HIMME, A NEXT-GENERATION SEQUENCING QUALITY ASSESSMENT AND CORRECTION TOOL BASED ON HIDDEN MARKOV MODELS

A Thesis

by

JORDI ABANTE LLENAS

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Aniruddha Datta
Committee Members,	Shankar P. Bhattacharyya
	Ulisses Braga-Neto
	Alan Dabney
	Noushin Ghaffari
Head of Department,	Miroslav M. Begovic

May 2016

Major Subject: Electrical Engineering

Copyright 2016 Jordi Abante Llenas

ABSTRACT

Both deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) play a crucial role in the existence and proper development of all living organisms. In addition, it is through these molecules that the genetic information is passed from parent to offspring. It is no surprise that, over the last decades, a lot of efforts have been put into developing technology that help us better understand their underlying mechanisms.

Similarly to how computers work using only ones and zeros, DNA and RNA only need four different characters to encrypt all the genetic information. Thanks to the sequencing technology development over the past decades, it is possible nowadays to sequence these molecules in a relatively fast and inexpensive way. However, as in any measurement, there is noise involved and this needs to be addressed if one is to reach conclusions based on these kind of data.

The hidden Markov model (HMM) is a perfect fit for this case. Through a Markov chain, the model can capture genetic patterns, while, by introducing the emission probabilities, the noise involved in the process can be taken into account. In addition, previous knowledge can be used by training the model to fit, for instance, a given organism or sequencing technology.

In this thesis, the HMM theory is applied for two purposes, (1) to assess the reliability of sequencing data, and (2) to correct potential errors in the sequences observed. The results show that the HMM model is capable of identifying genetic patterns in the sequence and to repair potential errors, thus improving the reliability of the data before any downstream analysis is performed. For these purposes, HiMMe has been developed and is publicly available on https://github.com/jordiabante/HiMMe.

DEDICATION

Before getting into matter, I would like to acknowledge those who have supported me during my tenure as a Master of Science student at Texas A&M University.

Firstly, I should thank Dr. Enjeti for convincing me to come to Aggieland as well as for all his support over the first weeks I spent in College Station.

Secondly, I should thank everyone in the Center for Bioinformatics and Genomic Signal Engineering (CBGSE) for their kindness and help. Specially, I would like to thank Dr. Johnson and Dr. Ghaffari for believing in me and giving me the opportunity of carrying out my research at the center while being involved in really exciting projects.

Thirdly, I want to thank Dr. Datta, Dr. Braga-Neto and Dr. Dabney for their support in the "la Caixa" fellowship application as well as my PhD applications. I believe that all that support has been instrumental for me to succeed and be granted with the fellowship and with the opportunity of going to one of the best schools in the world for my PhD.

I also want to thank my friends both from College Station and Barcelona, who have supported me unconditionally and helped me go through these two years of graduate school. Thanks Jordan and Crystal; moltes gràcies Lluís i Marc.

And last but not least, I want to thank my family for being there whenever I needed them. Although we might be 5,250 miles away, none of this would have been possible without their unconditional support. Moltes gràcies per ser com sou i per recolzar-me una vegada més en una de les meves aventures.

NOMENCLATURE

A	Adenine
bp	Base pair
C	Cytosine
cDNA	Complementary deoxyribonucleic acid
DNA	Deoxyribonucleic acid
G	Guanine
HMM	Hidden Markov model
LTP	Law of Total Probability
MC	Markov chain
MNase-seq	Micrococcal nuclease sequencing
mRNA	Messenger ribonucleic acid
NGS	Next-generation sequencing
NCBI	National Center for Biotechnology Information
PCR	Polymerase chain reaction
SEQC	Sequencing Quality Control
SNP	Single nucleotide polymorphism
RAM	Random-access memory
RNA	Ribonucleic acid
rRNA	Ribosomal ribonucleic acid
T	Thymine
U	Uracil
VCF	Variant Call Format

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

Back in 1977, gel electrophoresis was first used to separate deoxyribonucleic acid (DNA) fragments at single-base resolution. Over the next decade, electrophoresis sequencing became very popular and different groups adopted and improved it [13]. Some ambitious group of scientists believed that by 2005 the entire human genome could be sequenced. Nevertheless, a lot of skepticism arose given the high cost of the state of the art technology. At that time, the cost associated with sequencing a single base was approximately $10.

However, a lot of effort has been put into the improvement of sequencing technology. Consequently, the cost has been drastically reduced since then. This cost reduction has been so prominent that it has even surpassed Moore's Law's trend.

