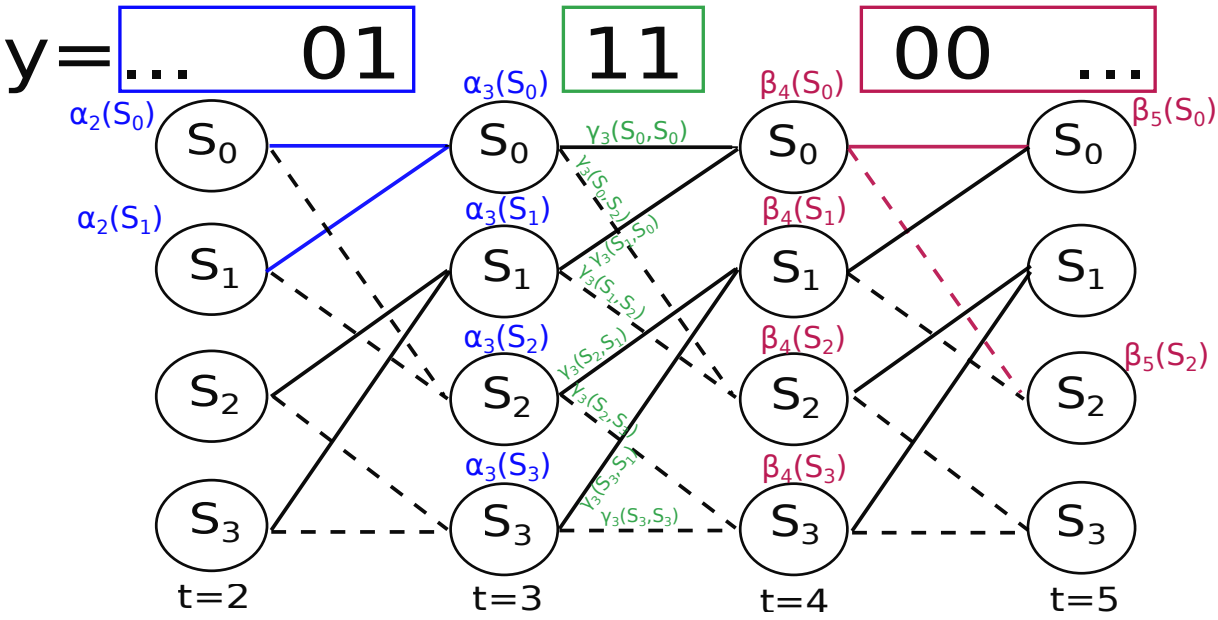


Figure 5.5: BCJR algorithm example on a partial trellis diagram, for time instants $t \in [\![2, 5]\!]$.

The value of w_t is estimated according to the sign of the log-likelihood ratio: if $\mathcal{L}(w_t|\mathbf{y}) > 0$, then it is estimated that $\hat{w}_t = 1$, and if $\mathcal{L}(w_t|\mathbf{y}) < 0$, then it is estimated that $\hat{w}_t = 0$. The absolute value of the log-likelihood indicates the confidence level of the estimation. Using the Bayes rule, the *a posteriori* probability $\mathbb{P}(w_t|\mathbf{y})$ can be written as

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

where the probability $\mathbb{P}(\mathbf{y})$ does not depend on w_t and can be considered as a constant term ignored in the computation.

The value of the input bit w_t defines the transition from s_t to s_{t+1}, denoted (s_t, s_{t+1}). Some possible transitions are shown in Figure 5.5, where $w_t = 1$ corresponds to the dashed lines, and $w_t = 0$ corresponds to the solid lines. Moreover, at each instant t, only one unique transition (s_t, s_{t+1}) is possible. Therefore, the probabilities $\mathbb{P}(w_t = 0|\mathbf{y})$ and $\mathbb{P}(w_t = 1|\mathbf{y})$ can be expressed as

$$\mathbb{P}(w_t = 0|\mathbf{y}) = \sum_{(s_t, s_{t+1}):w_t=0} \mathbb{P}(s_t, s_{t+1}|\mathbf{y})$$

and

$$\mathbb{P}(w_t = 1|\mathbf{y}) = \sum_{(s_t, s_{t+1}):w_t=1} \mathbb{P}(s_t, s_{t+1}|\mathbf{y})$$

where $\sum_{w_t} \mathbb{P}(s_t, s_{t+1})$ is the sum of all transition probabilities (s_t, s_{t+1}) allowed by the value w_t. The *a posteriori* log-likelihood ratio can then be expressed as

$$\mathcal{L}(w_t|\mathbf{y}) = \log\left(\frac{\sum_{(s_t,s_{t+1}):w_t=1} \mathbb{P}(s_t, s_{t+1}, \mathbf{y})}{\sum_{(s_t,s_{t+1}):w_t=0} \mathbb{P}(s_t, s_{t+1}, \mathbf{y})}\right)$$

Furthermore, thanks to the Hidden Markov Models inference process [68], the term $\mathbb{P}(\mathbf{y}, s_t, s_{t+1})$ can be decomposed and efficiently estimated as

$$\begin{aligned}\mathbb{P}(\mathbf{y}, s_t, s_{t+1}) =& \mathbb{P}(\mathbf{y}_1^{(t-1).n_c}, s_t)\mathbb{P}(\mathbf{y}_{(t-1).n_c+1}^{t.n_c}, s_{t+1}|s_t)\mathbb{P}(\mathbf{y}_{t.n_c+1}^{N}|s_{t+1}) \\ =& \alpha_t(s_t)\gamma_t(s_t, s_{t+1})\beta_{t+1}(s_{t+1})\end{aligned}$$

The term $\alpha_t(s_t)$ refers to a forward recursion, which allows to recursively compute $\mathbb{P}(\mathbf{y}_1^{(t-1).n_c}, s_t)$. For instance, in Figure 5.5 the partial sequence $\mathbf{y}_1^{(t-1).n_c}$ at $t = 4$ is highlighted in blue. It is also shown how $\alpha_3(s_0)$ is recursively computed from the alpha values $\alpha_2(s_0)$ and $\alpha_2(s_1)$ of the previous nodes at instant $t = 2$.

The term $\beta_{t+1}(s_{t+1})$ refers to a backward recursion, which allows to recursively compute $\mathbb{P}(\mathbf{y}_{t.n_c+1}^{N}|s_{t+1})$. For instance, Figure 5.5 shows the partial sequence $\mathbf{y}_{t.n_c+1}^{N}$ at $t = 4$ in red. The figure also shows how $\beta_4(s_0)$ is recursively computed from the beta values $\beta_5(s_0)$ and $\beta_5(s_2)$ of the next nodes at $t = 5$.

The term $\gamma_t(s_t, s_{t+1})$ is called a branch metric $\mathbb{P}(\mathbf{y}_{(t-1).n_c+1}^{t.n_c}, s_{t+1}|s_t)$. Figure 5.5 shows the partial sequence $(\mathbf{y}_{(t-1).n_c+1}^{t.n_c}, s_{t+1}|s_t)$ at $t = 4$ in green.

In what follows, we first introduce the convolutional decoder of [101, 121]. This decoder is based on a modified BCJR algorithm which allows to correct not only substitutions, but also insertions and deletions. We then show how to modify this convolutional decoder so as to consider all the knowledge provided by our memory channel model. Especially, we will take into account the memory introduced by previous events and by the k-mers.

5.3 CC decoder for synchronization errors

When designing the convolutional decoder, the first difficulty resides in the fact that not only substitutions, but also insertions and deletions should be corrected, as shown in Figure 5.6. Especially, it was observed in [101] that insertions and deletions break

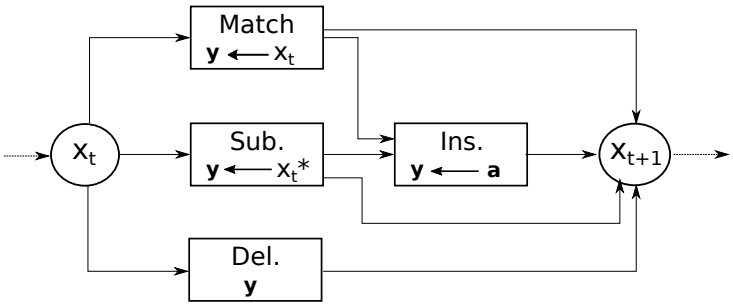


Figure 5.6: DNA data storage channel model.

the Markov property in the sequence of decoder internal states. In what follows, we first describe the convolutional decoder which was proposed in [101, 121] to correct the three types of errors for i.i.d. channel models. Especially, this decoder introduces an additional drift variable, which restores the Markov property.

5.3.1 state-of-the-art CC decoder with drifts (*Dec1*)

We now describe the CC decoder which was introduced in [115, 121] and later considered in [101] to correct both synchronization and substitution errors.

States of the decoder

The successive internal states of the CC are denoted s_t, where $t \in [\![0, T]\!]$, and $T = N/n_c$. For instance, if the CC has 4 states, s_t takes values in $\{S_0, S_1, S_2, S_3\}$. Further, [115, 123] 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 $d_t = \text{Nb(INS)}_t - \text{Nb(DEL)}_t$, 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 $t = T$, we assume that d_t lies in the interval $[\![D_{\min}, D_{\max}]\!]$. Both $I_{\max}$, $D_{\min}$, and $D_{\max}$, will be parameters of the decoder. Finally, the state of the decoder is denoted by the pair $\boldsymbol{\sigma}_t = (s_t, d_t)$. Figure 5.7 shows the trellis of the decoder, with state $\boldsymbol{\sigma}_t$ evolving between successive time instants $t = 0$, $t = 1$, and $t = 2$.

For instance, let us assume a sequence

$$\mathbf{x} = [A\mathbf{T}ACGTC]$$

sent through the DNA data storage channel, and a received sequence

$$\mathbf{y} = [AACGTC]$$

In this example, the channel has deleted the symbol T highlighted in red on $\mathbf{x}$. In Figure 5.7 we show the subsequences $\mathbf{y}_{\mathcal{B}}^{\mathcal{E}}$ considered on each branch. The solid lines correspond to the input bit $w_t = 0$, while the dashed lines correspond to the input bit $w_t = 1$. We now describe the different subsequences observed on the path highlighted in blue in the trellis. We observe that at instant $t = 0$ the blue branch which goes from $\boldsymbol{\sigma}_0 = (S_0, 0)$ to $\boldsymbol{\sigma}_1 = (S_0, 0)$ corresponds to the substring $\mathbf{y}_1^2 = A(A = 00)$. This is because $d_0 - d_1 = 0$, and thus, on this branch it is assumed that there is no insertion nor deletion. Then, at instant $t = 1$, the blue branch which goes from $\boldsymbol{\sigma}_1 = (S_0, 0)$ to $\boldsymbol{\sigma}_2 = (S_0, -1)$ corresponds to the empty substring $\mathbf{y}_3^2 = \emptyset$. This is because $d_t - d_{t+1} = -1$, and thus, on this branch it is assumed that there is a deletion. Finally, at instant $t = 2$, the blue branch which goes from $\boldsymbol{\sigma}_2 = (S_0, -1)$ to $\boldsymbol{\sigma}_3 = (S_0, -1)$ corresponds to the substring $\mathbf{y}_3^4 = A$. In addition, the blue path is the most probable path, since it allows to resynchronize the sequence, *i.e.*, the 1^{st} A and 2^{nd} A are located on the same positions in the trellis than in the sent message $\mathbf{x}$. Of course, this is just to illustrate how the drift allow to synchronize the sequence. We now describe into more details how the different elements of the BCJR decoding algorithm are computed.

A posteriori probability computation

The decoder aims to compute *a posteriori* probabilities $\mathbb{P}(w_t|\mathbf{y})$, where

$$\mathbb{P}(w_t|\mathbf{y}) = \frac{\mathbb{P}(w_t, \mathbf{y})}{\mathbb{P}(\mathbf{y})} \tag{5.2}$$

and

$$\mathbb{P}(w_t, \mathbf{y}) = \sum_{(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}):w_t} \mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}). \tag{5.3}$$

During the computation of these probabilities, the drift variable d_t allows to maintain a Markov property in the sequence of states [121]. Hence, $\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$ can be decomposed as

$$\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \mathbb{P}(\mathbf{y}_1^{(t-1).n_c+d_t.n_c}, \boldsymbol{\sigma}_t)\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, \boldsymbol{\sigma}_{t+1}|\boldsymbol{\sigma}_t)\mathbb{P}(\mathbf{y}_{t.n_c+d_{t+1}.n_c+1}^{T}|\boldsymbol{\sigma}_{t+1}) \tag{5.4}$$

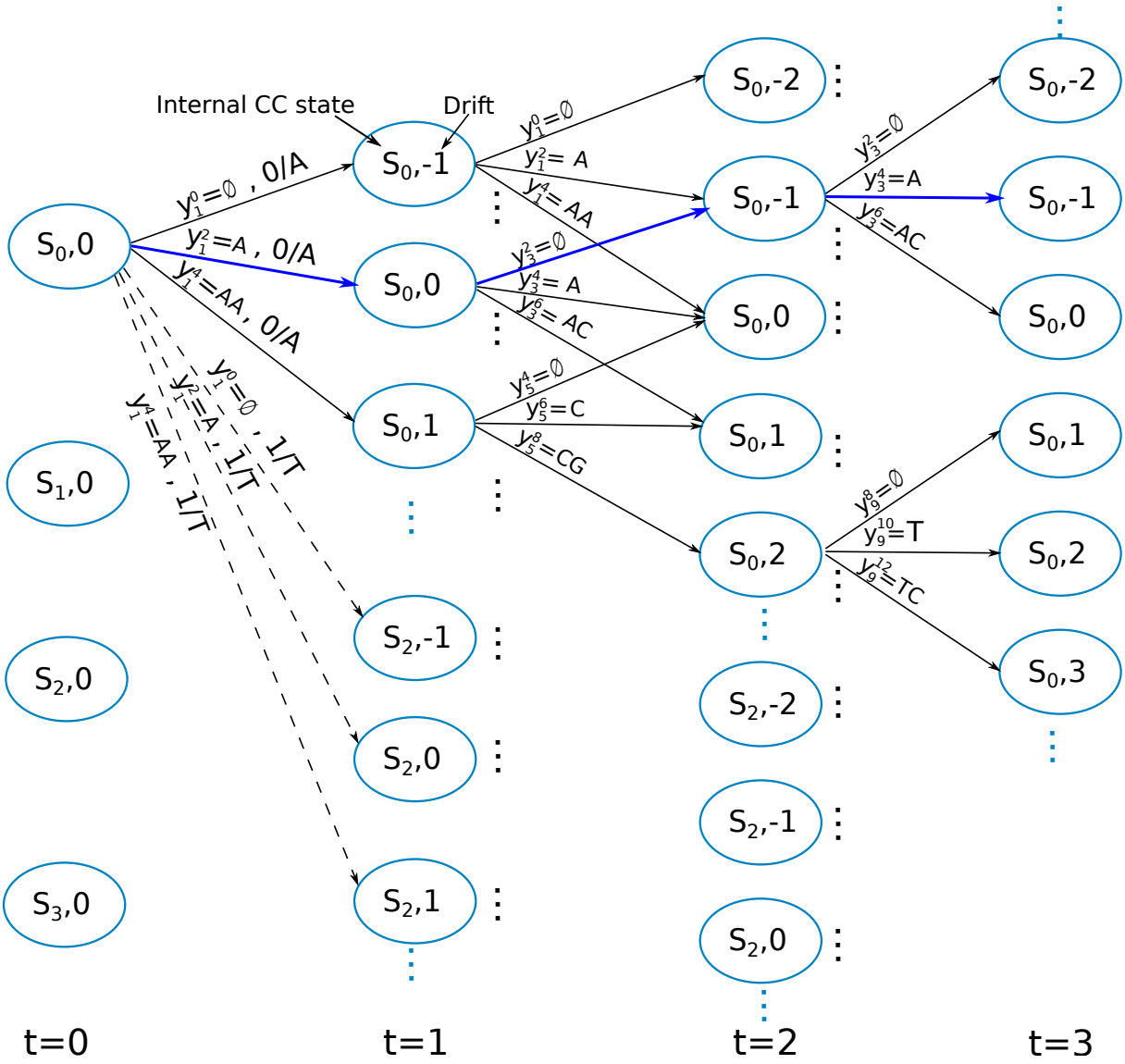


Figure 5.7: Partial trellis diagram for *Dec1*, for time instants $t \in [\![0,3]\!]$, with $I_{\max} = 1$ and $d_{t+1} \in [\![d_t - 1, d_t + 1]\!]$. It is assumed that the sequence $\mathbf{y} = [AACGTC]$ was received. The blue path is the most probable path, since it allows to resynchronize the sequence $\mathbf{y}$.

The three terms of the previous equation can be computed using a forward recursion for $\mathbb{P}(\mathbf{y}_1^{(t-1).n_c+d_t.n_c}, \boldsymbol{\sigma}_t) = \alpha_t(\boldsymbol{\sigma}_t)$, a backward recursion for $\mathbb{P}(\mathbf{y}_{t.n_c+d_{t+1}.n_c+1}^{T}|\boldsymbol{\sigma}_{t+1}) = \beta_{t+1}(\boldsymbol{\sigma}_{t+1})$, and a branch metric computation for $\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, \boldsymbol{\sigma}_{t+1}|\boldsymbol{\sigma}_t) = \gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$.

Forward and Backward recursions

The forward recursion allows to compute $\alpha_t(\boldsymbol{\sigma}_t)$ for all $t \in [\![0, T-1]\!]$ as

$$\alpha_t(\boldsymbol{\sigma}_t) = \sum_{\boldsymbol{\sigma}_{t-1}} \alpha_{t-1}(\boldsymbol{\sigma}_{t-1})\gamma_t(\boldsymbol{\sigma}_{t-1}, \boldsymbol{\sigma}_t) \tag{5.5}$$

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

$$\alpha_0(\boldsymbol{\sigma}_0) = \begin{cases} 1, & \text{if } \boldsymbol{\sigma}_0 = (0,0) \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

The backward recursion allows to compute $\beta_{t-1}(\boldsymbol{\sigma}_t)$ for all $t \in [\![1, T]\!]$ as

$$\beta_t(\boldsymbol{\sigma}_t) = \sum_{\boldsymbol{\sigma}_{t+1}} \beta_{t+1}(\boldsymbol{\sigma}_{t+1})\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) \tag{5.7}$$

where $\beta_T(\boldsymbol{\sigma}_T)$ is initialized as

$$\beta_T(\boldsymbol{\sigma}_T) = \begin{cases} 1, & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N) \\ 0, & \text{otherwise.} \end{cases} \tag{5.8}$$

Branch metric computation

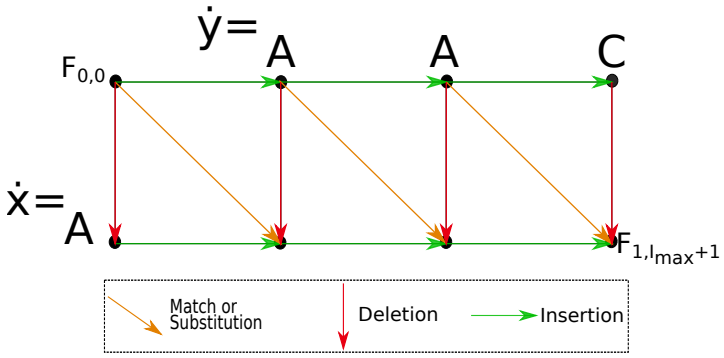


Figure 5.8: Lattice structure used to compute branch metrics in *Dec1*.

We also want to evaluate the branch metric $\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$, which can be expressed as

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1}|d_t, s_t, s_{t+1}) \tag{5.9}$$

where $\mathbb{P}(w_t) = 1/4$. The probability $\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t+1}^{t.n_c+d_{t+1}}, d_{t+1}|d_t, s_t, s_{t+1})$ is evaluated by using an efficient algorithm based on a lattice structure [124], see an example in Figure 5.8. This lattice will allow to compute the probability to pass from state s_t to state s_{t+1}, which corresponds to the emission of a certain symbol $\dot{x} = x_{(t-1).n_c+1}^{t.n_c}$. We consider the observed sequence $\dot{\mathbf{y}} = \mathbf{y}_{(t-1).n_c+d_t+1}^{t.n_c+d_{t+1}}$, whose length corresponds to passing from drift d_t to drift d_{t+1}. The lattice is equivalent to computing the probability to produce the observed sequence $\dot{\mathbf{y}}$ from $\dot{\mathbf{x}}$, by considering all the possible paths of the channel model in Figure 5.8. For instance, let us assume that the subsequence $\dot{\mathbf{x}} = A$ is sent through the channel (see Figure 5.6), and that the subsequence $\dot{\mathbf{y}} = AAC$ is the received sequence. One possible path that would produce $\dot{\mathbf{y}}$ from $\dot{\mathbf{x}}$ is as follows:

— The initial position is $F_{0,0}$.

— The first transition is a match, represented by the orange arrow. Hence, this match produces the first symbol A in the sequence $\dot{\mathbf{y}}$.

— The second transition is an insertion, represented by the the green arrow. This insertion produces the second symbol A in the sequence $\dot{\mathbf{y}}$.

— The third transition is an insertion, represented by the green arrow. This insertion produces the third symbol C in the sequence $\dot{\mathbf{y}}$.

Therefore, the probability to obtain $\dot{\mathbf{y}} = AAC$ from $\dot{\mathbf{x}} = A$ is evaluated by identifying all the possible path between points $F_{0,0}$ and $F_{1,I_{max}+1}$ in the lattice. We now describe into more details how the probability to produce $\dot{\mathbf{y}}$ from $\dot{\mathbf{x}}$ is recursively computed.

We recursively compute the probabilities on the lattice, as $\forall i \in [\![0, 1]\!]$, and $\forall j \in [\![0, I_{\max} + 1]\!]$,

$$F_{i,j} = \mathbb{P}_d F_{i-1,j} + \frac{1}{4}\mathbb{P}_i F_{i,j-1} + Q(\dot{x}_i|\dot{y}_j)F_{i,j-1} \tag{5.10}$$

where

$$Q(\dot{x}_i|\dot{y}_j) = \begin{cases} \mathbb{P}_m, & \text{if } \dot{x}_i = \dot{y}_j \\ \frac{1}{3}\mathbb{P}_s, & \text{otherwise.} \end{cases} \tag{5.11}$$

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 here,

these probabilities come from the *i.i.d.* channel model. The computation is initialized as

$$F_{i,j} = \begin{cases} 1, & \text{if } i = 0 \text{ and } j = 0 \\ 0, & \text{if } i < 0 \text{ or } j < 0 \end{cases} \tag{5.12}$$

$F_{i,j}$ represents the probability at the lattice node $[i, j]$ and it is computed recursively through the lattice using (5.10). Moving vertically on the lattice means that a deletion occurred, which corresponds to the first term of (5.10). Moving horizontally on the lattice means that an insertion occurred, which corresponds to the second term of (5.10), 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 $\dot{x}_i \neq \dot{y}_j$. In both cases, this corresponds to the third term of (5.10), 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

$$\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1} | d_t, s_t, s_{t+1}) = F_{1,d_{t+1}-d_t+1}.$$

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 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 $d_{t+1} - d_t$.

This CC decoder was shown to be very efficient when targeting i.i.d channels. However, as we saw in Chapter 3, error probabilities over the DNA data storage channel depend on the read k-mers and on previous observed events. Thus, the performance of *Dec1* may be improved by taking into account the DNA data storage channel. This is why we now propose to modify *Dec1* in order to take into account at first previous events (*Dec2*), and then both previous events and read k-mers (*Dec3*).

5.3.2 CC decoder taking into account previous events (*Dec2*)

We now extend the previous decoder with drifts in order to take into account the memory between successive events.

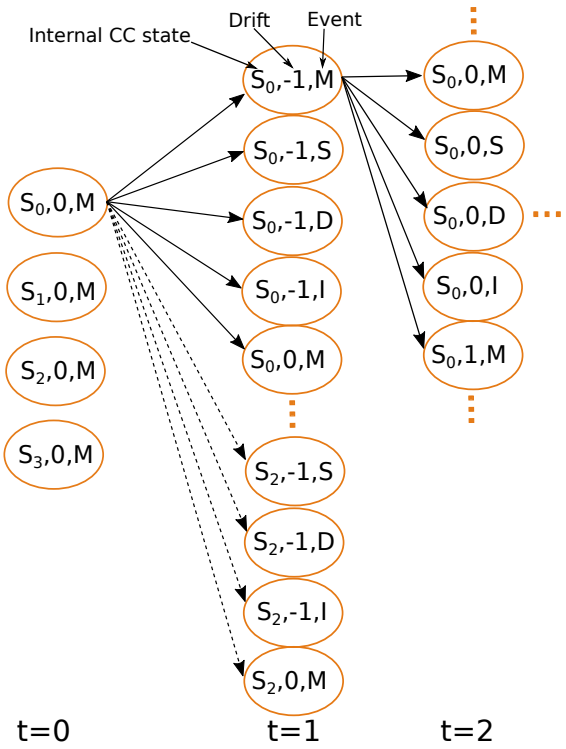


Figure 5.9: Partial trellis diagram for *Dec2*, for time instants $t = 0$, $t = 1$, $t = 2$, with $I_{\max} = 1$ and $d_{t+1} \in [\![d_t - 1, d_t + 1]\!]$.

States of the decoder

The state of the decoder is now given by a triplet $\boldsymbol{\sigma}_t = (s_t, d_t, e_t)$, where e_t is the last event (insertion, deletion, match, substitution) observed before emitting symbol x_t. The event variable e_t is added in order to preserve the Markov property when taking the previous event e_{t-1} into account. This results in a larger trellis which has 4 times more nodes than for *Dec1*, see figure 5.9 for an example. Note that at time instant, $t = 0$ we assume that the channel starts with a match "M".

A posteriori probability computation

With this new state definition, equation (5.4) still applies. To evaluate the three terms involved in the computation $\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$, we still use a forward recursion, a backward recursion, and a branch metric computation, each with modified expressions.

Forward and Backward recursions

The forward recursion is still given by (5.5), but the initialization is changed to also take into account the initial event e_0 which is defined as $e_0 = M$. This gives the following initialization:

$$\alpha_0(\boldsymbol{\sigma}_0) = \begin{cases} 1, & \text{if } \boldsymbol{\sigma}_0 = (0, 0, M) \\ 0, & \text{otherwise.} \end{cases} \tag{5.13}$$

In the same way, the backward recursion is still given by (5.7), but the initialization is changed so as to take into account the final event e_T as

$$\beta_T(\boldsymbol{\sigma}_T) = \begin{cases} \mathbb{P}_m, & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, M) \\ \mathbb{P}_s, & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, S) \\ \mathbb{P}_d, & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, D) \\ \mathbb{P}_i, & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, I) \\ 0, & \text{otherwise.} \end{cases} \tag{5.14}$$

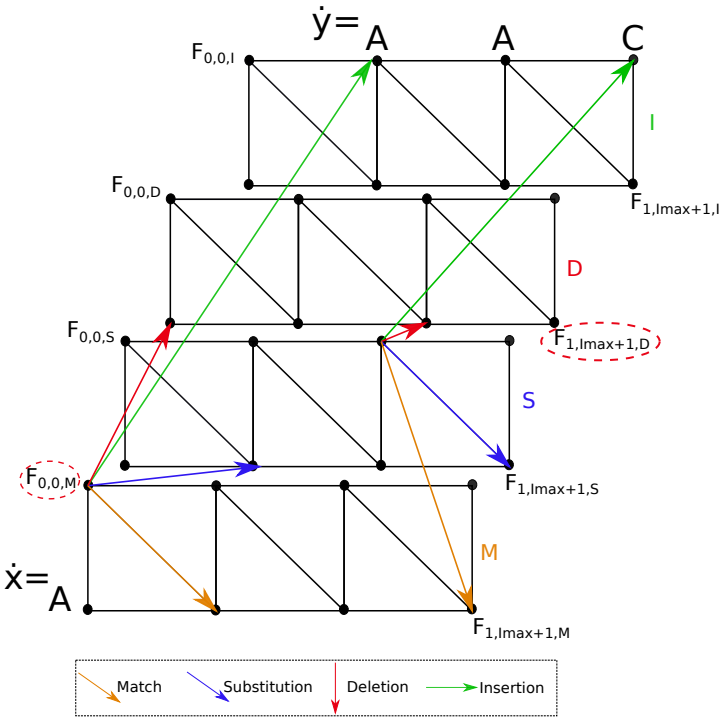


Figure 5.10: 3D Lattice structure used to compute branch metrics in *Dec2*, for $e_t = M$ and $e_{t+1} = D$.

Branch metric computation

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

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1}, e_{t+1}|d_t, e_t, s_t, s_{t+1}) \tag{5.15}$$

In this expression, the probability $\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1}, e_{t+1}|d_t, e_t, s_t, s_{t+1})$ is still evaluated recursively using a lattice. Especially, in *Dec2* a 3D lattice is necessary to consider additional paths added by the dependency to previous events. The new lattice contains four plans, where each plan represents a particular event $e_t \in \{M, S, D, I\}$, as shown in figure 5.10. The moving rules are the same as in the lattice for *Dec1* (see Figure 5.8), except for the fact that we can now move from a plan to another one, depending on the considered event e_t. For instance, in the case of an insertion, we should move horizontally from $[i,j-1]$ nodes related to each plan toward the insertion plan, see green arrows in Figure 5.10.

The recursive computation is initialized as

$$F_{i,j,e} = \begin{cases} 1, & \text{if } i = 0 \text{ and } j = 0 \text{ and } e = e_t \\ 0, & \text{if } i < 0 \text{ or } j < 0 \end{cases} \tag{5.16}$$

We now define $\mathbb{P}_{e_1 \to e_2}$ as the probability to observe event $e_2 \in \{M, S, D, I\}$ given that the previous event 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:

$$F_{i,j,M} = \mathbb{P}_{m\to m} F_{i-1,j-1,M} + \mathbb{P}_{s\to m} F_{i-1,j-1,S} + \mathbb{P}_{d\to m} F_{i-1,j-1,D} + \mathbb{P}_{i\to m} F_{i-1,j-1,I} \tag{5.17}$$

$$F_{i,j,S} = \frac{1}{3}(\mathbb{P}_{m\to s} F_{i-1,j-1,M} + \mathbb{P}_{s\to s} F_{i-1,j-1,S} + \mathbb{P}_{d\to s} F_{i-1,j-1,D} + \mathbb{P}_{s} F_{i-1,j-1,I}) \tag{5.18}$$

$$F_{i,j,D} = \mathbb{P}_{m\to d} F_{i-1,j,M} + \mathbb{P}_{s\to d} F_{i-1,j,S} + \mathbb{P}_{d\to d} F_{i-1,j,D} + \mathbb{P}_{i\to d} F_{i-1,j,I} \tag{5.19}$$

$$F_{i,j,I} = \frac{1}{4}(\mathbb{P}_{m\to i} F_{i,j-1,M} + \mathbb{P}_{s\to i} F_{i,j-1,S} + \mathbb{P}_{d\to i} F_{i,j-1,D} + \mathbb{P}_{i\to i} F_{i,j-1,I}) \tag{5.20}$$

where

$$\begin{cases} F_{i,j,S} = 0, & \text{if } \dot{x}_i = \dot{y}_j \\ F_{i,j,M} = 0, & \text{if } \dot{x}_i \neq \dot{y}_j \end{cases} \tag{5.21}$$

At the end, we get

$$\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1}, e_{t+1} | d_t, e_t, s_t, s_{t+1}) = F_{1,d_{t+1}-d_t+1,e_{t+1}} \tag{5.22}$$

which allows to evaluate $\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$.

In Chapter 3, we observed that the event probabilities also strongly depends on read k-mers, due to the way the nanopore sequencer works. This is why we now propose to modify *Dec2* in order to take into account both previous observed events and read k-mers.

5.3.3 CC decoder taking into account previous events and read k-mers (*Dec3*)

In the third decoder, in addition to previous events, we also take into account the statistical dependency between the current event and the underlying k-mer.

States of the decoder

The state of the decoder is now given by a quadruplet $\boldsymbol{\sigma}_t = (s_t, d_t, e_t, \boldsymbol{\eta}_t)$, 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 $t+1$. This results in a larger trellis which has 2^k times more nodes than for *Dec2*, see figure 5.11. Note that 2^k corresponds to the number of possible paths of length k in the trellis. At stage $t = 0$, this new state variable is initialized as $\boldsymbol{\eta_0} = \emptyset$. In addition, for $t < k$, we consider $\boldsymbol{\eta_t} = [\eta_1^{(t)}, \eta_2^{(t)}, \cdots, \eta_{t-1}^{(t)}, \eta_t^{(t)}]$.

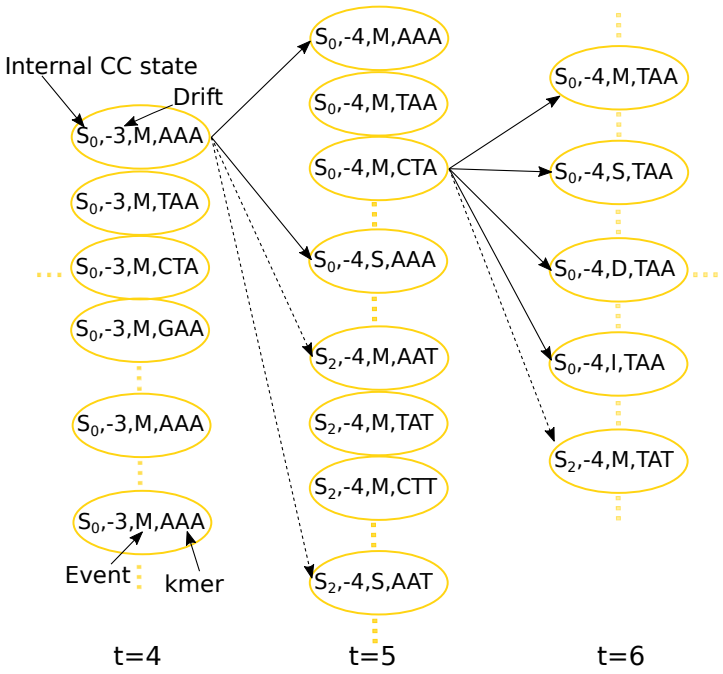


Figure 5.11: Partial trellis diagram for *Dec3*, for time instants $t = 4$, $t = 5$, $t = 6$, with $I_{\max} = 1$ and $d_{t+1} \in [\![d_t - 1, d_t + 1]\!]$.

A posteriori probability computation

With this new state definition, equation (5.4) still applies. To evaluate the three terms in $\mathbb{P}(\mathbf{y}, \boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1})$, we still use a forward recursion, a backward recursion, and a branch metric computation which we now describe.

Forward and Backward recursions

The forward recursion is still given by (5.5), with initialization given by (5.13). The backward recursion is still given by (5.7), but the initialization changes to take into account the k-mers as follows:

$$\beta_T(\boldsymbol{\sigma}_T) = \begin{cases} \mathbb{P}(M|\eta_t), & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, M) \\ \mathbb{P}(S|\eta_t), & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, S) \\ \mathbb{P}(D|\eta_t), & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, D) \\ \mathbb{P}(I|\eta_t), & \text{if } \boldsymbol{\sigma} = (0, N^{(v)} - N, I) \\ 0, & \text{otherwise.} \end{cases} \tag{5.23}$$

Branch metric computation

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

$$\gamma_t(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{t+1}) = \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1}, e_{t+1}, \boldsymbol{\eta}_{t+1}|d_t, e_t, \boldsymbol{\eta}_t, s_t, s_{t+1}) \tag{5.24}$$

$$= \mathbb{P}(w_t)\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_t.n_c+1}^{t.n_c+d_{t+1}.n_c}, d_{t+1}, e_{t+1}|d_t, e_t, \boldsymbol{\eta}_{t+1}, s_t, s_{t+1}) \tag{5.25}$$

because $\boldsymbol{\eta}_{t+1}$ is entirely determined by $\boldsymbol{\eta}_t$ and by the transition from s_t to s_{t+1}. The probability $\mathbb{P}(\mathbf{y}_{(t-1).n_c+d_{t+1}+1}^{t.n_c+d_{t+1}}, d_{t+1}, e_{t+1}|d_t, e_t, \boldsymbol{\eta}_{t+1}, s_t, s_{t+1})$ is evaluated from the same lattice in 3D shown in Figure 5.10 and used for *Dec2*. However, compared to *Dec2*, 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 $e_{t+1} \in \{M, S, D, I\}$ conditionally 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:

$$\begin{aligned} F_{i,j,M} = &\mathbb{P}(M|\eta_{t+1}, M)F_{i-1,j-1,M} + \mathbb{P}(M|\eta_{t+1}, S)F_{i-1,j-1,S} \\ &+ \mathbb{P}(M|\eta_{t+1}, D)F_{i-1,j-1,D} + \mathbb{P}(M|\eta_{t+1}, I)F_{i-1,j-1,I} \end{aligned} \tag{5.26}$$

$$\begin{aligned} F_{i,j,S} = &\frac{1}{3}(\mathbb{P}(S|\eta_{t+1}, M)F_{i-1,j-1,M} + \mathbb{P}(S|\eta_{t+1}, S)F_{i-1,j-1,S} \\ &+ \mathbb{P}(S|\eta_{t+1}, D)F_{i-1,j-1,D} + \mathbb{P}(S|\eta_{t+1}, I)F_{i-1,j-1,I}) \end{aligned} \tag{5.27}$$

$$\begin{aligned} F_{i,j,D} = &\mathbb{P}(D|\eta_{t+1}, M)F_{i-1,j,M} + \mathbb{P}(D|\eta_{t+1}, S)F_{i-1,j,S} \\ &+ \mathbb{P}(D|\eta_{t+1}, D)F_{i-1,j,D} + \mathbb{P}(D|\eta_{t+1}, I)F_{i-1,j,I} \end{aligned} \tag{5.28}$$

$$\begin{aligned} F_{i,j,I} = &\frac{1}{4}(\mathbb{P}(I|\eta_{t+1}, M)F_{i,j-1,M} + \mathbb{P}(I|\eta_{t+1}, S)F_{i,j-1,S} \\ &+ \mathbb{P}(I|\eta_{t+1}, D)F_{i,j-1,D} + \mathbb{P}(I|\eta_{t+1}, I)F_{i,j-1,I}) \end{aligned} \tag{5.29}$$

At the end, we get

$$\mathbb{P}(\mathbf{y}_{(t-1)n+d_t.n_c+1}^{tn+d_{t+1}.n_c}, d_t, e_t|d_{t+1}, e_{t+1}, \boldsymbol{\eta}_t, s_t, s_{t+1}) = F_{1,d_{t+1}-d_t+1,e_{t+1}} \tag{5.30}$$

as for Decoder 2.