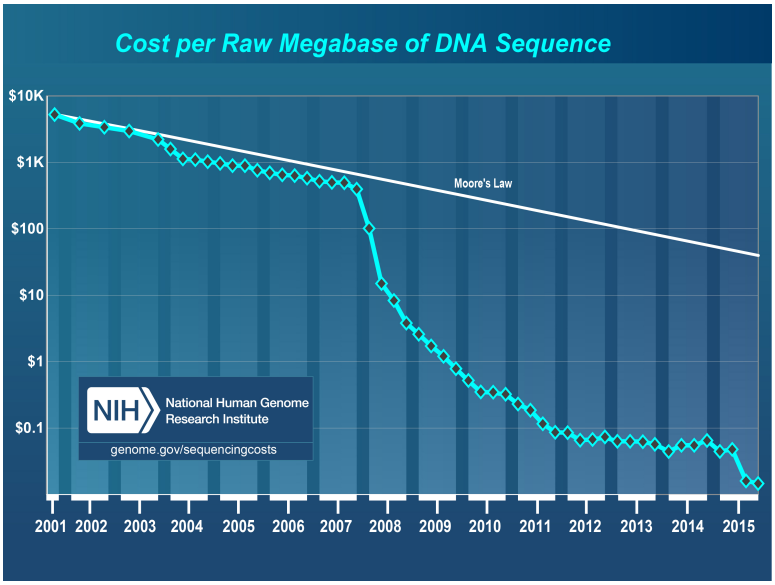


Figure 1.1: Cost per megabase of DNA sequence - the cost of determining one megabase (million bases) of DNA sequence of a specified quality [16].

Figure 1.1 illustrates the cost reduction of DNA Sequencing over the past fifteen years. Note the sharp decrease around 2008, when the sequencing centers began the transition from Sanger-based technology [10] to next-generation sequencing. This sharp improvement of the technology marked the sudden and profound out-pacing of Moore's Law [16].

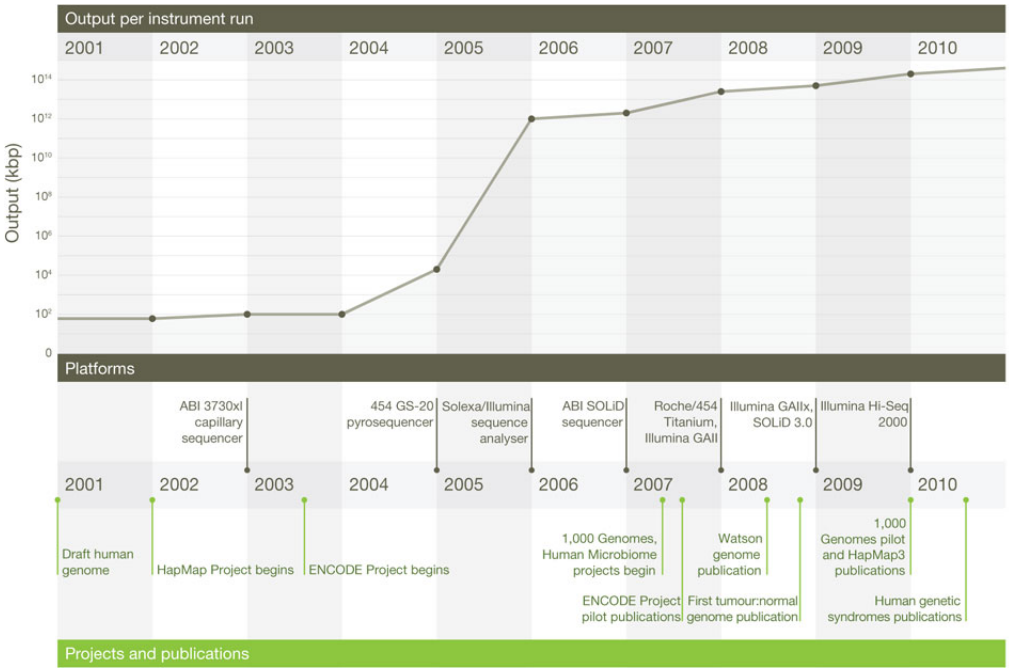


Figure 1.2: DNA-sequencing significant increase in the output per instrument run plotted on a logarithmic scale along some of the major breakthroughs in sequencing technology [7].

Moreover, not only has the cost associated with sequencing been significantly reduced, but the speed with which one can sequence biological samples has dramatically increased as well. Note in Figure 1.2 how there is a sudden increase in the output density per run when next-generation sequencing technology took over Sanger sequencing, reflecting again the huge impact that this sequencing technology had in

the field ten years ago. And, in addition to an increase in the output density per run, there has been a fierce competition between sequencing technology manufacturers that has led to a progressive increase in the read length as well as the base-calling accuracy [7].

However, such amount of information needs a place to be stored. Furthermore, extremely powerful hardware and software are required as well for it to be processed. Following Moore's Law in this case, computer technology has enabled scientists to deal with an important portion of these data in order to distill part of the biological insight in it. As a consequence, biological and medical research has lead to numerous discoveries during these last decades.

1.1 Central dogma of molecular biology

It is well known that all the living organisms store their genetic information in the DNA [15]. This huge double stranded molecule can be regarded as a sequence of nucleotides, which can be of four different kinds. These four types of nucleotides are: Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). Given DNA's double-stranded nature, these four bases are found in pairs in the DNA (Adenine with Thymine, Guanine with Cytosine) [14]. Just using these four different letters, living organisms are capable of encrypting all the genetic instructions for the proper functioning of their cells. However, it is not the DNA itself that carries out all the necessary reactions in the cell, there are some other molecules that play an important role in this process as well. These two molecules are (1) ribonucleic acid (RNA) and (2) proteins.

Through a process called *transcription*, the DNA is transcribed into RNA which will eventually leave the nucleus of the cell after undergoing certain intermediate processes in it. The RNA is similar to the DNA in that it is a molecule that can

be regarded as a sequence of nucleotides. However, there are two main differences between the RNA and the DNA. Firstly, the RNA is not a double-stranded molecule, which in turn makes it unstable compared to the DNA. Secondly, the four bases that the RNA uses to transfer the genetic information differ in one when compared to the DNA. Instead of Thymine, RNA relies on another base called Uracil (U).

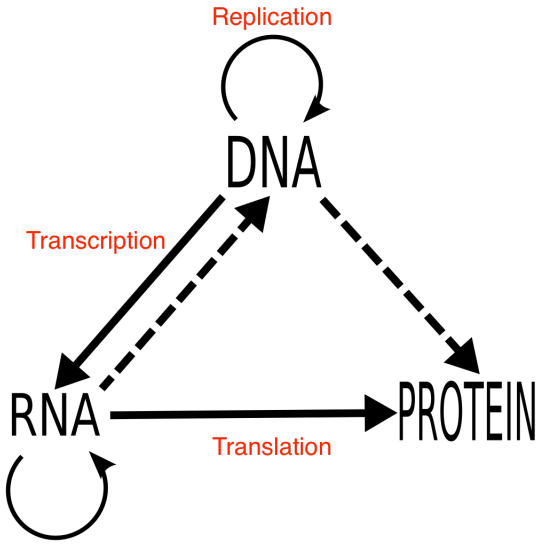


Figure 1.3: Central dogma of molecular biology.

RNA molecules, once they leave the nucleus of the cell and reach the ribosome, undergo a process called *translation*. Through this process, RNA molecules provide the necessary information to the ribosome to generate any needed protein. This whole process is usually referred to as the *central dogma of molecular biology*, and is illustrated in Figure 1.3. This dogma explains how the genetic information flows in living organisms. As Crick [2] stated:

> The central dogma of molecular biology deals with the detailed residue-by-residue transfer of sequential information. It states that such information

cannot be transferred back from protein to either protein or nucleic acid.

However, not all the nucleotides in the DNA are transcribed. The parts of the genome that undergo the mentioned process are commonly referred to as genes. These are made of two different kinds of sequences, mainly *exons* and *introns*. The former usually reach the ribosome to be *translated* into proteins. The latter, instead, are removed in an intermediate process called *splicing*.