5.3.4 Decoding with several sequences

The three previous decoders only take into account one output sequence $\mathbf{y}$, while the DNA data storage channel outputs V sequences $\mathbf{y}^{(v)}$. In [101], it was proposed to decode each sequence $\mathbf{y}^{(v)}$ independently and separately, and to aggregate the results *a posteriori* by relying on the following formula:

$$\mathbb{P}(w_t|\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \cdots, \mathbf{y}^{(V)}) = \frac{\prod_{v=1}^{V} \mathbb{P}(w_t, \mathbf{y}^{(v)})}{\mathbb{P}(w_t)^{V-1}} \tag{5.31}$$

Note that this aggregation could be realized at different levels of the decoding, but it was shown in [101] that it is more efficient to apply it on the *a posteriori* probabilities. In addition [101] also proposed another more efficient but also more complex technique to take into account multiple sequences directly inside the CC decoder. We did not consider this second technique, as in this work, we rather investigate whether taking into account previous events and read k-mers allow to improve the decoding.

5.4 Numerical results

In this section, we evaluate the performance of *Dec1*, *Dec2*, and *Dec3*. We first evaluate the performance of each decoder in terms of BER and FER. We then evaluate the performance of *Dec1* in the full DnarXiv pipeline.

5.5 FER and BER evaluation

We first evaluate the performance of *Dec1*, *Dec2*, and *Dec3*, by running Monte-Carlo simulations over our memory channel model. We consider both 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: it is about 10% on the model trained on *SetE*, and about 3% on the model trained on *SetG*, as explained in Chapter 3.

Each simulation run considers a randomly generated sequence $\mathbf{w}$, and each of its elements w_i takes value in the alphabet $\{A, C, G, T\}$ uniformly at random. Then $\mathbf{w}$ is encoded with a $(k_c = 1, n_c = 2, K_c = 3)$ CC, which uses the generator polynomial $\delta_{poly} = [\delta^2 + 1, \delta^2 + \delta + 1]$, and outputs a binary encoded sequence $\mathbf{x}$ of length $N = 54$. The sequence $\mathbf{x}$ is then transmitted through our memory channel model, which outputs V sequences $\mathbf{y^{(v)}}$ ($v \in [\![1, V]\!]$). Then the Convolutional decoder takes as input M sequences ($M \leq V$) randomly selected from the set $\mathbf{y^{(v)}}$. We consider 10000 simulation runs to evaluate the FER and BER of each decoder. Note that we consider short sequences due to the complexity that is introduced by *Dec3*.

Figures 5.12 and 5.13 show the FER and BER, respectively, of the three decoders over the memory channel model trained onto *SetE*. We observe on these figures that *Dec3* has the best performance, which is expected since it fully takes into account the channel model. The performance gain is even more significant when the number of sequences M considered by *Dec3* increases. We also observe that the performance of *Dec2* is the worst, most probably because this decoder does not take into account all aspects of our memory channel model. We also notice that increasing M does not improve much the performance of *Dec1*. This is because *Dec1* assumes an i.i.d. channel. Thus, aggregating the results of separate decoding will less impact than when considering the memory in the channel.

Figures 5.14 and 5.15 show the FER and BER, respectively, of the three decoders over the memory channel model trained onto *SetG*. As previously, we observe that *Dec3* has

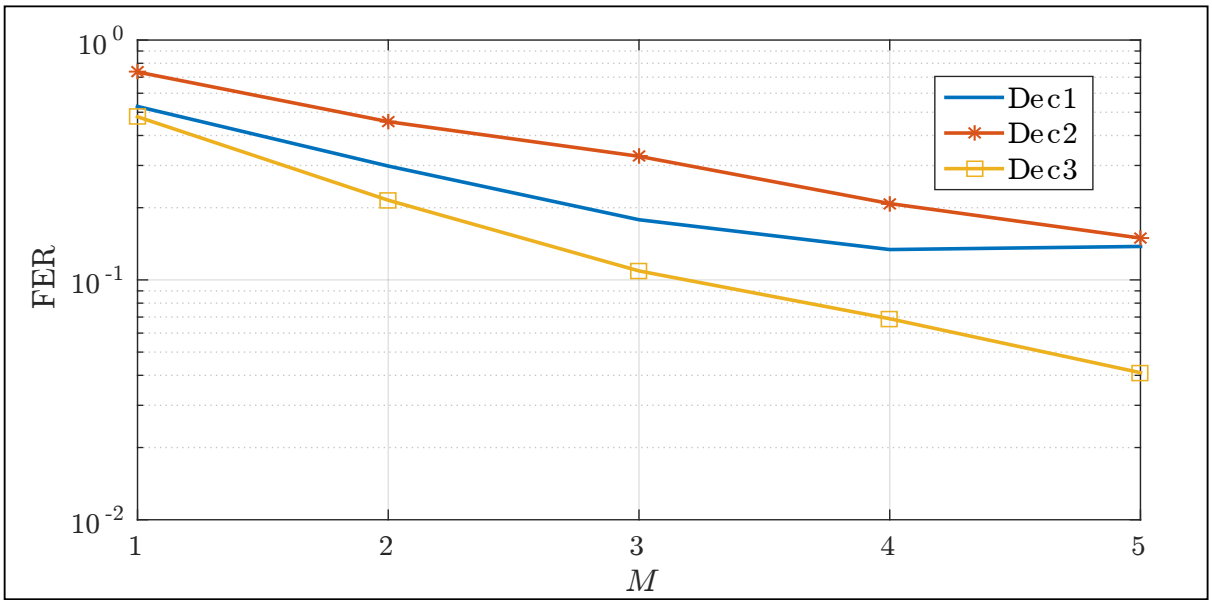


Figure 5.12: FER with respect to the number of sequences M, over the memory channel model trained onto *SetE*.

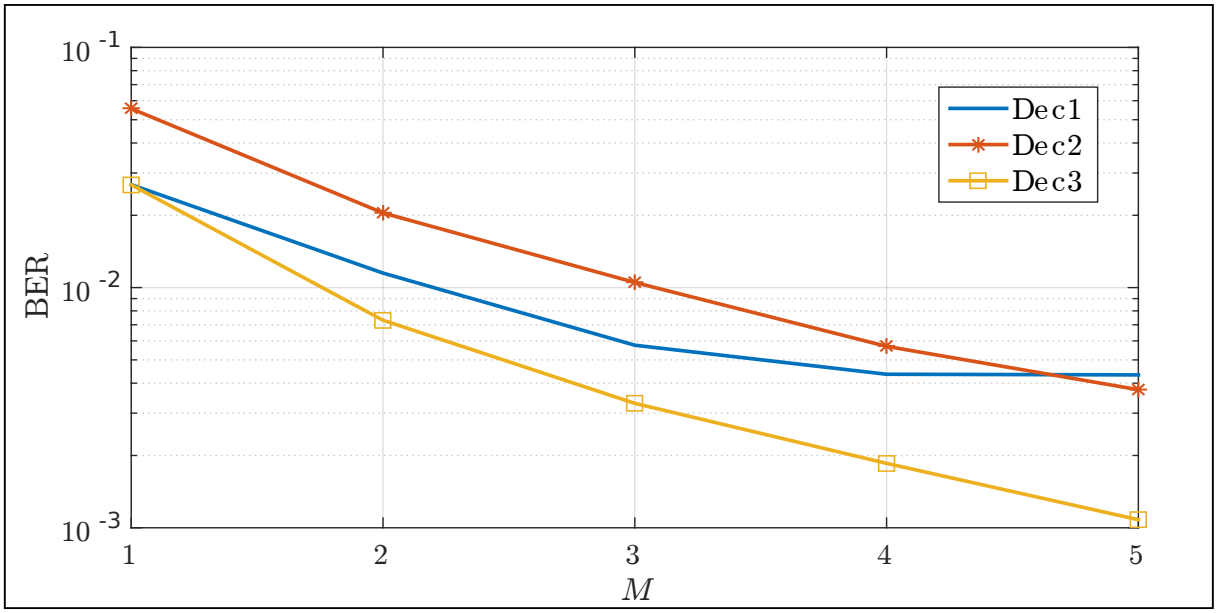


Figure 5.13: BER with respect to the number of sequences M, over the channel model trained onto *SetE*.

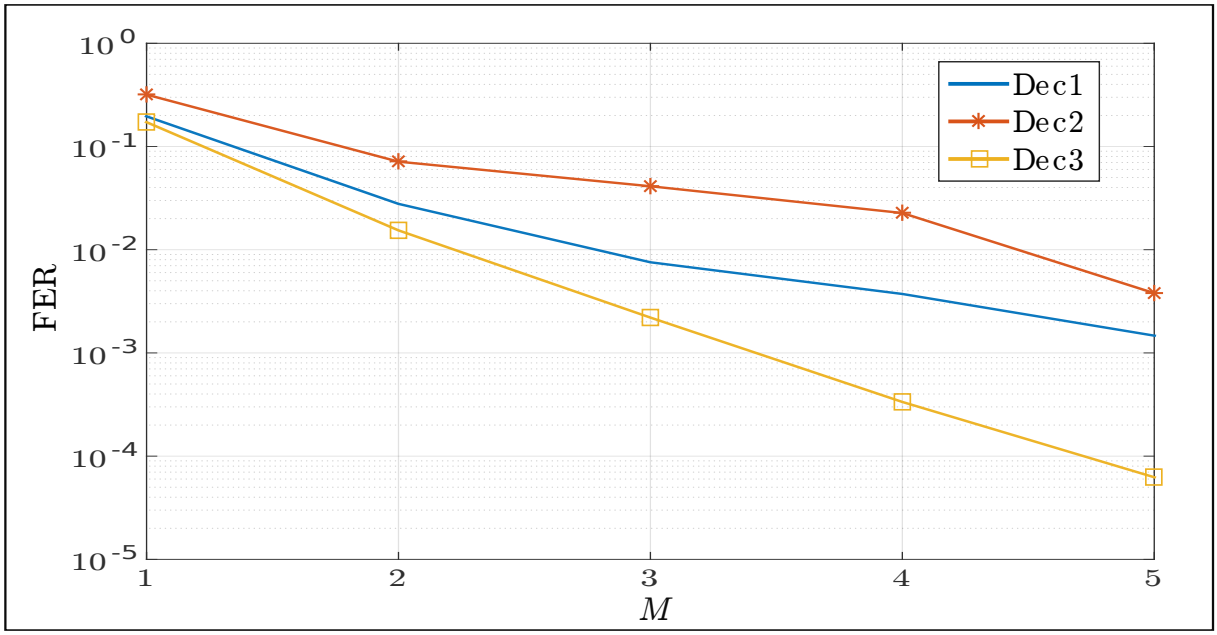


Figure 5.14: FER with respect to the number of sequences M, over the channel model trained onto *SetG*.

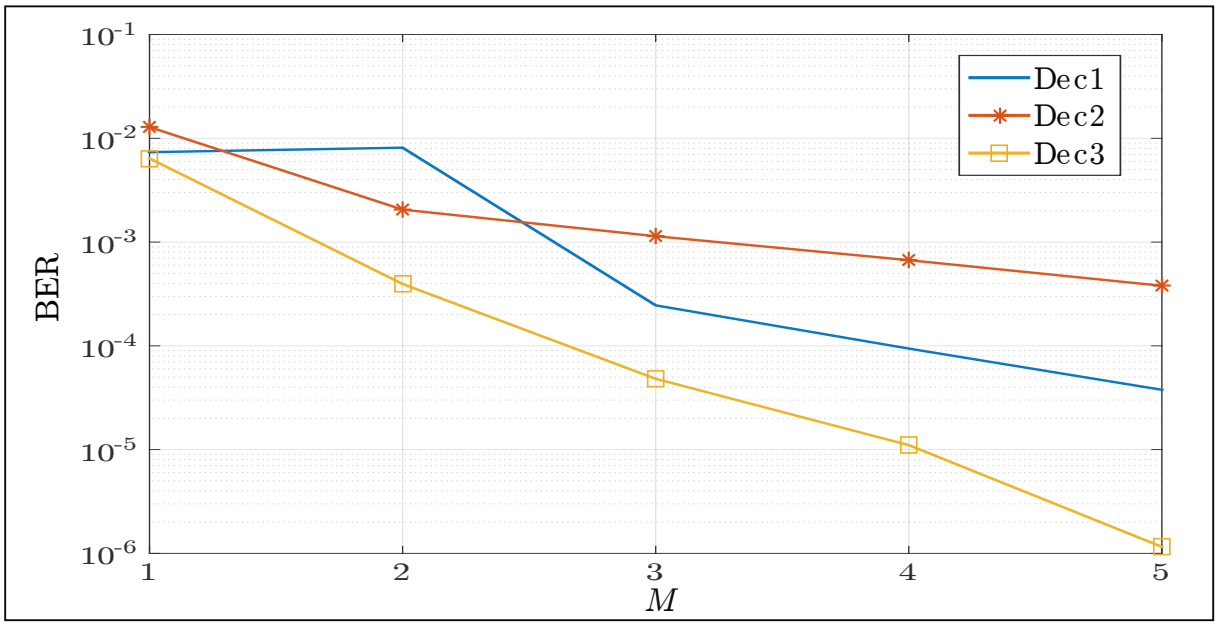


Figure 5.15: BER with respect to the number of sequences M, over the channel model trained onto *SetG*.

the best performance, especially when M increases. Furthermore, performance of *Dec2* is still the worst, most probably for the same reasons mentioned before. Note that on these figures the obtained BER and FER are lower than on Figures 5.14 and 5.15, because overall error probability on *SetG* is lower than on *SetE*.

5.6 The convolutional decoder solution in the full DnarXiv pipeline

Dec1 has also been tested by Olivier Boullé using the DnarXiv pipeline described in Chapter 4. In this new set of simulations, the consensus and NB-LDPC decoder highlighted in green on Figure 4.13 have been replaced by *Dec1*, which takes into account only drifts. For now, only *Dec1* was tested due to the high complexity introduced by *Dec3*. The purpose of these simulations is the same as in Chapter 4, *i.e.*, compare the CCSA consensus algorithm, which does not use channel coding, to *Dec1* which relies on the CC. To perform this comparison, the following protocol was used:

1. Convert a document into 200 fragments.
2. Encode each fragment with the $(1, 2, 3)$ CC. The encoded fragments are of length $N = 100$ in GF(4) in a first setup, and of length $N = 200$ in GF(4) in a second setup.
3. The obtained fragments are used as inputs for our memory channel model trained on *SetE*, which outputs several edited replicas of each fragment.
4. The obtained sequences are then separated into 200 clusters, where we expect that each cluster contains sequences related to one fragment of the document.
5. In order to reconstruct the fragments, on each cluster, M sequences are selected at random and used either by *Dec1* or by the CCSA consensus algorithm.
6. The average precision of all reconstructed fragments is then computed. The precision means the percentage of correctly aligned and equal bases between the reconstructed fragment and the original one.

Figures 5.16 and 5.17 show the precision of the two solutions when reconstructing fragments of length $N = 100$ and $N = 200$, respectively. On both figures, we observe that *Dec1* significantly outperforms the consensus algorithm. These results are not surprising, since the consensus algorithm is based on a majority vote. As a results, it needs much

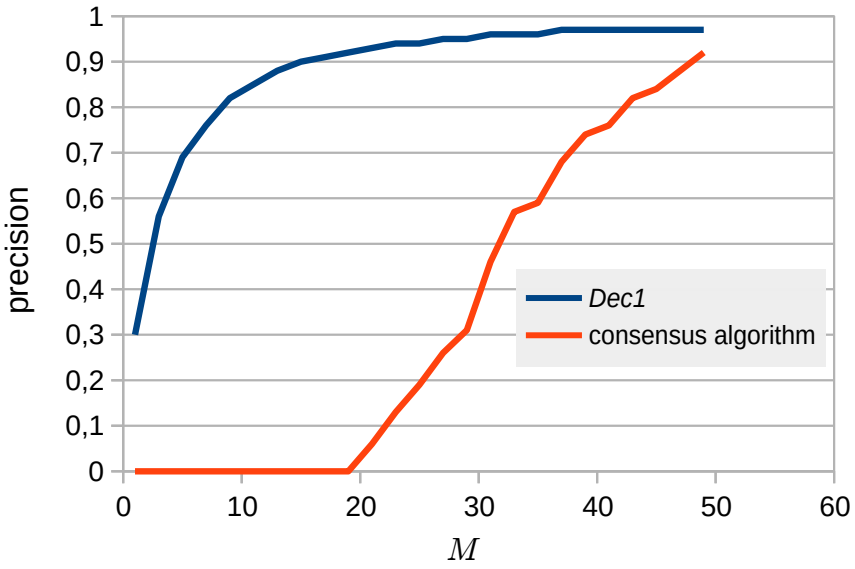


Figure 5.16: *Dec1* and CCSA consensus reconstruction precision with respect to the number of sequences M, over the channel model trained onto *SetE*. Reconstructed fragments are of length $N = 100$ in GF(4).
Credit: Figure provided by Olivier Boullé.