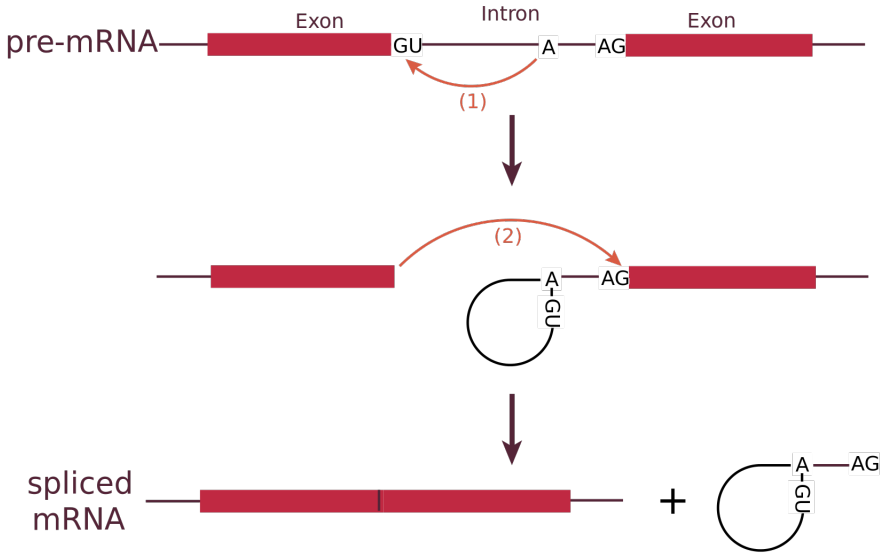


Figure 1.4: RNA splicing process example.

Through this process, the cell selects the *exons* that are required and gets rid of the introns in the pre-mature RNA to produce mRNA. Figure 1.4 illustrates this process in a simplified fashion. In this case, the gene contains two exons and an intermediate intron. The figure depicts how, through splicing, the cell gets rid of the

intron to produce the mRNA. Note that this process can lead to the production of different proteins coming from the same gene.

1.2 Next-generation sequencing (NGS) and some applications

As of 2016, the state of the art sequencing technology is usually referred to as next-generation sequencing (NGS).

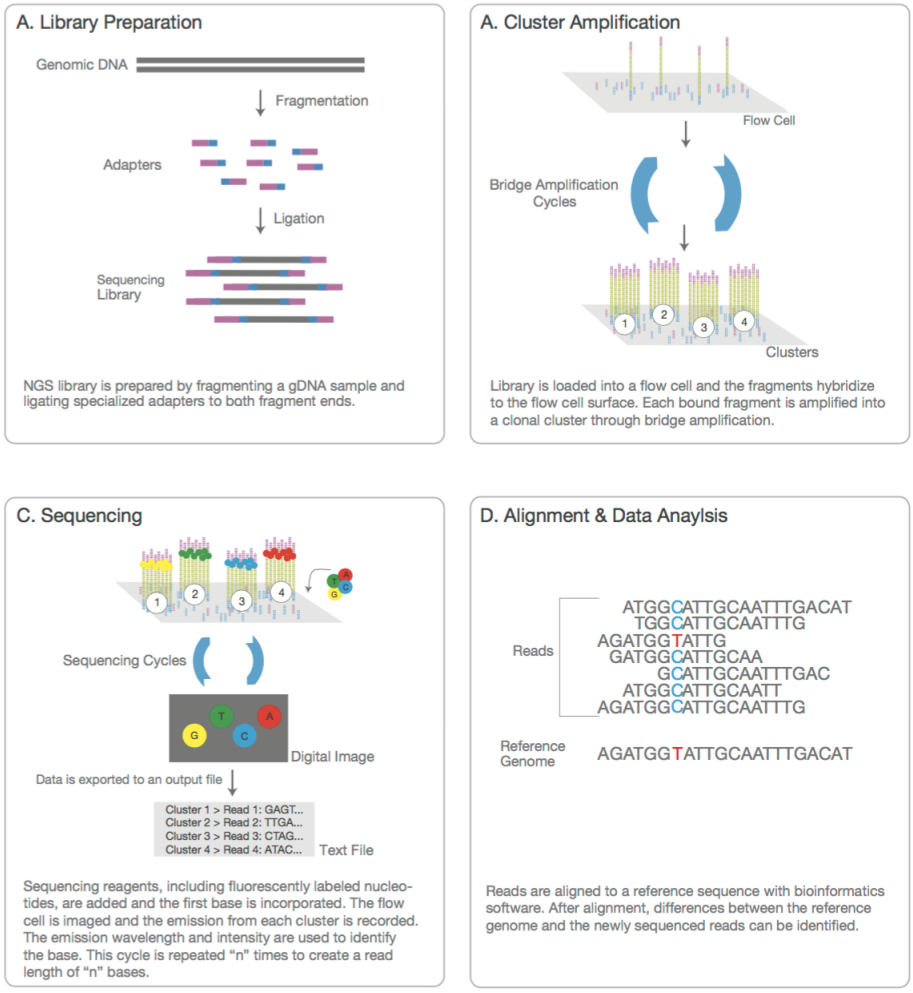


Figure 1.5: Next-generation sequencing chemistry overview [4].