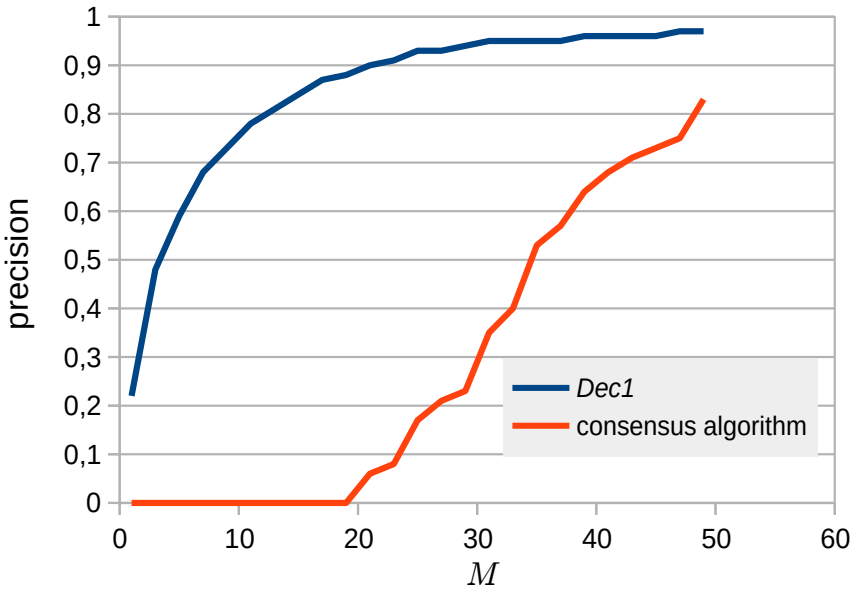


Figure 5.17: *Dec1* and CCSA consensus reconstruction precision with respect to the number of sequences M, over the channel model trained onto *SetE*. Reconstructed fragments are of length $N = 200$ in GF(4).
Credit: Figure provided by Olivier Boullé.

more sequences during the reconstruction. Unlike in Chapter 4, where the NB-LDPC decoding relied on the consensus algorithm for sequence synchronization, *Dec1* tackles both synchronization and substitution errors, and can perform efficient decoding even with a small number M of sequences. We also observe that the precision of the consensus algorithm on Figure 5.17 where $N = 200$, is lower than its precision on Figure 5.16 where $N = 100$. Hence, for larger values of N the consensus algorithm performance decreases. This is because the consensus algorithm needs more sequences to perform reliable majority votes. On the opposite, the performance of *Dec1* remains the same since it does not depend on the sequence length N, but on the amount of errors on the sequences.

These results clearly show the contribution of channel coding to reduce the number M of sequences to process, and to increase the reconstruction precision. Furthermore, based on the previous BER and FER evaluation, we have good reasons to think that in practice, *Dec3* could provide a better precision of reconstruction than *Dec1*. In addition, unlike the consensus algorithm, performance of *Dec3* could increase when N becomes larger, since generally speaking, the performance of channel codes is better when N increases.

5.7 Conclusion

In this chapter, we introduced a DNA data storage coding scheme based on CCs. We then proposed two modified versions of the CC decoder initially proposed in [101]. We saw through the numerical results that our modified decoder, which takes into account both the memory of the observed events and the read k-mers, offers better performance than the decoder of [101]. We also saw from numerical simulations that inside the DnarXiv pipeline, the decoder from [101] outperforms the consensus algorithm. There is a need to reduce the complexity of *Dec3* before integrating it into the DnarXiv pipeline, which may further improve the decoding performance.

Chapter 6

Deduplication algorithms and models for efficient data storage

In this chapter, we investigate data deduplication algorithms for efficient data storage. We present data duplication over an edit channel which is very close to the DNA storage channel. We describe the Pivot-Based Deduplication Algorithm PBDA initially proposed in [125], which allows to reduce the deduplication process complexity. We then propose a modified version of the PBDA, called PBDA-SW (Sliding-Window), which improves the PBDA deduplication ratios when considering high error rates. Finally, we compare the performance of our PBDA-SW to the PBDA and to other existing solutions.

The work presented in this chapter was carried out during the first year of this thesis, i.e., it was done before the work on the previous chapters. This chapter is also separate from the previous ones in the sense that it addresses data deduplication, while previous chapters were dedicated to channel coding. This is because, the DnarXiv project was accepted at the end of the first year of this thesis. Therefore, the work on data deduplication was put aside in order to focus on the priorities of the DnarXiv project, namely channel modeling and coding for DNA data storage. Therefore, the work presented in this chapter targets data storage systems in general. At the end of the chapter, we discuss how this work could be used in the context of DNA data storage.

6.1 Data Deduplication

Data deduplication [28, 126] aims to remove redundant copies of the same data on a Data Storage System (DSS), so as to reduce the space used on the DSS. By doing so, it can also reduce the network bandwidth. In this work, a DSS means one or several drives accross one or several machines. Hence, data deduplication is a key feature for many cloud and enterprise servers [127].

Figure 6.1 illustrates how deduplication reduces the amount of used space on a DSS.

In this figure, each letter refers to a particular block of data, while same letters refer to the same data. We see that the deduplication process removes the duplicated data by saving only a unique copie of each data, while pointers (&) replace the redundant data and point to the location of the unique copie of each data.

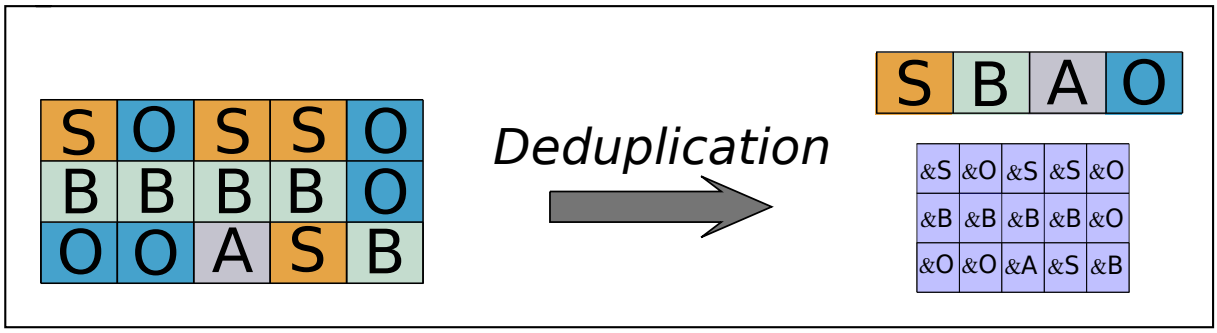


Figure 6.1: This figure illustrates the data deduplication process. On the left side we observe the initial blocks of data, which goes through the deduplication process. The right part shows how a unique copy of each initial blocks is saved, while pointers, which are smaller than the original blocks, are used to refer to the saved copies.

6.1.1 Deduplication VS compression

Deduplication can be considered as a compression operating on a large-scale. However, deduplication is different [126] from classical compression [128, 129] on several aspects.

— Compression operates on one file at a time, by removing intra-file redundancy. Oppositely, deduplication operates across an entire DSS to save a unique copy of each data.

— Compression implies that a decompression step is necessary to retrieve the original data. In contrast, deduplication uses pointers which point to unique copies of data. Therefore, there is no need for a reconstruction step.

Classical compression is not well adapted for large-scale DSS, in the sense that it would compress files individually, while deduplication operates over large sets of files.

6.1.2 Deduplication techniques

When applying deduplication, we consider that the data is split into a large number of pieces called "chunks", where either a chunk corresponds to one file [127, 130], or each file is split into several chunks [127, 130]. Prior works on data deduplication are mostly

based on hash algorithms [28, 131, 127]. In these works, a hash key is generated for each chunk contained in the DSS. Then, if it is a new chunk, its hash key is computed and then compared to the lookup table of existing hashes. If the hash already exists in the table, then the chunk is replaced by a pointer to the existing copie. Otherwise, a new entry is added to the lookup table.

The deduplication can be executed as an inline [131, 127] or as a post-processing [131, 127] process, as we now describe.

— **Inline deduplication** processes the data before it is written onto the DSS. Inline deduplication first sends the hash of each chunk to the DSS. A given chunk is transmitted to the DSS only if its hash does not exist in the lookup table. In addition to reducing the amount of stored data, inline deduplication also reduces the bandwidth because it transmits only the chunks for which the hash is not in the lookup table. However, the main drawback of this technique is the increased writting latency [131, 127], *i.e*, several comparisons are performed before writing the data to the DSS.

— **Post-processing deduplication** first writes the data to the DSS, and then calculates the corresponding hashes. In this case, there is no overhead in the writing latency. However, in order to write the original data, this process requires more storage space [131]. Furthermore, this process needs additional resources to continuously scan the DSS and look for deduplicable data.

In this work we consider inline deduplication so as to also reduce bandwidth.

6.1.3 Deduplication Granularity

Data chunking can be done at two different levels [130], and it has a direct influence on the deduplication performance.

1. **File-level**: each chunk corresponds to a whole file. In this case, only a small number of hashes (one per file) are generated. Therefore, the deduplication process has a low complexity. However, if two large files only differ by one bit, no data deduplication is possible.
2. **Block-level**: each file is divided into several blocks. The size of each block is either fixed or variable.
   — *Fixed-size*: each file is splitted into blocks of the same size and each block corresponds to a chunk. This makes it possible to deduplicate data whenever

the compared files have some chunks in common. The complexity of this approach is higher than the file-level approach, but it offers a better deduplication performance.

— *Variable-size*: each file is splitted into blocks of variable sizes, and each block corresponds to a chunk. Although it has the highest complexity, this approach offers the best deduplication performance [127].

In this chapter, we consider the Pivot-Based Deduplication Algorithm (PBDA) introduced in [125]. This algorithm aims to reduce the deduplication complexity compared to existing approaches, while maintaining a reasonable deduplication performance. This algorithm considers an inline approach, and a variable-size block-level deduplication.

6.2 Edit channel model

In this chapter, we consider an initial file $\mathbf{x}$ stored on a DSS, and a file $\mathbf{y}$ which is an edited version of $\mathbf{x}$. Here, we consider that edits are either insertions or deletions. We then try to achieve the best possible deduplication of $\mathbf{y}$. The file $\mathbf{x}$ corresponds to a non-binary random sequence of length N, where each element x_i takes values in the alphabet $\mathbf{\Omega} = \{0, 1, \ldots, q-1\}$ with a uniform probability distribution. We consider the same i.i.d. channel model as in [125, 132]. This model is shown in Figure 6.2. It sequentially introduces insertions with a probability $\mathbb{P}_i$, and deletions with a probability $\mathbb{P}_d$. Note that a substitution can occur when a deletion is immediately followed by an insertion. Let us assume a sequence $\mathbf{e}$ of length R ($R \geq N$), which corresponds to the edit pattern, such that $\mathbf{e} = (e_1, e_2, \ldots, e_R)$, and $e_r \in \{-1, 0, 1\}$ such that:

— If $e_r = -1$, then a deletion occurs, and hence, $\mathbf{y}$ is not updated. This event has probability $\mathbb{P}_d$.

— If $e_r = 0$, then there is no edit, and hence, $\mathbf{y} \leftarrow x_j$. This event has probability $1 - \mathbb{P}_i - \mathbb{P}_d$.

— If $e_r = 1$, a new symbol a taken at random from $\mathbf{\Omega}$ is inserted, and hence $\mathbf{y} \leftarrow a$. This event has probability $\mathbb{P}_i$. The process remain on x_j

Therefore, the sequence $\mathbf{y}$ is obtained from the sequence $\mathbf{x}$, according to the edit pattern $\mathbf{e}$.

In what follows, we use d_n ($n \in [\![1, N]\!]$) to denote the drift value, which corresponds to the number of insertions minus the number of deletions observed before transmitting

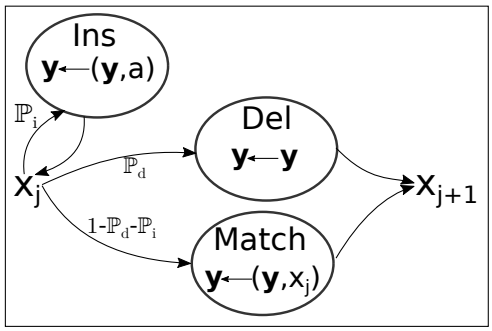


Figure 6.2: Edit channel model.

the symbol x_{n+1}. Therefore, d_n is defined as $d_n = \sum_{i=1}^{n} e_i$. Furthermore, we assume that the probability of an edit $\mathbb{P} = \mathbb{P}_d + \mathbb{P}_i$ is relatively small, and that $\mathbb{P}_d = \mathbb{P}_i$.

This model allows to consider data deduplication from an information theory point of view. Only a few works [125, 133, 134] have considered this approach.

6.3 Data representation

We now introduce some definitions before describing the PBDA in [125]. The sequence $\mathbf{x}$ can be represented as

$$\mathbf{x} = \mathbf{s}^{(1)}, \mathbf{p}^{(1)}, \mathbf{s}^{(2)}, \mathbf{p}^{(2)}, \ldots, \mathbf{s}^{(M-1)}, \mathbf{p}^{(M-1)}, \mathbf{s}^{(M)}, \mathbf{p}^{(M)}$$

where $\mathbf{s}^{(i)}$ is a *segment* which contains L_s symbols, and $\mathbf{p}^{(i)}$ is a *pivot* which contains L_p symbols, and $i \in [\![1, M]\!]$. M denote the number of blocks in $\mathbf{x}$, and the block i is given by the concatenation of the substrings $\mathbf{s}^{(i)}, \mathbf{p}^{(i)}$. We assume that L_p is relatively small compared to L_s ($L_s \ll L_p$). Moreover, L_p has to be small enough so that the probability of edit in a pivot $\mathbf{p}^{(i)}$ is relatively small, and large enough so that $\mathbf{p}^{(i)}$ contains enough symbols to perform reliable comparisons. We also assume that the file length N is divisible by the block length $L_B = L_s + L_p$. We can remark that the number of blocks on a file of length N is $M = \frac{N}{L_B}$.

We use $\mathbf{c}$ to denote a chunk, which corresponds to the concatenation of G consecutive segments and pivots as

$$\mathbf{c} = \mathbf{s}^{(i)}, \mathbf{p}^{(i)}, \ldots, \mathbf{s}^{(i+G)}, \mathbf{p}^{(i+G)}, \quad \text{and } i + G \leq M$$

.

Finally, $N = \sum_{t=1}^{T} l_t$, where l_t is the length of the chunk $\mathbf{c}^{(t)}$, and T corresponds to the number of chunks $\mathbf{c}$. Since we consider a variable-size block-level deduplication, l_t can differ from one chunk to another.

6.4 Deduplication algorithms based on pivots

We now describe the PBDA initially introduced in [125], which considers an inline approach with a variable-size block-level deduplication.

6.4.1 Operators

The symbol matcher operator $\odot$ that is used to compare pivots $\mathbf{p}^{(i,\mathbf{x})}$ and $\mathbf{p}^{(i,\mathbf{y})}$, of $\mathbf{x}$ and $\mathbf{y}$, respectively, is defined as

$$\mathbf{p}^{(i,\mathbf{x})} \odot \mathbf{p}^{(i,\mathbf{y})} = (p_1^{(i,\mathbf{x})} \odot p_1^{(i,\mathbf{y})}, p_2^{(i,\mathbf{x})} \odot p_2^{(i,\mathbf{y})}, \ldots, p_{L_p}^{(i,\mathbf{x})} \odot p_{L_p}^{(i,\mathbf{y})}), i \in [\![1, M]\!]$$

with

$$p_j^{(i,\mathbf{x})} \odot p_j^{(i,\mathbf{y})} = \begin{cases} 1, \text{ if } p_j^{(i,\mathbf{x})} = p_j^{(i,\mathbf{y})} \\ 0, \text{ if } p_j^{(i,\mathbf{x})} \neq p_j^{(i,\mathbf{y})} \end{cases} , j \in [\![1, L_p]\!]$$

Therefore, $\mathbf{p}^{(i,\mathbf{x})}$ perfectly matches $\mathbf{p}^{(i,\mathbf{y})}$ only if $\forall j \in [\![1, L_p]\!],\ p_j^{(i,\mathbf{x})} \odot p_j^{(i,\mathbf{y})} = 1$. In this case, the drift d_n is equal to 0 for $n = iL_B$. Note that $d_n = 0$ can either mean that there was no edit before the pivot $\mathbf{p}^{(i,\mathbf{y})}$, or that the number of deletions is equal to the number of insertions.

We use $\mathcal{S}(\mathbf{p}, u)$ to denote a shift operator, which shifts the pivot $\mathbf{p}$ by $|u|$ positions to the left or to the right, when $u < 0$ or $u > 0$, respectively. Hence,

— if $u > 0$, then $\mathcal{S}(\mathbf{p}, u) = (\underbrace{\Delta, \ldots, \Delta}_{u}, p_1, p_2, \ldots, p_{L_p - u})$.

— if $u < 0$, then $\mathcal{S}(\mathbf{p}, u) = (p_{|u|+1}, p_{|u|+2}, \ldots, p_{L_p}, \underbrace{\Delta, \ldots, \Delta}_{u})$.

where Δ corresponds to a null value, and $\forall j \in [\![1, L_p]\!],\ p_j \odot \Delta = 0$.

6.4.2 Principle of the PBDA

To determine if the block $(\mathbf{s}^{(i)}, \mathbf{p}^{(i)})$ can be deduplicated, the PBDA only compares the pivot part $\mathbf{p}^{(i)}$. Therefore, it decreases the complexity of the deduplication process.

The PBDA works as follow:

1. **Initialization**: the PBDA partitions the files $\mathbf{x}$ and $\mathbf{y}$, respectively, into M segments $\mathbf{s}^{(i,\mathbf{x})}$ and $\mathbf{s}^{(i,\mathbf{y})}$ of length L_s, and into M pivots $\mathbf{p}^{(i,\mathbf{x})}$ and $\mathbf{p}^{(i,\mathbf{y})}$ of length L_p, with $i \in [\![1, M]\!]$ and $L_p \ll L_s$. D_x and D_y are parameters which corresponds to the positions of the next pivots to process in $\mathbf{x}$ and $\mathbf{y}$, respectively. They are both initialized to the position of the first pivot, that is $D_x = D_y = L_s + 1$.
2. **Chunk detection**: a Consecutive Pivots Module (CPM) is used to determine the size of the chunk $\mathbf{c}$ that can be deduplicated. The CPM, which will be described into more details in the next section, identifies all the G consecutives blocks $(\mathbf{s}^{(g)}, \mathbf{p}^{(g)})$ that are free of edits, with $g \in [\![1, G]\!]$. The CPM also determines the drift d_n (with $n = (G+1)L_B$) on the pivot $\mathbf{p}^{(G+1)}$, *i.e.*, the pivot that comes after the deduplicated chunk.
3. **Update**: the parameters D_x and D_y are updated as
   — $D_x = D_x + (G + 1) \times (L_s + L_p)$
   — $D_y = D_x + d_n$

The algorithm repeats steps two and three until $i = M$, or until an edit on a pivot is encountered. We explain later in this section why the algorithm cannot handle edits on pivots.

6.4.3 Description of the CPM

The CPM is the core component of the PBDA. It allows to identify deduplicable chunks $\mathbf{c}$, and to compute the drift d_n, which is needed to update D_y if there is an edit. We now explain into more details how the CPM works.