The main workflow of the current dominant NGS platform, Illumina, is depicted in Figure 1.5 and its steps are described as follows:

1. Library Preparation: the DNA or the cDNA is randomly fragmented and adapters are attached to both the 5' and the 3' end of the strands. The product is then amplified by a polymerase chain reaction (PCR) and purified.

2. Cluster Generation: the products of the library preparation step bind to some oligos, which are complementary to the library adapters used in the previous step. Through bridge amplification, each fragment is amplified into clonal clusters.

3. Sequencing: fluorescent-labeled nucleotides are used to sequentially determine the nature of each nucleotide. The flow cell is imaged at each iteration, and the emission wavelength and intensity is later on used to identify the base. This process is repeated as many times as bases there are in the reads.

4. Data Analysis: once the reads are obtained, one can align them to the corresponding reference genome. Many different analysis are possible at this point such as read counting for RNA methods or studying single nucleotide polymorphisms (SNP) among others.

NGS data have been found to have several applications. For instance, through RNA-seq, one can study which genes are associated with a given condition by comparing the gene expression levels between the control and the condition subjects. To do so, RNA samples from multiple replicates are sequenced for both conditions. Once the reads are obtained, usually in FASTQ format, reads are aligned to the reference genome of the organism being studied. The number of reads that fall in each annotated region of interest, are compared between conditions. By doing so, gene

expression and *allele* frequency differences can be found between conditions in case such differences existed. This type of analysis is usually referred to as *differential gene expression analysis*. While it was already possible with the so-called microarray technology ten years ago, by taking advantage of NGS, one can speed up the process significantly given the amount of genes that can be studied in a single run [8].

Another crucial topic in the computational biology field that relies on NGS data is the study of epigenetic structures. Scientists have come to believe that the DNA does not have the last say in how living organisms develop. Different external factors to the DNA, mainly proteins, play an important role in this process by binding to the promoter region and modifying the expression level of the corresponding gene in turn [11]. Other examples of epigenetic processes discovered so far are changes in the chromatin structure [6] or DNA methylation [5]. For instance, one can study the position of nucleosomes and the effect that these have in the expression level of a given gene through micrococcal nuclease sequencing (MNase-seq) data analysis. Nucleosomes are protein complexes that the DNA uses to compact its structure in a rather efficient way. DNA wraps around nucleosomes and these are organized in a very compacted fashion to form the chromatin structure. Using the micrococcal nuclease, researchers get rid of the parts of the DNA that are not binded to any nucleosome and only those sequences wrapped around nucleosomes are sequenced. By mapping these sequences back to the reference genome, one can study the position of these protein complexes and look for potential correlations with gene expression levels.

From the moment that biologists extract DNA or RNA samples to the point in which one analyzes the data, there are multiple intermediate steps that can affect the final result. For instance, due to the inherent random sampling of the sequencing technology, RNA-seq has been found to have measurement noise [12]. This has

led to different outcomes in previous *differential gene expression analysis* studies. Therefore, it is of high relevance to develop tools that allow researchers to make sure that the reads that they use for their downstream analysis are as error-free and reliable as possible. This makes the hidden Markov model (HMM) approach very useful, since it allows one to deal with observing noisy signals, i.e. the sequences obtained, which are used as hints to infer the real signal, i.e. the real sequences. This model will be formally introduced in section 2 and applied to genomic sequences in section 3.

1.3 Thesis layout

The previous sections have underlined the relevance of the recent sequencing technology breakthroughs and some applications have been described. The relevance of sequencing data reliability to biological and medical research has also been highlighted. In section 2, the theoretical foundations of hidden Markov models (HMM) are introduced as well as the notation used throughout the thesis. In section 3, this theory is applied to genomic sequences by modeling the observed data as hints that can help one to find the real sequences and to evaluate the reliability of the observed sequences. Once the theory is adapted to NGS data, section 4 describes the way the model is implemented in a computational efficient fashion. Following, in section 5 the two main results of this thesis are presented (1) scoring sequences reliability and (2) repairing potential errors. Finally, in section 6, conclusions are provided based on the results obtained.

2. HIDDEN MARKOV MODELS (HMM)

Hidden Markov models (HMM) are used to describe the behavior of a system based on hints, usually referred to as observable events. For instance, one could try to infer the position of the upstairs neighbor based on the sound of his or her steps. The observable events, the sounds produced by the steps in this case, are usually called 'symbols'. On the other hand, the underlying or invisible factor one is trying to understand, in this case the position of the neighbor, is usually referred to as 'state'.

More technically, a HMM can be regarded as the interaction between two stochastic processes. That is why it is usually referred to as a *doubly-embedded stochastic process* [9]. The probability distribution of the symbols depend on the underlying states, while the latter form a *Markov chain*. Thus, given that the present state is known, the future states are conditionally independent of the past.

2.1 Markov chain

Let us formally define an HMM, and for that purpose, let us first define the *Markov chain* that the underlying states follow. The state space of the *Markov chain* is denoted as E. We denote a general sequence of states as follows:

$$X = X_1 \cdots X_m \tag{2.1}$$

where m is the number of states in the sequence. The stochastic process $X = \{X_n; n \in \mathbb{N}\}$ is called a *Markov chain* provided that

$$P\{X_{n+1} = j | X_0, ..., X_n\} = P\{X_{n+1} = j | X_n\} \tag{2.2}$$

for all $j \in E$ and $n \in \mathbb{N}$ [1]. This conditional probability can be expressed also using the following notation:

$$P\{X_{n+1} = j | X_n = i\} = P(i, j) \tag{2.3}$$

The probabilities $P(i, j)$ are called *transition probabilities* and can be arranged in a matrix P called *transition matrix* or *Markov matrix*. Thus, if $E = \{0, 1, 2, ...\}$ this matrix has the following form:

$$P = \begin{bmatrix} P(0,0) & P(0,1) & P(0,2) & \cdots \\ P(1,0) & P(1,1) & P(1,2) & \cdots \\ P(2,0) & P(2,1) & P(2,2) & \cdots \\ \cdot & \cdot & \cdot & \cdots \\ \cdot & \cdot & \cdot & \cdots \end{bmatrix} \tag{2.4}$$

For any square matrix to be a *transition matrix* defined over E the following conditions have to be met:

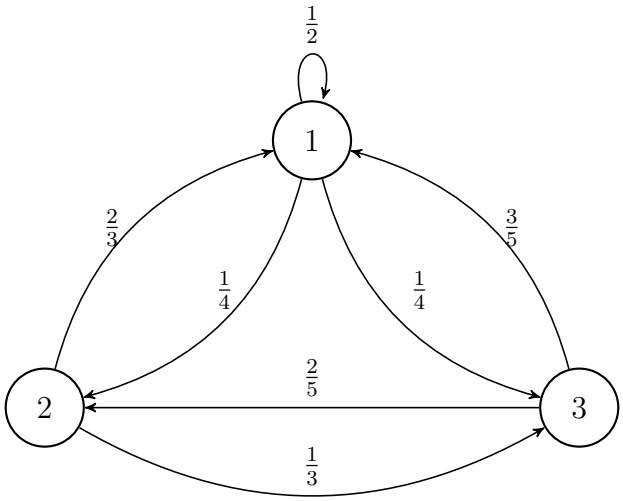
1. $P(i, j) \geq 0 \ \forall i, j \in E$, and
2. $\sum_{j \in E} P(i, j) = 1, \ \forall i \in E$

Therefore, each row of the *transition matrix* has to add up to one and all the entries have to be non-negative. In addition, when this matrix is symmetric, the associated *Markov chain* has some special properties that simplify the problem. Such a discrete stochastic process is usually referred to as a *double chain Markov model*. However, this will not be the case for the intended application of the model as described in this thesis. For instance, the following could be considered a *transition*

matrix for state space $E = \{1, 2, 3\}$:

$$P = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{2}{3} & 0 & \frac{1}{3} \\ \frac{3}{5} & \frac{2}{5} & 0 \end{bmatrix} \tag{2.5}$$

Such a transition matrix represents mathematically the following *Markov chain*:



Note that all the edges leaving each node in the graph add up to one, since each row in the corresponding *transition matrix* fulfills that condition as well. From the graph it is usually easier to understand the dynamics of the *Markov chain*. Consider now the following sequence of states:

$$X_1 = 1, X_2 = 1, X_3 = 3, X_1 = 2 \tag{2.6}$$

For instance, considering that $X_0 = 1$, the probability of such a path can be computed as follows:

$$P\{X_1 = 1, X_2 = 1, X_3 = 3, X_4 = 2 | X_0 = 1\} = P(1,1)P(1,1)P(1,3)P(3,2)$$

$$= \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{2}{5}$$
$$= \frac{1}{40} \tag{2.7}$$

It is of interest as well the following characteristic of a *Markov chain*. The probability that the chain moves from state i to state j in r steps is the (i, j) entry of the rth power of the *transition matrix* [1].

$$P\{X_{n+r} = j | X_n = i\} = P^r(i, j) \tag{2.8}$$

Thus, one can compute the probability of going from state i to state j in r steps, taking into account all the possible states in between, by using the preceding formula.