— The size of the deduplicable chunk is initialized as $G = 0$.
— In the case where $\mathbf{p}^{(i,\mathbf{x})} = \mathbf{p}^{(i,\mathbf{y})}$, the size of the deduplicable chunk increases: $G = G + 1$, and the positions of the next pivots are updated: $D_x = D_x + (L_s + L_p)$, and $D_y = D_x$.
— As soon as $\mathbf{p}^{(i,\mathbf{x})} \neq \mathbf{p}^{(i,\mathbf{y})}$, the drift d_n (with $n = iL_B$) is computed so as to correct the position D_y of the next pivot $\mathbf{p}^{(i+1,\mathbf{y})}$.
— The drift d_n is then computed to detect insertions or deletions. According to the form of the resulting vector, the drift is set to

$$d_n = \begin{cases} u, \text{ if } \mathbf{p}^{(i,\mathbf{x})} \odot \mathcal{S}(\mathbf{p}^{(i,\mathbf{y})}, -u) = (\underbrace{1,\ldots,1}_{L_P-u}, \underbrace{0,\ldots,0}_{u}) \\ -u, \text{ if } \mathbf{p}^{(i,\mathbf{x})} \odot \mathcal{S}(\mathbf{p}^{(i,\mathbf{y})}, u) = (\underbrace{0,\ldots,0}_{u}, \underbrace{1,\ldots,1}_{L_P-u}) \end{cases}, \forall u \in \left[\!\left[1, \frac{L_p}{2}\right]\!\right]$$

The CPM then outputs G and d_n. Note that to perform reliable comparisons, the maximum value of the shift u is fixed to $\frac{L_p}{2}$ as in [125].

The PBDA allows to decrease the number of comparisons needed for chunks deduplication. However, when the channel error probability increases, the deduplication performance decreases. This is because edits appear in the pivots, and hence, the algorithm cannot deduplicate the remaining file. In this case, the PBDA stops its execution because it cannot compute the drift since the resulting vector when comparing $\mathbf{p}^{(i,\mathbf{x})}$ and $\mathbf{p}^{(i,\mathbf{y})}$ has neither the form $(\underbrace{1,\ldots,1}_{L_P-u}, \underbrace{0,\ldots,0}_{u})$ nor $(\underbrace{0,\ldots,0}_{u}, \underbrace{1,\ldots,1}_{L_P-u})$.

6.4.4 PBDA performance

In this Section, we evaluate the deduplication ratios of the PBDA under various edit probabilities $\mathbb{P}$. The deduplication ratio corresponds to the amount of deduplicated data divided by the amount of initial data. Each simulation run considers a couple of files $\mathbf{x}$, $\mathbf{y}$. The file $\mathbf{x}$ of length $n = 120000$ is generated randomly, and each of its elements x_i takes value in the alphabet $\mathbf{\Omega} = \{0, \ldots, q-1\}$ uniformly at random, with $q = 64$. The edit channel has a certain probability of edit $\mathbb{P} = \mathbb{P}_i + \mathbb{P}_d$, with $\mathbb{P}_i = \mathbb{P}_d$. We perform 1000 simulation runs for each considered value of $\mathbb{P}$.

Figure 6.3 shows the deduplication ratios with respect to the edit probability $\mathbb{P}$. We observe that the amount of deduplicated data drastically decreases when $\mathbb{P} \geq 10^{-4}$. This is because in this range, errors occur more often in the pivots, and then stop the deduplication process. Therefore, the PBDA should be improved in order to tolerate higher error rates. This is why we now propose a modified version of the PBDA, called PBDA-SW (Pivots-Based Deduplication Algorithm with a Sliding Window) [135], which can tolerate edits in the pivots, and thus, increase the deduplication ratios.

6.4.5 Principle of the PBDA-SW

We now introduce the PBDA-SW [135], which is a modified version of the PBDA in [125]. The PBDA-SW allows to continue the deduplication proccess even if there is an edit in a pivot.

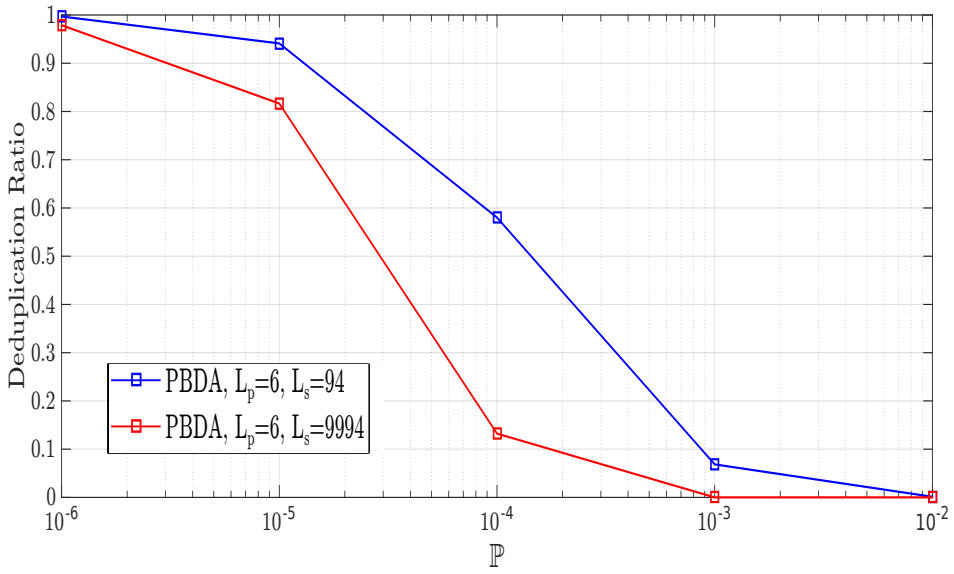


Figure 6.3: Data deduplication ratio as a function of $\mathbb{P}$. $L_p = 6$ and $L_s = 94$ (*i.e.*, $M = 1200$)

In the channel model described in Figure 6.2, $\mathbb{P}_i = \mathbb{P}_d$. Therefore, the expected value of the drift d_n over the file is zero, *i.e.*, $\mathbb{E}(d) = 0$. As a result, each component x_i of $\mathbf{x}$ produces component $y_{i'}$ of $\mathbf{y}$, with $i' = i$, or $i' \neq i$ but i and i' are close to each other. Therefore, whenever there is an edit in a pivot, we propose to use a sliding-window of length $L_w = L_p$, which slides the pivot to the right by $w \in [\![1, n_s]\!]$ positions, n_s being the maximum number of slides. After each slide, there is an attempt to compute d_n. It is possible to compute d_n with the CPM if there is no edits in the window. The sliding-window stops its execution if d_n is computed, or if $w = n_s$.

For instance, let us consider a set of symbols Ω with $q = 16$ and $L_p = 5$. Let the beginning of file $\mathbf{x}$ be:

$$\mathbf{x} = \underbrace{1, ..., A, B, 3}_{\mathbf{s}^{(1,\mathbf{x})}}, \underbrace{4, C, D, 5, E}_{\mathbf{p}^{(1,\mathbf{x})}}, \underbrace{6, F, ..., 8, 9}_{\mathbf{s}^{(2,\mathbf{x})}} \underbrace{B, 2, C, 3, 1}_{\mathbf{p}^{(2,\mathbf{x})}} C, B, A, ...$$

and the corresponding edit pattern:

$$\mathbf{e} = \underbrace{0, \ldots, 0, -1, -1}_{\mathbf{s}^{(1)}} \underbrace{0, \ldots, 0}_{\mathbf{p}^{(1)},\mathbf{s}^{(2)}} \underbrace{0, 0, 0, -1, 0}_{\mathbf{p}^{(2)}}$$

The beginning of file $\mathbf{y}$ is then:

$$\mathbf{y} = 1, ..., A, 4, C, \underbrace{D, 5, E, 6, F}_{\mathbf{p}^{(1,\mathbf{y})}}, \underbrace{..., 8, 9, B, 2}_{\mathbf{s}^{(2,\mathbf{y})}} \underbrace{C, 1, C, B, A}_{\mathbf{p}^{(2,\mathbf{y})}}, ...$$

The CPM performs the pivot matching with

$$\mathbf{p}^{(1,x)} = (4, C, D, 5, E) \text{ and } \mathbf{p}^{(1,\mathbf{y})} = (D, 5, E, 6, F)$$

Since $\mathbf{p}^{(1,\mathbf{x})} \odot \mathbf{p}^{(1,\mathbf{y})} \neq (1, ..., 1)$, the CPM outputs $G = 0$ as if there was no deduplicable chunk. Then, the computation of d_n (with $n = 1L_B$) starts. There is no error in the pivot, and since $\mathbf{p}^{(1,\mathbf{x})} \odot \mathcal{S}(\mathbf{p}^{(1,\mathbf{y})}, 2) = (0, 0, 1, 1, 1)$, then $u = 2$, and $d_n = -2$. Therefore, the next pivot $\mathbf{p}^{(2,\mathbf{y})}$ is shifted by $d_n = -2$ positions to the left.

The second execution of the CPM performs the pivot matching with

$$\mathbf{p}^{(2,\mathbf{x})} = (B, 2, C, 3, 1) \text{ and } \mathbf{p}^{(2,\mathbf{y})} = (B, 2, C, 1, C)$$

Since $\mathbf{p}^{(2,\mathbf{x})} \odot \mathbf{p}^{(2,\mathbf{y})} \neq (1, ..., 1)$, then $G = 0$. Since the result of

$$\mathbf{p}^{(2,\mathbf{x})} \odot \mathcal{S}(\mathbf{p}^{(2,\mathbf{y})}, u), \forall u \in \left[\!\left[1, \frac{L_p}{2}\right]\!\right]$$

is neither of the form $(\underbrace{1, \ldots, 1}_{L_P - u}, \underbrace{0, \ldots, 0}_{u})$ nor $(\underbrace{0, \ldots, 0}_{u}, \underbrace{1, \ldots, 1}_{L_P - u})$, there is an edit (deletion of the symbol 3) in the pivot $\mathbf{p}^{(2,\mathbf{y})}$. Therefore, the drift d_n (with $n = 2L_B$) cannot be computed, and the sliding window technique is used:

— for $w = 1$, $\mathbf{p}^{(2,\mathbf{x})} = (2, C, 3, 1, C)$ and $\mathbf{p}^{(2,\mathbf{y})} = (2, C, 1, C, B)$. The position of the deleted symbol is still in $\mathbf{p}^{(2,\mathbf{y})}$. Thus, d_n cannot be computed, and the initial pivot is shifted by $w \leftarrow w + 1$ positions.

— for $w = 2$, $\mathbf{p}^{(2,\mathbf{x})} = (C, 3, 1, C, B)$ and $\mathbf{p}^{(2,\mathbf{y})} = (C, 1, C, B, A)$. The position of the deleted symbol is still in $\mathbf{p}^{(2,\mathbf{y})}$. Thus, d_2 cannot be computed, and the initial pivot is shifted by $w \leftarrow w + 1$ positions.

— for $w = 3$, $\mathbf{p}^{(2,\mathbf{x})} = (3, 1, C, B, A)$ and $\mathbf{p}^{(2,\mathbf{y})} = (1, C, B, A, 4)$. The position of the deleted symbol is not in $\mathbf{p}^{(2,\mathbf{y})}$, and $\mathbf{p}^{(2,\mathbf{x})} \odot \mathcal{S}(\mathbf{p}^{(2,\mathbf{y})}, u = 1) = (0, 1, 1, 1, 1)$. Therefore, the drift $d_{2L_B} = -1$ is computed.

Then the position of the next pivot D_x and D_y are updated the same way than in the

CPM in [125], such that

$$D_x = D_x + (G+1) \times (L_s + L_p) \text{ and } D_y = D_x + d_{G+1}$$

The deduplication process can continue.

Therefore, compared to the PBDA in [125], the PBDA-SW allows to continue the deduplication process even if there are edits in a pivot.

6.5 Performance of the PBDA-SW

In this section, we compare the performance of the PBDA in [125] to some Brute Force Methods (BFMs) for deduplication [125] and to our PBDA-SW through a theoretical analysis and numerical simulations.

6.5.1 Theoretical analysis

We introduce a theoretical analysis that consists of evaluating the cost $\mathcal{C}$ defined as the average number of comparisons of each algorithm. We consider an initial file $\mathbf{x}$ of length N, and a file $\mathbf{y}$ which is an edited version of $\mathbf{x}$. We then provide expressions of the average number of comparisons necessary to deduplicate the file $\mathbf{y}$ given an edit probability $\mathbb{P}$.

File-level BFM complexity

The file-level BFM [125] compares all symbols of $\mathbf{x}$ and $\mathbf{y}$. The comparison stops if it encounters unequal symbols or if it reaches the end of the files. The cost $\mathcal{C}_F$ is then expressed as

$$\mathcal{C}_F = \sum_{i=1}^{N-1} \Big[(1-\mathbb{P})^{i-1}(i\mathbb{P})\Big] + (1-\mathbb{P})^{N-1}(N\mathbb{P}) + N(1-\mathbb{P})^N \tag{6.1}$$

In this expression, the first term corresponds to the average number of comparisons given that the $(i-1)^{th}$ symbol is correct (has no edit), while the i^{th} symbol is edited. The second and third terms are special cases which correspond to the case where the $(N-1)$ symbols of the file are not edited while the N^{th} one is edited, and to the case where none of the N symbols was edited, respectively.

Block-level BFM complexity

The block-level BFM [125] first divides $\mathbf{x}$ and $\mathbf{y}$ into M blocks, and then compares the symbols of each block. The comparison stops if the algorithm encounters unequal symbols, or if it reaches the end of the block. In both cases, the next blocks are then compared, and so on until the last block was compared.

The corresponding cost $\mathcal{C}_B$ can be expressed as

$$\mathcal{C}_B = M\left[\sum_{i=1}^{L_B-1}\Big[(1-\mathbb{P})^{i-1}(i\mathbb{P})\Big] + (1-\mathbb{P})^{L_B-1}(L_B\mathbb{P}) + L_B(1-\mathbb{P})^{L_B}\right] \tag{6.2}$$

where M corresponds to the number of blocks to compare. Except for considering blocks of length L_B rather than the entire file, all three terms are the same as in equation (6.1).

PBDA complexity

The PBDA [125] only compares the pivot part of each block rather than the whole block. We ignore the case where an edit exists in the pivot because it stops the execution of the PBDA, *i.e.*, the deduplication is aborted. The cost $\mathcal{C}_{\text{PBDA}}$ is then expressed as

$$\mathcal{C}_{\text{PBDA}} = M\Big[L_p(1-\mathbb{P})^{L_s} + L_p^2\Big(1-(1-\mathbb{P})^{L_s}\Big)\Big] \tag{6.3}$$

In this expression, the term $L_p(1-\mathbb{P})^{L_s}$ corresponds to the average number of comparisons given that the segment $\mathbf{s}$ of length L_s is not edited. The term $L_p^2\Big(1-(1-\mathbb{P})^{L_s}\Big)$ corresponds to the average number of comparisons given that the segment $\mathbf{s}$ contains at least one edition, where L_p^2 correponds to the number of performed comparisons to determine the drift d_n.

PBDA-SW complexity

Like the PBDA, our PBDA-SW only compares the pivot part of each block. However, if there is an edit in a pivot, our algorithm can continue the deduplication process after performing some extra comparisons through the sliding-window. The cost $\mathcal{C}_{\text{PBDA-SW}}$ is

then expressed as

$$\begin{aligned}\mathcal{C}_{\text{PBDA-SW}} = M\Big[\ & L_P(1-\mathbb{P})^{L_B} \\ & + {L_P}^2 n_s (1-\mathbb{P})^{L_S}\Big(1-(1-\mathbb{P})^{L_P}\Big) \\ & + {L_P}^2 n_s \Big(1-(1-\mathbb{P})^{L_S}\Big)\Big(1-(1-\mathbb{P})^{L_P}\Big) \\ & + {L_P}^2 [1-(1-\mathbb{P})^{L_S}](1-\mathbb{P})^{L_P}\Big]\end{aligned} \tag{6.4}$$

The first term corresponds to the average number of comparisons given that there is no edit in the block (segment and pivot). The second term corresponds to the average number of comparisons given that there is no edit in the segment **s**, but there is at least one edit in the pivot **p**. The third term corresponds to the average number of comparisons given that there is at least one edit in the segment **s** and at least one edit in the pivot **p**. The fourth term corresponds to the average number of comparisons given that there is at least one edit in the segment **s** but no edit in the pivot **p**. The parameters M and n_s correspond to the number of blocks in **x** and the number of maximum slides of the sliding-window, respectively.

Numerical cost comparison

We consider an initial file of length $N = 1.2 \times 10^5$, and we evaluate the previous costs given various edit probabilities $\mathbb{P}$, see Figures 6.4 and 6.5. The better algorithms are those with the smaller costs. In the case of the PBDA-SW, we consider various maximum number of slides $n_s = \{L_p,\ 2L_p,\ 3L_p\}$.

In Figure 6.4 we consider segments of length $L_s = 94$ and pivots of length $L_p = 6$. Hence, the number of blocks is $M = 1200$. In the case of Figure 6.5, we consider segments of length $L_s = 9994$ and pivots of length $L_p = 6$. Hence, the number of blocks is $M = 12$. In both figures, we observe that for low edit probabilities, the number of compared symbols is the smallest in both the PBDA and PBDA-SW, and that the PBDA-SW number of comparisons is practically the same as in the PBDA. This is because the probability to observe an error in the pivot is low. Therefore, the sliding-window is rarely used. We also observe that when $\mathbb{P} > 10^{-4}$, the PBDA-SW cost starts to increase because the sliding-window is often used. In addition, although for larger edit probabilities the file-level BFM has a small number of comparisons, its deduplication ratios will become very poor as we will see in the next section. Furthermore, the block-level deduplication has the highest number of comparisons, and until $\mathbb{P} = 10^{-2}$, it is worst than the PBDA-SW.

Therefore, compared to the PBDA, the complexity of our PBDA-SW only increases slightly when $\mathbb{P} > 10^{-4}$ because the sliding-window is more often used.

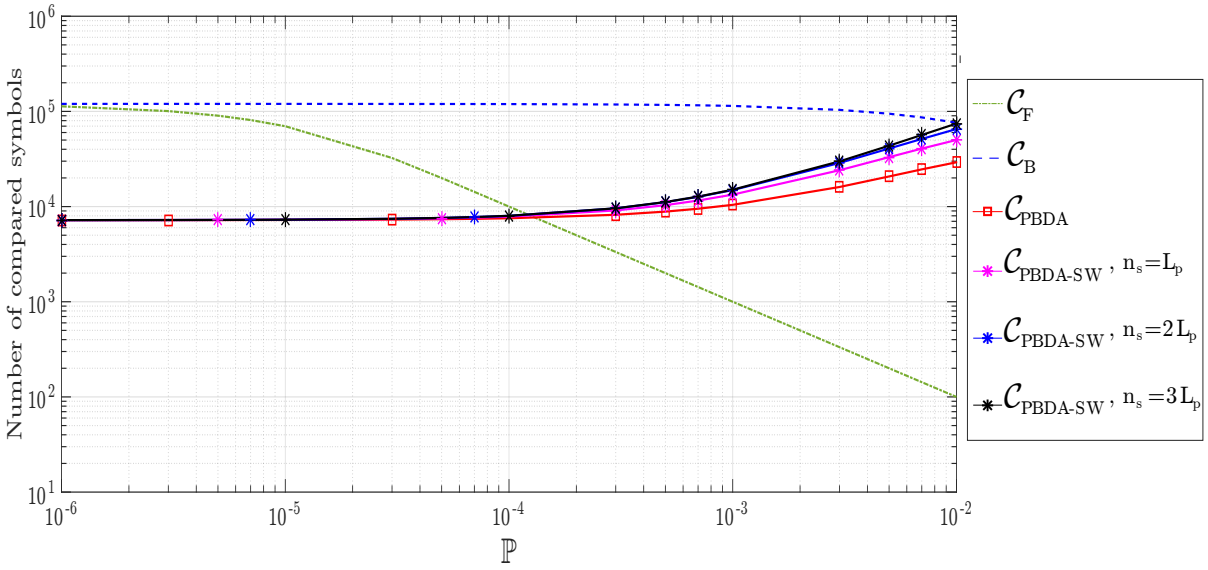


Figure 6.4: Number of compared symbols as a function of $\mathbb{P}$. This figure considers parameters $N = 120000$, $L_s = 94$, $L_p = 6$, $M = 1200$.