2.2 Emission probabilities

Once introduced the *Markov chain* that the underlying states follow, let us define the emission probabilities, which bridge the gap between the observed symbols and the underlying sequence of states. Each symbol Y_i is a random variable that takes on a set of possible observations $O = \{O_1, ..., O_L\}$ based on a probability distribution conditional on the current underlying state. We denote a sequence of symbols as follows:

$$Y = Y_1 \cdots Y_m \tag{2.9}$$

Then, since the random variable Y_i takes on O based on the current state only, we have:

$$P\{Y_n = y | Y_1, ..., Y_{n-1}, X_1, ..., X_{n-1}, X_n = i\} = P\{Y_n = y | X_n = i\} \tag{2.10}$$

for all $y \in O$ and all $i \in E$. This conditional probability is referred as *emission probability* of y at state i and represented by $e(\cdot|\cdot)$.

$$P\{Y_n = y|X_n = i\} = e(y|i) \tag{2.11}$$

2.3 Complete model

The initial state probability of the *Markov chain* X can be denoted as:

$$\pi_0(i) = P\{X_1 = i\} \tag{2.12}$$

Note that the HMM is completely specified by the three probability measures $\pi_0(i)$, $P(i,j)$ and $e(y|i)$. This set of probabilities is going to be denoted as Θ for convenience from now on. For a realization Y and X we have:

$$\begin{aligned} P\{Y, X|\Theta\} &= \frac{P\{Y, X, \Theta\}}{P\{\Theta\}} \\ &= \frac{P\{Y|X, \Theta\}P\{X, \Theta\}\}}{P\{\Theta\}} \\ &= \frac{P\{Y|X, \Theta\}P\{X|\Theta\}P\{\Theta\}}{P\{\Theta\}} \\ &= P\{Y|X, \Theta\}P\{X|\Theta\} \end{aligned} \tag{2.13}$$

where

$$P\{Y|X, \Theta\} = e(Y_1|X_1)e(Y_2|X_2)\cdots e(Y_m|X_m) \tag{2.14}$$

and

$$P\{X|\Theta\} = \pi_0(X_1)P(X_1, X_2)\cdots P(X_{m-1}, X_m) \tag{2.15}$$

Thus, when the underlying state sequence $X = X_1 \cdots X_m$ is known, it is easy to compute the observation probability of a given realization $Y = Y_1 \cdots Y_m$. Note as well that by the Law of Total Probability (LTP):

$$P\{Y|\Theta\} = \sum_{x \in \Omega_m} P\{Y, X = x|\Theta\} \tag{2.16}$$

where Ω_m is the set of all possible sequences of states of length m. Note that one could potentially go through every single possible hidden state sequence to find the marginal distribution of the observation based on the model used. In other words, without knowing the actual hidden state sequence, one can compute the probability of observing Y given the model, i.e. Θ. Depending on how large the state space is and the number of elements in the chain, this might not even be feasible. However, by taking advantage of the Bayes' rule, equation 2.16 can be expanded in the following way:

$$\begin{aligned} \sum_{x \in \Omega_m} P\{Y, X = x|\Theta\} &= \sum_{x \in \Omega_m} P\{Y|X = x, \Theta\} P\{X = x|\Theta\} \\ &= \sum_{x \in \Omega_m} \pi_0(x_1) e(Y_1|x_1) \prod_{i=2}^{m} e(Y_i|x_i) P(x_{i-1}, x_i) \end{aligned} \tag{2.17}$$

Note that if the distributions $\{Y|X, \Theta\}$ and $\{X|\Theta\}$ are known, i.e. Θ is known, then some sort of sampling technique could be used in order to find the marginal distribution of the symbol sequence independent on the hidden states sequence. For instance, a hierarchical modeling approach could be used to sample first X^* from $P\{X|\Theta\}$ and then sample Y^* from $P\{Y|X^*, \Theta\}$ and iterate as many times as needed. By doing so, one could eventually find the empirical marginal distribution of the observation Y given the model Θ.