6.5.2 Deduplication ratios

We also evaluate the algorithms in terms of deduplication ratios through numerical simulations. Each simulation run considers a couple of files $\mathbf{x}$ and $\mathbf{y}$. The file $\mathbf{x}$ of length $n = 120000$ is randomly generated, and each of its elements x_i takes value in the alphabet set $\mathbf{\Omega} = \{0, \ldots, q-1\}$ uniformly at random, with $q = 64$. We perform 1000 simulation runs for each considered value of edit probability $\mathbb{P}$.

Figure 6.6 shows the deduplication ratio for the considered algorithms with respect to edit probabilities $\mathbb{P}$. Except for the BFM file algorithm, we consider two cases for each algorithm. In the first case, we fix $L_s = 94$ and $L_p = 6$ (*i.e.*, $M = 1200$), while in the second case we fix $L_s = 9994$ and $L_p = 6$ (*i.e.*, $M = 12$). It is not surprising to observe that the BFM block algorithm has the best results, because it goes through each block symbol by symbol. Hence, if there is a deduplicable block, the BFM block will find it, but at a high cost. In addition and very interestingly, the PBDA-SW allows to significantly increase the performance of the initial PBDA in [125], and it almost fits the BFM block algorithm performance for a much lower cost. Indeed, when $\mathbb{P} < 10^{-4}$, compared to the

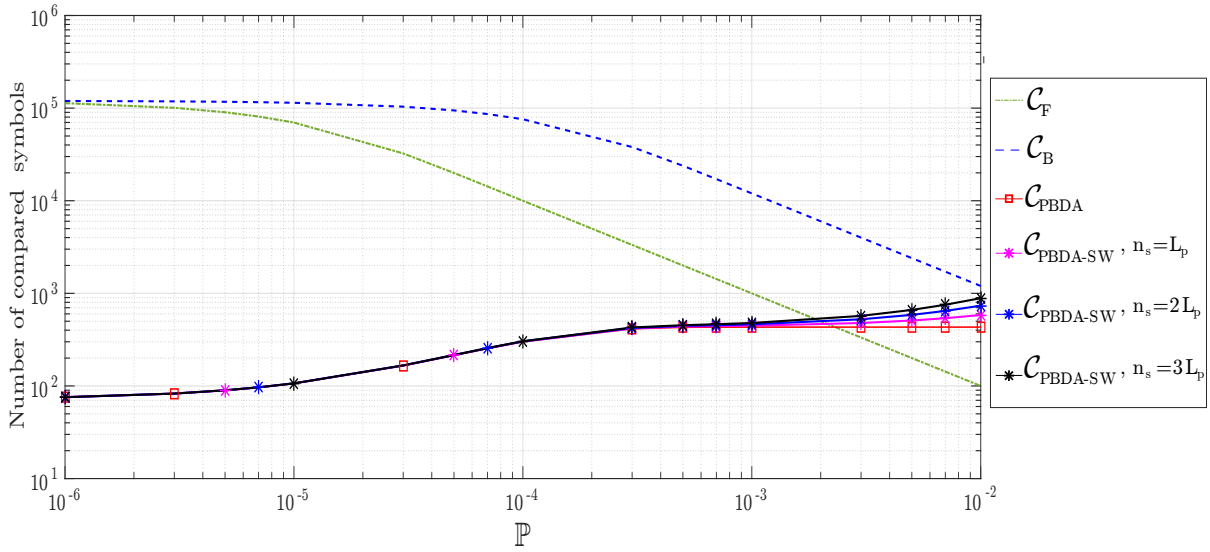


Figure 6.5: Number of compared symbols as a function of $\mathbb{P}$. This figure considers parameters $N = 120000$, $L_s = 9994$, $L_p = 6$, $M = 12$.

BFM algorithm block, the PBDA-SW reduces the deduplication complexity by a factor of 10 when $L_s = 94$ and $L_p = 6$, and by almost a factor of 10^3 when $L_s = 9994$ and $L_p = 6$. The BFM file deduplication ratios are smaller because if only one symbol of the file is edited, then the deduplication fails. We can also notice that for all the considered algorithms, the deduplication ratio starts to drastically decrease when $\mathbb{P} > 10^{-3}$. This is because there are errors on most of the blocks.

Figure 6.7 shows the impact of the maximum number of slides n_s on the performance of the PBDA-SW. In this figure, we consider different values of parameters M and n_s with respect to $\mathbb{P}$. In the first case, we consider $M = 1200$ (*i.e.*, $L_s = 94$ and $L_p = 6$), and three different values of $n_s = \{L_p, 2L_p, 3L_p\}$. We observe that compared to $n_s = L_p$, setting $n_s = 2L_p$ and $n_s = 3L_p$ allow to improve the deduplication ratios, particularly when $\mathbb{P}$ increases. This is because the deduplication is operated with a smaller granularity (large number of small blocks). Therefore, the sliding-window is often used. The deduplication ratios between $n_s = 2L_p$ and $n_s = 3L_p$ are quite the same because when $\mathbb{P}$ is small, sliding the window by only a few positions is enough to remove the edits from the window. On the opposite, for large values of $\mathbb{P}$, considering $n_s = 3L_p$ does not improve the results because other errors can be encountered on the $3L_P$ positions. Therefore, $n_s = 2L_p$ offers the best tradeoff between cost and performance. For the other cases where $M = 120$ (*i.e.*, $L_s = 994$ and $L_p = 6$) or $M = 12$ (*i.e.*, $L_s = 9994$ and $L_p = 6$), increasing n_s has a slight effect or no effect at all. This is because the granularity is too high (few number of large

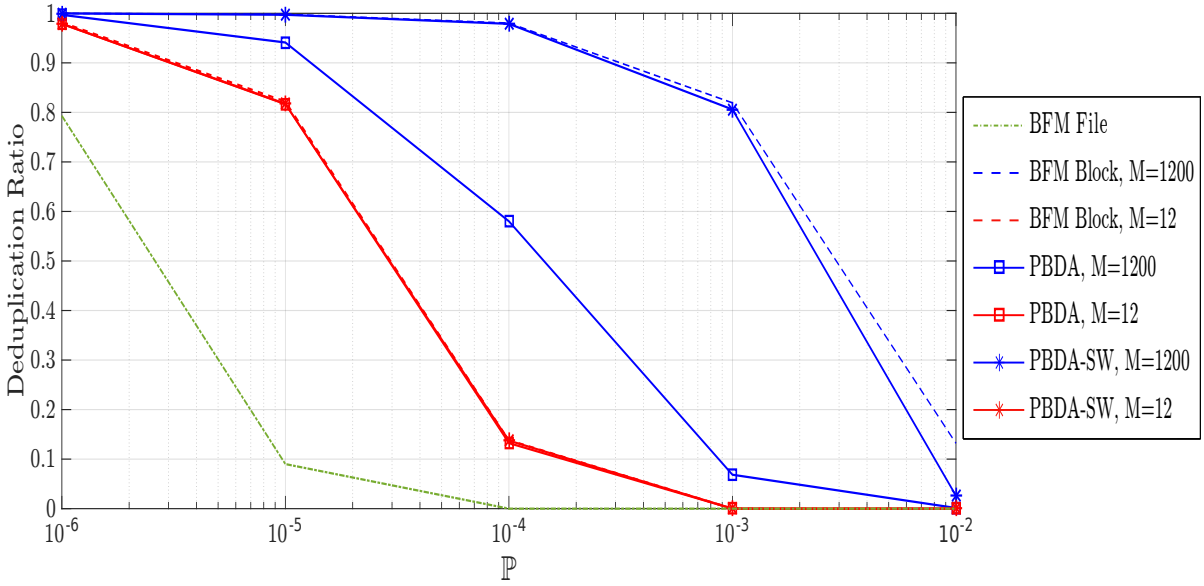


Figure 6.6: Data deduplication ratio as a function of $\mathbb{P}$ for the four different algorithms. For the PD and SW-PD, we consider $L_p = 6$, $L_s = 94$ (i.e., $M = 1200$), and $L_p = 6$, $L_s = 9994$ (i.e., $M = 12$).

blocks). Therefore, the probability to observe an edit in the pivot decreases, and hence, the sliding-window is not used much.

Therefore, the effect of n_s is more important when dealing with a smaller granularity. Furthermore, it is not necessary to consider a large values of sliding n_s because for small values of $\mathbb{P}$ only a few slides are necessary, while for large values of $\mathbb{P}$, other edits appears after a few slides.

6.6 PBDA-SW in the context of DNA data storage

We now discuss how the PBDA-SW could be used in the context of DNA data storage. The edit channel described in Section 6.2 is very similar to the DNA data storage channel. Indeed, the DNA data storage channel takes as input a sequence $\mathbf{x}$ and outputs V sequences $\mathbf{y}^{(\mathbf{v})}$ ($v \in [\![1, V]\!]$), where each sequence $\mathbf{y}^{(\mathbf{v})}$ is an edited version of $\mathbf{x}$. Moreover, depending on the amount of sequenced data, the FastQ file produced by the sequencer contains a few to hundreds gigabytes of redundant sequences ($\mathbf{y}^{(\mathbf{v})}$). Therefore, it could be possible to deduplicate the $\mathbf{y}^{(\mathbf{v})}$ sequences, for storage on a DSS.

However, the DNA data storage channel presented in Chapter 3 has a high edit probability $\mathbb{P}$ over the sequences $\mathbf{y}^{(\mathbf{v})}$. Indeed, depending on the sequencing protocol, $\mathbb{P}$ can vary

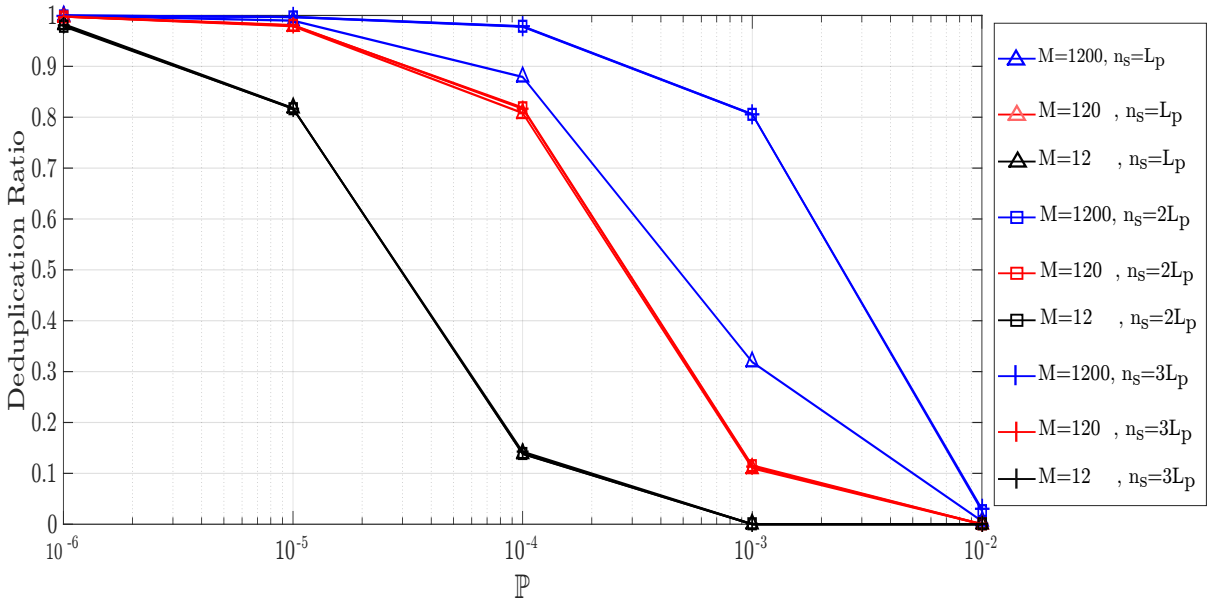


Figure 6.7: Deduplication ratios of the PBDA-SW, with respect to the maximum number of slides. This figure considers parameters $L_P = 6$, $M = \{12, 120, 1200\}$ and $n_s = \{L_p, 2L_p, 3L_p\}$.

from 10^{-2} to 10^{-1} [19, 20]. Furthermore, the DNA data storage channel also introduces substitutions. As a result, $\mathbb{P} = \mathbb{P}_i + \mathbb{P}_d + \mathbb{P}_s$, and $\mathbb{P}_i \neq \mathbb{P}_d$. Therefore, the PBDA-SW should be further improved: (i) tackle higher error probabilities, (ii) handle substitution errors, (iii) consider the asymetric case $\mathbb{P}_i \neq \mathbb{P}_d$.

Interestingly, some notions used in the PBDA-SW, show connections with some ideas developed in the previous chapters on channel coding. First, in Chapter 4, the sequence output by the consensus has a low error probability, and it is divided into small blocks for the synchronisation. Although it does not rely on a sliding window, the synchronisation algorithm tries to insert some bases on each considered block. Second, in Chapter 5, the CC decoders use the drift variables d_n as state variables.

6.7 Conclusion

In this chapter, we introduced the PBDA-SW as an extended version of the PBDA initially proposed in [125] for data deduplication. We saw through a theoretical analysis and through numerical simulations that our PBDA-SW significantly improves the deduplication performance at the expense of a slightly increased complexity. However, the

deduplication ratios of the PBDA-SW remain poor when the probability of edit $\mathbb{P} \geq 10^{-2}$. Therefore, to use the PBDA-SW in DNA data storage, it should be further improved.

Chapter 7

Conclusion & Perspectives

In this thesis, we addressed several issues toward implementing practical DNA data storage systems. We first proposed a memory channel model, which accurately represents the DNA data storage channel. In addition, we proposed and evaluated two error-correction solutions. Especially, the second solution based on convolutional codes allowed for an important gain in performance compared to the first solution which relies on a consensus algorithm followed by LDPC codes. In addition, the second solution improved state-of-the-art convolutional codes for DNA data storage, by taking into account our memory channel model. Finally, we also proposed a data deduplication algorithm called PBDA-SW, which showed improved data deduplication performance compared to existing solutions, while maintaining a low algorithm complexity.

We now describe some perspectives to improve the proposed solutions, and we identify other important research directions.

7.1 Memory channel model

In this section, we describe some perspectives that could improve the memory channel model:

— **Consider chimeric and unaligned reads:** when training our memory channel model, both unaligned and chimeric reads were discarded from the training process. These reads were ignored because the length of the unaligned parts were too large. We observed that these reads are either completely different from the original sequence [136], or only show some small similarities. Thus, they introduce some specific error patterns which should be taken into account during the training. The amount of chimeric reads is relatively small compared to the amount of total reads. For instance, in [136] and [137] the amount of chimeric reads is about 1%, while in our *SetG* and *SetE*, the amount of chimeric reads is about 2% and 10%, respectively.

One possible solution to take chimeric reads into account could be to consider two running modes for the memory channel model. One mode would be dedicated to the chimeric and unaligned reads, and another one would be dedicated to primary reads. The memory channel model could then switch from a mode to another during the simulation.

— **Build a genomic data simulator:** the proposed memory channel model allows to simulate a whole sequence, *i.e.*, the model takes as input a sequence $\mathbf{x}$ and outputs V sequences $\mathbf{y}^{(v)}$, which are edited version of the whole sequence $\mathbf{x}$. This behavior is well adapted to the case of DNA data storage. However, when considering genomic data, a sequence $\mathbf{x}$ can produce V sequences $\mathbf{y}^{(v)}$, but each sequence $\mathbf{y}^{(v)}$ is an edited version of a small region (substring) of $\mathbf{x}$. Therefore, it could be interesting to find a way to model the sequences $\mathbf{y}^{(v)}$ so that they correspond to regions of $\mathbf{x}$ rather than to the whole sequence. Hence, this model could also be used for genomic simulations, which would be of interest for the bioinformatics community.

— **Sequencing protocol database:** in our work, we trained our memory channel model on data sequenced with a particular procotol. We used a specific library preparation kit (used to prepare the DNA), a specific flongle flow cell (sensor that is used to sequence the DNA), and basecalled with a specific version of guppy. Hence, given that some experimental data is accessible, it would be interesting to build various error profiles, each one related to a particular sequencing protocol. This would allow users to choose the error profile of the memory channel model that best fits their needs.

7.2 CSL solution

In this section, we describe solutions that could improve the performance of the CSL decoding scheme. At first, in this solution, there is a need to improve the consensus algorithm. But the synchronization algorithm could also be improved as follows.

This algorithm splits the sequence into segments of length l_s, and then arbitrary inserts the base "A" at the beginning of each segment to compute the score $\mathcal{S}$. To further improve the synchronization performance, it could be interesting to insert the arbitrary base "A" by dichotomy over the sequence $\overline{\overline{\mathbf{y}_{cons}}}$.

The code rate of the NB-LDPC code could also be optimized. Indeed, the amount of errors after the consensus step is relatively small. Hence, if $\overline{\overline{\mathbf{y}_{cons}}}$ is correctly synchronized,

the NB-LDPC decoder could correct the remaining errors even with a high code rate.

7.3 CC decoder

The CC decoder called *Dec3* in Chapter 5 could be improved in terms of algorithm complexity and performance. We now describe some of the possible improvements.

— **Consider a smaller memory order** k: we observed in Figure 3.8 that the memory channel model is also accurate for values of k smaller than 6. Therefore, one could consider a smaller memory order k (*i.e.*, $k < 6$) in the decoder, which would reduce the decoding complexity while maintaining the performance hopefully better than *Dec1* and close to *Dec3* (with $k = 6$).

— **Consider dynamic parameters** D_{min} **and** D_{max}: currently, we consider that $D_{min} = D_{max}$. However, given that in our model $\mathbb{P}_i \neq \mathbb{P}_d$ and $\mathbb{P}_i < \mathbb{P}_d$, considering $D_{min} \neq D_{max}$ could decrease the complexity of the algorithm since several nodes of the trellis would be removed. D_{min} and D_{max} should be fixed according to the edit probabilities of our memory channel model.

— **Consider forward and reverse strands**: except in the case of long homopolymers, considering both forward and reverse strands could improve the performance of *Dec3*. This is because while a k-mer on the forward strand introduces errors on a particular position, its reverse on the reverse strand may be less sensitive to errors. However, in this case, it would be necessary to add a step which identifies forward and reverse reads before decoding them. The identification could be done either by adding two different short sequences of about 10 bases at the beginning and the end of the strand to synthesize, or by performing alignments between the reads. The first solution would increase the synthesis cost, while decreasing the identification complexity. The second solution would not require to add bases to the strand to synthesize, but the identification complexity would increase.

— **Consider FastQ files metadata**: among the metadata produced by the sequencer, a score called "phred score" indicates the reliability of the basecalling on each symbol of the sequence. The phred score can also be expressed as a probability of error. Hence, it could be possible to initialize the decoder *a priori* probabilities with this information, so as to improve the decoder performance.

7.4 Toward a reliable DNA data storage decoder

Error-correction decoding can be placed at different stages after DNA sequencing, but the best decoding performance could maybe be obtained when directly decoding the electrical current signal. This is because the current signal contains an analog information that is richer than the digital information (*e.g.*, DNA bases values) contained in the FasQ file. In addition, the data in the FastQ file may contain additional errors introduced by the basecaller.

The basecaller in the MinION relies on a DeepLearning approach to transform the current signal into DNA bases. To the best of our knowledge, there is no practical solution that purely relies on channel coding to convert the current signal into DNA bases. This is not surprising since the MinION sequencer is mainly used for genomic data experiments, so as to sequence the genome of living organisms, which of course do not contain channel codes.

However, in the case of DNA data storage, we can encode the DNA sequences. Therefore, rather than using the basecaller, it could be possible to rely on a channel decoder in order to transform the current signal into DNA bases while correcting errors at the same time. Furthermore, since the basecalling is a high time-consuming step, the channel decoder could maybe reduce DNA sequencing time.

PUBLICATIONS

— Laura Conde-Canencia, Belaid Hamoum, Emeline Roux, and Dominique Lavenier, "Error Correction Schemes for DNA Storage with Nanopore", Journées Ouvertes Biologie, Informatique et Mathématiques (JOBIM), 2019.

— Laura Conde-Canencia and Belaid Hamoum, "Deduplication algorithms and models for efficient data storage", 24th International Conference on Circuits, Systems, Communications and Computers (CSCC), 2020.

— Belaid Hamoum, Elsa Dupraz, Laura Conde-Canencia, and Dominique Lavenier, "Channel Model with Memory for DNA Data Storage with Nanopore Sequencing", 11th International Symposium on Topics in Coding (ISTC), 2021.

— Belaid Hamoum, Elsa Dupraz, and Laura Conde-Canencia, "A DNA Data Storage Channel Model Trained on Genomic Data with Nanopore Sequencing", 1st International Conference on Data Storage in Molecular Media (DSMM), 2022.

— **In preparation for submission to a journal**: Belaid Hamoum, Lorenz Welter, Andreas Lenz, Anisha Banerjee, Elsa Dupraz, Dominique Lavenier, and Antonia Wachter-Zeh, "Channel Model and Decoder with Memory for DNA Data Storage with Nanopore Sequencing".

Bibliography

[1] David Reinsel, John Gantz, and John Rydning, « The Digitization of the World. From Edge to Core. », *in*: (2018).

[2] Luis Ceze, Jeff Nivala, and Karin Strauss, « Molecular digital data storage using DNA », *in*: *Nature Reviews Genetics* 20.*8* (Aug. 2019), pp. 456–466.

[3] S. Bobba et al., « Critical Raw Materials for Strategic Technologies and Sectors in the EU », *in*: (2020).

[4] Kazuo Goda and Masaru Kitsuregawa, « The History of Storage Systems », *in*: *Proceedings of the IEEE* 100.*Special Centennial Issue* (2012), pp. 1433–1440, DOI: 10.1109/JPROC.2012.2189787.

[5] George M. Church, Yuan Gao, and Sriram Kosuri, « Next-Generation Digital Information Storage in DNA », *in*: *Science* 337.*6102* (2012), pp. 1628–1628, ISSN: 0036-8075.

[6] Nick Goldman et al., « Towards practical, high-capacity, low-maintenance information storage in synthesized DNA », *in*: *Nature* 494.*7435* (2013), nature11875[PII], pp. 77–80, ISSN: 1476-4687.

[7] Robert N Grass et al., « Robust chemical preservation of digital information on DNA in silica with error-correcting codes », en, *in*: *Angew. Chem. Int. Ed Engl.* 54.*8* (Feb. 2015), pp. 2552–2555.

[8] James Bornholt et al., « A DNA-Based Archival Storage System », *in*: *SIGARCH Comput. Archit. News* 44.*2* (Mar. 2016), pp. 637–649, ISSN: 0163-5964, DOI: 10.1145/2980024.2872397, URL: https://doi.org/10.1145/2980024.2872397.

[9] Alexandre Maes et al., « La révolution de l'ADN: biocompatible and biosafe DNA data storage », *in*: *bioRxiv* (2022), DOI: 10.1101/2022.08.25.505104.

[10] Lee Organick et al., « An Empirical Comparison of Preservation Methods for Synthetic DNA Data Storage », *in*: *Small Methods* 5.*5* (2021), p. 2001094, DOI: https://doi.org/10.1002/smtd.202001094.

[11] Norbert Wiener, « Interview: machines smarter than men », *in*: *US News World Rep* 56 (1964), pp. 84–6.

[12] MS Neiman, « On the molecular memory systems and the directed mutations », *in*: *Radiotekhnika* 6 (1965), pp. 1–8.

[13] Christopher N. Takahashi et al., « Demonstration of End-to-End Automation of DNA Data Storage », *in*: *Scientific Reports* 9.*1* (2019), p. 4998, ISSN: 2045-2322.

[14] Jorge S Reis-Filho, « Next-generation sequencing », *in*: *Breast Cancer Research* 11.*3* (Dec. 2009), S12.

[16] Min Hao, Jianjun Qiao, and Hao Qi, « Current and Emerging Methods for the Synthesis of Single-Stranded DNA », *in*: *Genes* 11.*2* (2020), ISSN: 2073-4425, DOI: 10.3390/genes11020116, URL: https://www.mdpi.com/2073-4425/11/2/116.

[17] Yan Han, « Cloud storage for digital preservation: optimal uses of Amazon S3 and Glacier », *in*: *Library Hi Tech* 33.*2* (Jan. 2015), pp. 261–271, ISSN: 0737-8831, DOI: 10.1108/LHT-12-2014-0118, URL: https://doi.org/10.1108/LHT-12-2014-0118.

[18] Amazon, *Amazon Glacier storage*, https://docs.aws.amazon.com/amazonglacier/latest/dev/introduction.html, Accessed on 14.09.2022.

[19] Reinhard Heckel, Gediminas Mikutis, and Robert N. Grass, « A Characterization of the DNA Data Storage Channel », en, *in*: *Scientific Reports* 9.*1* (2019), p. 9663, ISSN: 2045-2322, (visited on 10/08/2020).

[20] Clara Delahaye and Jacques Nicolas, « Sequencing DNA with nanopores: Troubles and biases », *in*: *PLOS ONE* 16.*10* (Oct. 2021), pp. 1–29, DOI: 10.1371/journal.pone.0257521, URL: https://doi.org/10.1371/journal.pone.0257521.

[21] Emily M LeProust et al., « Synthesis of high-quality libraries of long (150mer) oligonucleotides by a novel depurination controlled process », en, *in*: *Nucleic Acids Res.* 38.*8* (May 2010), pp. 2522–2540.

[22] Yunhao Wang et al., « Nanopore sequencing technology, bioinformatics and applications », *in*: *Nature Biotechnology* 39.*11* (Nov. 2021), pp. 1348–1365.

[23] Wentu Song et al., « Codes With Run-Length and GC-Content Constraints for DNA-Based Data Storage », *in*: *IEEE Communications Letters* 22.*10* (2018), pp. 2004–2007, DOI: 10.1109/LCOMM.2018.2866566.

[24] Chen Yang et al., « NanoSim: nanopore sequence read simulator based on statistical characterization », *in*: *GigaScience* 6.*4* (Apr. 2017), gix010.

[25] C. Berrou and A. Glavieux, « Near optimum error correcting coding and decoding: turbo-codes », *in*: *IEEE Transactions on Communications* 44.*10* (1996), pp. 1261–1271, DOI: 10.1109/26.539767.

[26] David JC MacKay and Radford M Neal, « Near Shannon limit performance of low density parity check codes », *in*: *Electronics letters* 33.*6* (1997), pp. 457–458.

[27] Silvio A Abrantes, « From BCJR to turbo decoding: MAP algorithms made easier », *in*: (2004).

[28] Wen Xia et al., « A Comprehensive Study of the Past, Present, and Future of Data Deduplication », *in*: *Proceedings of the IEEE* 104.*9* (2016), pp. 1681–1710, DOI: 10.1109/JPROC.2016.2571298.

[29] DnarXiv, *DnarXiv project*, https://project.inria.fr/dnarxiv/.

[30] O T Avery, C M Macleod, and M McCarty, « STUDIES ON THE CHEMICAL NATURE OF THE SUBSTANCE INDUCING TRANSFORMATION OF PNEUMOCOCCAL TYPES : INDUCTION OF TRANSFORMATION BY A DESOXYRIBONUCLEIC ACID FRACTION ISOLATED FROM PNEUMOCOCCUS TYPE III », en, *in*: *J Exp Med* 79.*2* (Feb. 1944), pp. 137–158.

[31] Ralf Dahm, « Discovering DNA: Friedrich Miescher and the early years of nucleic acid research », *in*: *Human Genetics* 122.*6* (Jan. 2008), pp. 565–581, ISSN: 1432-1203, DOI: 10.1007/s00439-007-0433-0, URL: https://doi.org/10.1007/s00439-007-0433-0.

[32] Richard R Sinden, *DNA structure and function*, Gulf Professional Publishing, 1994.

[34] J D Watson and F H C Crick, « Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid », *in*: *Nature* 171.*4356* (Apr. 1953), pp. 737–738.

[35] Leslie Pray, « Discovery of DNA structure and function: Watson and Crick », *in*: *Nature Education* 1.*1* (2008).

[36] Randall A Hughes and Andrew D Ellington, « Synthetic DNA synthesis and assembly: Putting the synthetic in synthetic biology », en, *in*: *Cold Spring Harb. Perspect. Biol.* 9.*1* (Jan. 2017), a023812.

[37] Geoffrey M Cooper, *DNA Replication*, Sunderland, MA: Sinauer Associates, 2000.

[38] Chris Simon, Adrian Franke, and Andrew Martin, « The Polymerase Chain Reaction: DNA Extraction and Amplification », *in*: *Molecular Techniques in Taxonomy*, ed. by Godfrey M. Hewitt, Andrew W. B. Johnston, and J. Peter W. Young, Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 329–355, ISBN: 978-3-642-83962-7, DOI: 10.1007/978-3-642-83962-7_22, URL: https://doi.org/10.1007/978-3-642-83962-7_22.

[39] S M Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic, « Portable and Error-Free DNA-Based Data Storage », *in*: *Scientific Reports* 7.*1* (July 2017), p. 5011.

[40] Eojin Yoo et al., « Mini review: Enzyme-based DNA synthesis and selective retrieval for data storage », *in*: *Computational and Structural Biotechnology Journal* 19 (2021), pp. 2468–2476, ISSN: 2001-0370, DOI: https://doi.org/10.1016/j.csbj.2021.04.057, URL: https://www.sciencedirect.com/science/article/pii/S2001037021001690.

[41] Michael J. Czar et al., « Gene synthesis demystified », *in*: *Trends in Biotechnology* 27.*2* (2009), pp. 63–72, ISSN: 0167-7799, DOI: https://doi.org/10.1016/j.tibtech.2008.10.007, URL: https://www.sciencedirect.com/science/article/pii/S0167779908002850.

[42] Sriram Kosuri and George M Church, « Large-scale de novo DNA synthesis: technologies and applications », *in*: *Nature Methods* 11.*5* (May 2014), pp. 499–507.

[43] Phillip Kuhn et al., « Next generation gene synthesis: From microarrays to genomes », en, *in*: *Eng. Life Sci.* 17.*1* (Jan. 2017), pp. 6–13.

[44] Michael Eisenstein, « Enzymatic DNA synthesis enters new phase », *in*: *Nature Biotechnology* 38.*10* (Oct. 2020), pp. 1113–1115.

[45] F Sanger et al., « Nucleotide sequence of bacteriophage φX174 DNA », en, *in*: *Nature* 265.*5596* (Feb. 1977), pp. 687–695.

[46] Ayman Grada and Kate Weinbrecht, « Next-generation sequencing: methodology and application », *in*: *The Journal of investigative dermatology* 133.*8* (2013), e11.

[47] Sam Behjati and Patrick S Tarpey, « What is next generation sequencing? », *in*: *Archives of Disease in Childhood - Education and Practice* 98.*6* (2013), pp. 236–238, ISSN: 1743-0585, DOI: 10.1136/archdischild-2013-304340, eprint: https://ep.bmj.com/content/98/6/236.full.pdf, URL: https://ep.bmj.com/content/98/6/236.

[48] Xiangchun Zhou, Xufeng Bai, and Yongzhong Xing, « A rice genetic improvement boom by next generation sequencing », en, *in*: *Curr. Issues Mol. Biol.* (2018), pp. 109–126.

[49] Eugene Y. Chan, « Next-Generation Sequencing Methods: Impact of Sequencing Accuracy on SNP Discovery », *in*: *Single Nucleotide Polymorphisms: Methods and Protocols*, ed. by Anton A. Komar, Totowa, NJ: Humana Press, 2009, pp. 95–111, ISBN: 978-1-60327-411-1, DOI: 10.1007/978-1-60327-411-1_5, URL: https://doi.org/10.1007/978-1-60327-411-1_5.

[50] Jose Antonio Garrido-Cardenas et al., « DNA Sequencing Sensors: An Overview », *in*: *Sensors* 17.*3* (2017), ISSN: 1424-8220, DOI: 10.3390/s17030588, URL: https://www.mdpi.com/1424-8220/17/3/588.

[51] Hayan Lee et al., « Third-generation sequencing and the future of genomics », *in*: *bioRxiv* (2016), DOI: 10.1101/048603, eprint: https://www.biorxiv.org/content/early/2016/04/13/048603.full.pdf, URL: https://www.biorxiv.org/content/early/2016/04/13/048603.

[52] Tiantian Xiao and Wenhao Zhou, « The third generation sequencing: the advanced approach to genetic diseases », *in*: *Transl. Pediatr.* 9.*2* (Apr. 2020), pp. 163–173.

[53] Andrea D. Tyler et al., « Evaluation of Oxford Nanopore's MinION Sequencing Device for Microbial Whole Genome Sequencing Applications », *in*: *Scientific Reports* 8.*1* (July 2018), p. 10931, ISSN: 2045-2322, DOI: 10.1038/s41598-018-29334-5, URL: https://doi.org/10.1038/s41598-018-29334-5.

[54] Travis C Glenn, « Field guide to next-generation DNA sequencers », en, *in*: *Mol. Ecol. Resour.* 11.*5* (Sept. 2011), pp. 759–769.

[55] S M Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic, « Portable and Error-Free DNA-Based Data Storage », *in*: *Scientific Reports* 7.*1* (July 2017), p. 5011.

[58] Yaniv Erlich and Dina Zielinski, « DNA Fountain enables a robust and efficient storage architecture », *in*: *Science* 355.*6328* (2017), pp. 950–954, ISSN: 0036-8075.

[59] Christy Agbavwe et al., « Efficiency, error and yield in light-directed maskless synthesis of DNA microarrays », en, *in*: *J. Nanobiotechnology* 9.*1* (Dec. 2011).

[60] Nick Vereecke et al., « High quality genome assemblies of Mycoplasma bovis using a taxon-specific Bonito basecaller for MinION and Flongle long-read nanopore sequencing », *in*: *BMC Bioinformatics* 21.*1* (Nov. 2020), p. 517, ISSN: 1471-2105, DOI: 10.1186/s12859-020-03856-0, URL: https://doi.org/10.1186/s12859-020-03856-0.

[61] Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt, « Performance of neural network basecalling tools for Oxford Nanopore sequencing », *in*: *Genome Biology* 20.*1* (2019), p. 129, ISSN: 1474-760X.

[62] Shubham Chandak et al., « Improved read/write cost tradeoff in DNA-based data storage using LDPC codes », *in*: *Allerton*, 2019, pp. 147–156, DOI: 10.1109/ALLERTON.2019.8919890.

[63] Ethan Alexander Garcıa Baker et al., « SiLiCO: A Simulator of Long Read Sequencing in PacBio and Oxford Nanopore », *in*: *bioRxiv* (2016), DOI: 10.1101/076901, eprint: https://www.biorxiv.org/content/early/2016/09/22/076901.full.pdf, URL: https://www.biorxiv.org/content/early/2016/09/22/076901.

[64] Shubham Chandak et al., « Overcoming High Nanopore Basecaller Error Rates for DNA Storage via Basecaller-Decoder Integration and Convolutional Codes », *in*: *ICASSP*, 2020, pp. 8822–8826, DOI: 10.1109/ICASSP40776.2020.9053441.

[65] Yu Li et al., « DeepSimulator: a deep simulator for Nanopore sequencing », *in*: *Bioinformatics* 34.*17* (2018), pp. 2899–2908, ISSN: 1367-4803.

[66] Ryan R. Wick, « Badread: simulation of error-prone long reads », *in*: *Journal of Open Source Software* 4.*36* (2019), p. 1316, DOI: 10.21105/joss.01316, URL: https://doi.org/10.21105/joss.01316.

[67] Belaid Hamoum et al., « Channel Model with Memory for DNA Data Storage with Nanopore Sequencing », *in*: *2021 11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1–5, DOI: 10.1109/ISTC49272.2021.9594243.

[68] Lawrence R Rabiner, « A tutorial on hidden Markov models and selected applications in speech recognition », *in*: *Proceedings of the IEEE* 77.*2* (1989), pp. 257–286.

[70] ENA, *database*, https://www.ebi.ac.uk/ena/browser/home.

[71] GenBank, *database*, https://www.ncbi.nlm.nih.gov/genbank/.

[72] Aaron David Goldman and Laura F. Landweber, « What Is a Genome? », *in*: *PLOS Genetics* 12.*7* (July 2016), pp. 1–7, DOI: 10.1371/journal.pgen.1006181, URL: https://doi.org/10.1371/journal.pgen.1006181.

[73] Monya Baker, « De novo genome assembly: what every biologist should know », *in*: *Nature Methods* 9.*4* (Apr. 2012), pp. 333–337.

[74] Ali Karami, « Largest and Smallest Genome in the World », *in*: (Jan. 2013).

[75] Belaid Hamoum, Elsa Dupraz, and Laura Conde-Canencia, « A DNA Data Storage Channel Model Trained on Genomic Data with Nanopore Sequencing », *in*: *1st International Conference on Data Storage in Molecular Media (DSMM)*, 2022, URL: https://mosla.mathematik.uni-marburg.de/wp-content/uploads/2022/03/DSMM2022.pdf.

[76] Wenjun Li, Didier Raoult, and Pierre-Edouard Fournier, « Bacterial strain typing in the genomic era », *in*: *FEMS Microbiology Reviews* 33.*5* (Sept. 2009), pp. 892–916, ISSN: 0168-6445, DOI: 10.1111/j.1574-6976.2009.00182.x, eprint: https://academic.oup.com/femsre/article-pdf/33/5/892/18141693/33-5-892.pdf, URL: https://doi.org/10.1111/j.1574-6976.2009.00182.x.

[77] Emeline Roux et al., « The genomic basis of the Streptococcus thermophilus health-promoting properties », *in*: *BMC Genomics* 23.*1* (Mar. 2022), p. 210.

[78] Heng Li et al., « The Sequence Alignment/Map format and SAMtools », *in*: *Bioinformatics* 25.*16* (Aug. 2009), pp. 2078–2079.

[79] Heng Li, « Minimap2: pairwise alignment for nucleotide sequences », *in*: *Bioinformatics* 34.*18* (May 2018), pp. 3094–3100, ISSN: 1367-4803, DOI: 10.1093/bioinformatics/bty191, eprint: https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf, URL: https://doi.org/10.1093/bioinformatics/bty191.

[80] James M. Joyce, « Kullback-Leibler Divergence », *in*: *International Encyclopedia of Statistical Science*, ed. by Miodrag Lovric, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722, ISBN: 978-3-642-04898-2, DOI: 10.1007/978-3-642-04898-2_327, URL: https://doi.org/10.1007/978-3-642-04898-2_327.

[81] S. Kullback and R. A. Leibler, « On Information and Sufficiency », *in*: *The Annals of Mathematical Statistics* 22.*1* (1951), pp. 79–86, DOI: 10.1214/aoms/1177729694, URL: https://doi.org/10.1214/aoms/1177729694.

[82] Don Johnson and Sinan Sinanovic, « Symmetrizing the kullback-leibler distance », *in*: *IEEE Transactions on Information Theory* (2001).

[83] Erdal Arikan, « Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels », *in*: *IEEE Transactions on information Theory* 55.*7* (2009), pp. 3051–3073.

[84] M.C. Davey and D.J.C. MacKay, « Low density parity check codes over GF(q) », *in*: *1998 Information Theory Workshop (Cat. No.98EX131)*, 1998, pp. 70–71, DOI: 10.1109/ITW.1998.706440.

[85] C. E. Shannon, « A mathematical theory of communication », *in*: *The Bell System Technical Journal* 27.*3* (1948), pp. 379–423, DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[86] R. W. Hamming, « Error detecting and error correcting codes », *in*: *The Bell System Technical Journal* 29.*2* (1950), pp. 147–160, DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[88] Ray Li, « New developments in coding against insertions and deletions », PhD thesis, Honors thesis, Carnegie Mellon University, Pittsburgh, PA, 2017.

[89] Michael Mitzenmacher, « A survey of results for deletion channels and related synchronization channels », *in*: *Probability Surveys* 6 (2009), pp. 1–33.

[90] R. R. Varshamov and G. M. Tenengol'ts, « A code that corrects single unsymmetric errors », *in*: *Avtomatika Telemekhanika* 26.*2* (1965), pp. 288–292.

[91] Vladimir I Levenshtein et al., « Binary codes capable of correcting deletions, insertions, and reversals », *in*: *Soviet physics doklady*, vol. 10, 8, Soviet Union, 1966, pp. 707–710.

[92] A.S.J. Helberg and H.C. Ferreira, « On multiple insertion/deletion correcting codes », *in*: *IEEE Transactions on Information Theory* 48.*1* (2002), pp. 305–308, DOI: 10.1109/18.971760.

[93] VI Levenshtein, « Asymptotically optimum binary code with correction for losses of one or two adjacent bits », *in*: *Problemy Kibernetiki* 19 (1967), pp. 293–298.

[94] Clayton Schoeny et al., « Codes Correcting a Burst of Deletions or Insertions », *in*: *IEEE Transactions on Information Theory* 63.*4* (2017), pp. 1971–1985, DOI: 10.1109/TIT.2017.2661747.

[95] Shubham Chandak et al., « Improved read/write cost tradeoff in DNA-based data storage using LDPC codes », *in*: *Allerton*, 2019, pp. 147–156, DOI: 10.1109/ALLERTON.2019.8919890.

[96] Andreas Lenz et al., « Coding Over Sets for DNA Storage », *in*: *IEEE Transactions on Information Theory* 66.*4* (2020), pp. 2331–2351, DOI: 10.1109/TIT.2019.2961265.

[97] S. M. Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic, « Portable and Error-Free DNA-Based Data Storage », eng, *in*: *Scientific reports* 7.*1* (2017), pp. 5011–5011, ISSN: 2045-2322.

[98] Sundara Rajan Srinivasavaradhan et al., « Trellis BMA: Coded Trace Reconstruction on IDS Channels for DNA Storage », *in*: *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2453–2458, DOI: 10.1109/ISIT45174.2021.9517821.

[99] Mahed Abroshan et al., « Coding for Deletion Channels with Multiple Traces », *in*: *ISIT*, 2019, pp. 1372–1376, DOI: 10.1109/ISIT.2019.8849647.

[100] Tianbo Xue and Francis C. M. Lau, « Construction of GC-Balanced DNA With Deletion/Insertion/Mutation Error Correction for DNA Storage System », *in*: *IEEE Access* 8 (2020), pp. 140972–140980, DOI: 10.1109/ACCESS.2020.3012688.

[101] Andreas Lenz et al., « Concatenated codes for recovery from multiple reads of DNA sequences », *in*: *2020 IEEE Information Theory Workshop (ITW)*, IEEE, 2021, pp. 1–5.

[102] D. Lavenier, « Constrained Consensus Sequence Algorithm for DNA Archiving », *in*: *arXiv* (2021), URL: https://arxiv.org/abs/2105.04993.

[103] Robert C. Edgar, « MUSCLE: multiple sequence alignment with high accuracy and high throughput », *in*: *Nucleic Acids Research* 32.*5* (Mar. 2004), pp. 1792–1797, ISSN: 0305-1048, DOI: 10.1093/nar/gkh340, URL: https://doi.org/10.1093/nar/gkh340.

[104] C Notredame, D G Higgins, and J Heringa, « T-Coffee: A novel method for fast and accurate multiple sequence alignment », en, *in*: *J Mol Biol* 302.*1* (Sept. 2000), pp. 205–217.

[105] Robert Gallager, « Low-density parity-check codes », *in*: *IRE Transactions on information theory* 8.*1* (1962), pp. 21–28.

[106] Paul H Siegel, « An introduction to low-density parity-check codes », *in*: *Electrical and Computer Engineering University of California, San Diego* (2007).

[107] Bernhard MJ Leiner, « LDPC Codes-a brief Tutorial », *in*: *Stud. ID.: 53418L April* 8.*8* (2005).

[108] Sarah J Johnson, « Introducing low-density parity-check codes », *in*: *University of Newcastle, Australia* 1 (2006), p. 2006.

[109] R Tanner, « A recursive approach to low complexity codes », *in*: *IEEE Transactions on information theory* 27.*5* (1981), pp. 533–547.

[110] James Westall and James Martin, « An introduction to Galois fields and Reed-Solomon coding », *in*: *School of Computing Clemson University Clemson* (2010), pp. 29634–1906.

[111] *Intro to Galois fields*, `https://galois.readthedocs.io/en/v0.0.21/tutorials/intro-to-prime-fields.html`.

[113] David Declercq and Marc Fossorier, « Decoding Algorithms for Nonbinary LDPC Codes Over GF(q) », *in*: *IEEE Transactions on Communications* 55.*4* (2007), pp. 633–643, DOI: 10.1109/TCOMM.2007.894088.

[114] Xiao-Yu Hu, Evangelos Eleftheriou, and Dieter-Michael Arnold, « Regular and irregular progressive edge-growth tanner graphs », *in*: *IEEE transactions on information theory* 51.*1* (2005), pp. 386–398.

[115] M.P.F. dos Santos et al., « Correction of insertions/deletions using standard convolutional codes and the Viterbi decoding algorithm », *in*: *Proceedings 2003 IEEE Information Theory Workshop (Cat. No.03EX674)*, 2003, pp. 187–190, DOI: 10.1109/ITW.2003.1216726.

[116] *Convolutional Coding*, `http://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf`.

[117] Charan Langton, « Coding and decoding with convolutional codes », *in*: *Signal Processing and Simulation Newsletter* (1999), pp. 1–28.

[118] A. Viterbi, « Error bounds for convolutional codes and an asymptotically optimum decoding algorithm », *in*: *IEEE Transactions on Information Theory* 13.*2* (1967), pp. 260–269, DOI: 10.1109/TIT.1967.1054010.

[119] Lalit Bahl et al., « Optimal decoding of linear codes for minimizing symbol error rate (corresp.) », *in*: *IEEE Transactions on information theory* 20.*2* (1974), pp. 284–287.

[120] B.M. Kurkoski, P.H. Siegel, and J.K. Wolf, « Analysis of convolutional codes on the erasure channel », *in*: *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.* 2004, pp. 460–, DOI: 10.1109/ISIT.2004.1365495.

[121] Victor Buttigieg and Noel Farrugia, « Improved bit error rate performance of convolutional codes with synchronization errors », *in*: *2015 IEEE International Conference on Communications (ICC)*, IEEE, 2015, pp. 4077–4082.

[122] Mohamed F Mansour and Ahmed H Tewfik, « Convolutional decoding in the presence of synchronization errors », *in*: *IEEE Journal on Selected Areas in Communications* 28.*2* (2010), pp. 218–227.

[123] Matthew C Davey and David JC MacKay, « Reliable communication over channels with insertions, deletions, and substitutions », *in*: *IEEE Transactions on Information Theory* 47.*2* (2001), pp. 687–698.

[124] L. Bahl and F. Jelinek, « Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition », *in*: *IEEE Transactions on Information Theory* 21.*4* (1975), pp. 404–411, DOI: 10.1109/TIT.1975.1055419.

[125] Laura Conde-Canencia, Tyson Condie, and Lara Dolecek, « Data Deduplication with Edit Errors », *in*: *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6, DOI: 10.1109/GLOCOM.2018.8647415.

[126] Qinlu He, Zhanhuai Li, and Xiao Zhang, « Data deduplication techniques », *in*: *2010 International Conference on Future Information Technology and Management Engineering*, vol. 1, 2010, pp. 430–433, DOI: 10.1109/FITME.2010.5656539.

[127] Ahmed El-Shimi et al., « Primary Data Deduplication—Large Scale Study and System Design », *in*: *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, Boston, MA: USENIX Association, June 2012, pp. 285–296, ISBN: 978-931971-93-5, URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/el-shimi.

[128] J. Ziv and A. Lempel, « A universal algorithm for sequential data compression », *in*: *IEEE Transactions on Information Theory* 23.*3* (May 1977), pp. 337–343.

[129] J. Cleary and I. Witten, « Data Compression Using Adaptive Coding and Partial String Matching », *in*: *IEEE Transactions on Communications* 32.*4* (Apr. 1984), pp. 396–402, ISSN: 0090-6778, DOI: 10.1109/TCOM.1984.1096090.

[130] A. Venish and K. Siva Sankar, « Study of Chunking Algorithm in Data Deduplication », *in*: *Proc. of Int. Conf. on Soft Computing Systems, Advances in Intelligent Systems and Computing, Springer India* (2016).

[131] João Paulo and José Pereira, « A Survey and Classification of Storage Deduplication Systems », *in*: *ACM Comput. Surv.* 47.*1* (June 2014), ISSN: 0360-0300, DOI: 10.1145/2611778, URL: https://doi.org/10.1145/2611778.

[132] Frederic Sala et al., « Synchronizing files from a large number of insertions and deletions », *in*: *IEEE Transactions on Communications* 64.*6* (2016), pp. 2258–2273.

[133] Urs Niesen, « An Information-Theoretic Analysis of Deduplication », *in*: *IEEE Transactions on Information Theory* 65.*9* (2019), pp. 5688–5704, DOI: 10.1109/TIT.2019.2916037.

[134] Hao Lou and Farzad Farnoud, « Data Deduplication with Random Substitutions », *in*: *IEEE Transactions on Information Theory* (2022), pp. 1–1, DOI: 10.1109/TIT.2022.3176778.

[135] Laura Conde-Canencia and Belaid Hamoum, « Deduplication algorithms and models for efficient data storage », *in*: *2020 24th International Conference on Circuits, Systems, Communications and Computers (CSCC)*, 2020, pp. 23–28, DOI: 10.1109/CSCC49995.2020.00013.

[136] Ruby White et al., « Investigation of chimeric reads using the MinION », en, *in*: *F1000Res* 6 (May 2017), p. 631.

[137] Leandro Lima et al., « Comparative assessment of long-read error correction software applied to Nanopore RNA-sequencing data », en, *in*: *Brief Bioinform* 21.*4* (July 2020), pp. 1164–1181.

[138] Siying Ma, Nicholas Tang, and Jingdong Tian, « DNA synthesis, assembly and applications in synthetic biology », en, *in*: *Curr. Opin. Chem. Biol.* 16.*3-4* (Aug. 2012), pp. 260–267.

[139] Winston Timp et al., « Nanopore sequencing: Electrical measurements of the code of life », *in*: *IEEE Trans. Nanotechnol.* 9.*3* (May 2010), pp. 281–294.

[140] J.F. Hess et al., « Library preparation for next generation sequencing: A review of automation strategies », *in*: *Biotechnology Advances* 41 (2020), p. 107537, ISSN: 0734-9750, DOI: https://doi.org/10.1016/j.biotechadv.2020.107537, URL: https://www.sciencedirect.com/science/article/pii/S0734975020300343.

[141] Eric W Sayers et al., « GenBank », en, *in*: *Nucleic Acids Res.* (Oct. 2019).

[142] H. Mercier, V. K. Bhargava, and V. Tarokh, « A survey of error-correcting codes for channels with symbol synchronization errors », *in*: *IEEE Communications Surveys & Tutorials* 12.*1* (2010), pp. 87–96.

[143] Clayton Schoeny et al., « Codes Correcting a Burst of Deletions or Insertions », *in*: *IEEE Transactions on Information Theory* 63.*4* (2017), pp. 1971–1985, DOI: 10.1109/TIT.2017.2661747.

[144] Amin Shokrollahi, « LDPC codes: An introduction », *in*: *Coding, cryptography and combinatorics*, Springer, 2004, pp. 85–110.

[145] Elsa Dupraz, « Codage de sources avec information adjacente et connaissance incertaine des corrélations », Theses, Université Paris Sud - Paris XI, Dec. 2013, URL: https://tel.archives-ouvertes.fr/tel-01136663.

[146] Melpomeni Dimopoulou, « Encoding techniques for long-term storage of digital images into synthetic DNA », Theses, Université Côte d'Azur, Dec. 2020, URL: https://tel.archives-ouvertes.fr/tel-03185468.

[147] Laura Conde-Canencia et al., « Error Correction Schemes for DNA Storage with Nanopore Sequencing », *in*: *Les Journées Ouvertes en Biologie, Informatique et Mathématiques (JOBIM)*, 2019, URL: https://jobim2019.sciencesconf.org/resource/page/id/11.

[148] nanoporetech, *Microbiology with Oxford Nanopore Technologies QA*, https://nanoporetech.com/events/Microbiology-with-Oxford-Nanopore-Technologies-qa.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université Bretagne Sud

Titre : Algorithmes pour la synchronisation de données et leur stockage sur ADN

Mot clés : Stockage de données sur ADN, modèle de canal, NB-LDPC, Codes convolutifs, erreurs d'insertion et de suppression, déduplication de données

Résumé : 175 zettaoctets. C'est la capacité estimée pour pouvoir stocker les données numériques en 2025. Malgré le fait que des centres de données plus grands que des stades et à forte empreinte carbone sont déployés chaque année, la croissance de la capacité de stockage est inférieure aux besoins. Le Stockage de Données sur ADN (SDA) pourrait être la solution. En effet, l'ADN est un support extrêmement dense de stockage de données. De plus, il a une très longue durée de vie et peut être stocké à température ambiante. Cependant, le principal inconvénient du SDA est sa grande quantité d'erreurs d'insertions, suppressions, et substitutions. Par conséquent, pour construire des SDA pratiques et fiables, il est nécessaire de mettre en œuvre des solutions de correction d'erreurs. Cependant, la plupart des solutions de correction d'erreurs conventionnelles ne corrigent que les erreurs de substitution et échouent complètement à corriger les insertions et les suppressions. Cette thèse vise à résoudre plusieurs problèmes liés à la mise en œuvre de systèmes pratiques de SDA. Nous avons d'abord proposé un modèle de canal avec mémoire, qui modélise avec précision le canal de SDA. Ce modèle de canal permet notamment de faire des simulations numériques et de concevoir des codes correcteurs d'erreurs efficaces. Nous avons ensuite proposé et évalué deux solutions de correction d'erreurs. La deuxième solution basée sur des codes convolutifs a notamment permis un gain de performance important par rapport à la première solution et aux codes convolutifs de l'état de l'art. Enfin, nous avons également proposé un algorithme de déduplication de données appelé PBDA-SW, qui améliore l'état de l'art.

Title: DNA data storage algorithms and synchronization

Keywords: DNA data storage, channel model, NB-LDPC, Convolutional Codes, insertion and deletion errors, data deduplication

Abstract: 175 zettabytes. This is the predicted digital data storage needs for 2025. Despite the fact that data centers larger than stadiums and with a high carbon footprint are deployed every year, data storage capacity growth is less than required. DNA data storage could be the solution. Indeed, DNA is an extremely dense data storage media. In addition, it has a very long durability, and can be stored at a room temperature. However, the main drawback of DNA data storage is its high amount of insertion, deletion, and substitution errors. Hence, to build reliable practical DNA data storage systems, it is necessary to implement error-correction solutions. However, most conventional error-correction solutions only correct substitution errors, and completely fail at correcting insertions and deletions. This thesis aims to address several issues toward implementing practical DNA data storage systems. We first propose a memory channel model, which accurately models the DNA data storage channel. Especially, this channel model allows to run numerical simulations and to design efficient error-correction codes. We then introduce and evaluate two error-correction solutions. Especially, the second solution based on convolutional codes allows for an important gain in performance compared to the first solution and to state-of-the-art convolutional codes. Finally, we also propose a data deduplication algorithm called PBDA-SW, which improves state-of-the-art on data deduplication.