3. SEQUENCING DATA MODELING

In section 1, NGS data and multiple applications of this sequencing technology have been introduced. In section 2, the theory behind a hidden Markov model has been introduced. The aim of this section is to apply this theory to assess the reliability of this type of data and to correct potential errors in it as well.

As introduced in the previous section, an HMM is a *doubly-embedded stochastic process*. Such a process is composed of an observed sequence of symbols and a sequence of states which, on the other hand, is not observed. Here, each sequence obtained through NGS is going to be regarded as a sequence of symbols. These observed sequences are going to be scored without knowing the corresponding underlying sequence of states. Then, an error-free version, i.e. the most likely underlying state sequence, is going to be provided for each of the observed sequences.

3.1 Amount of information in a k-mer

A sequence of k nucleotides is usually referred to as a *k-mer*. When breaking down a DNA or RNA sequence into substrings, one can use different *k-mer* sizes. The number of possible states in the *Markov chain* that defines the behavior of the underlying state sequence, clearly depends on the choice of *k-mer* size, since there would be as many as $|N|^k$ different states, where $|N|$ is the cardinality of the set of characters used at each position in the *k-mer* and k is the *k-mer* size. Therefore, following the notation introduced in section 2, the cardinality of the set of states for the hidden layer would be:

$$|E| = |N|^k \tag{3.1}$$

For instance, if the set of characters used at each position in the *k-mer* contained the four possible nucleotides that exist in DNA, i.e. $N = \{A, C, G, T\}$, then the cardinality of the space state of the *Markov chain* would be 4^k.

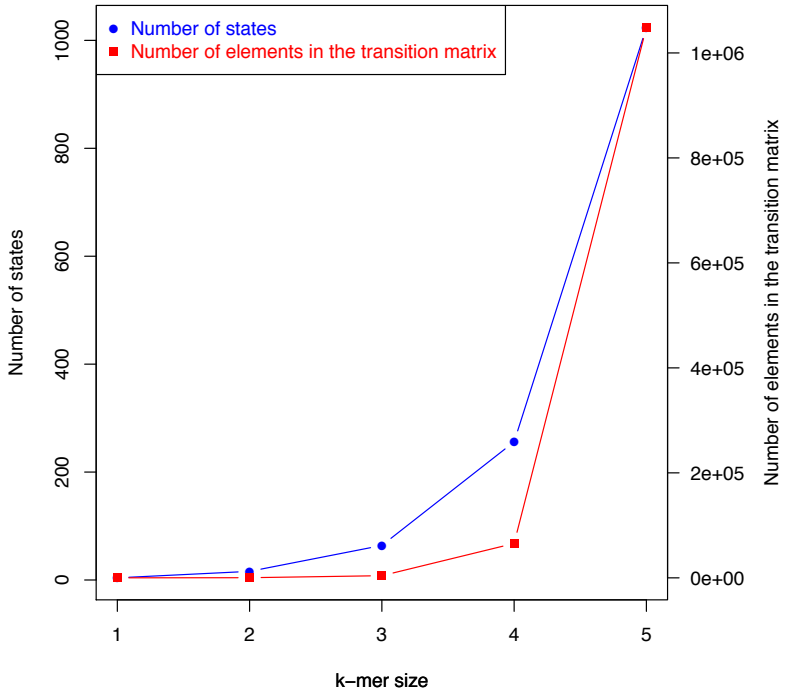


Figure 3.1: Number of states and size of the transition matrix as a function of the *k-mer* size.

Note that the size of the *transition matrix*, which defines the underlying sequences of states, would grow quadratically with the number of states as Figure 3.1 suggests. Therefore, the memory requirements also grow quadratically with the size of *k-mer* used. However, one can benefit from using larger *k-mers*. For instance, one could break the sequence down into individual nucleotides, thus using *1-mers*, or one could break it down into groups of three nucleotides or *3-mers*. When choosing the *k-mer*

size, one has to be aware that, as it becomes smaller, information is lost. In the previous example, the second choice of *k-mer* size would have more memory and computational requirements, but intuitively, the model would be able to detect more patterns in the data.

Looking at it from an Information Theory standpoint, the amount of information obtained from observing an event A with probability p_A is:

$$I(A) = -\log(p_A) \tag{3.2}$$

Therefore, as the cardinality of the state space grows, more information can be obtained from an observation on average.

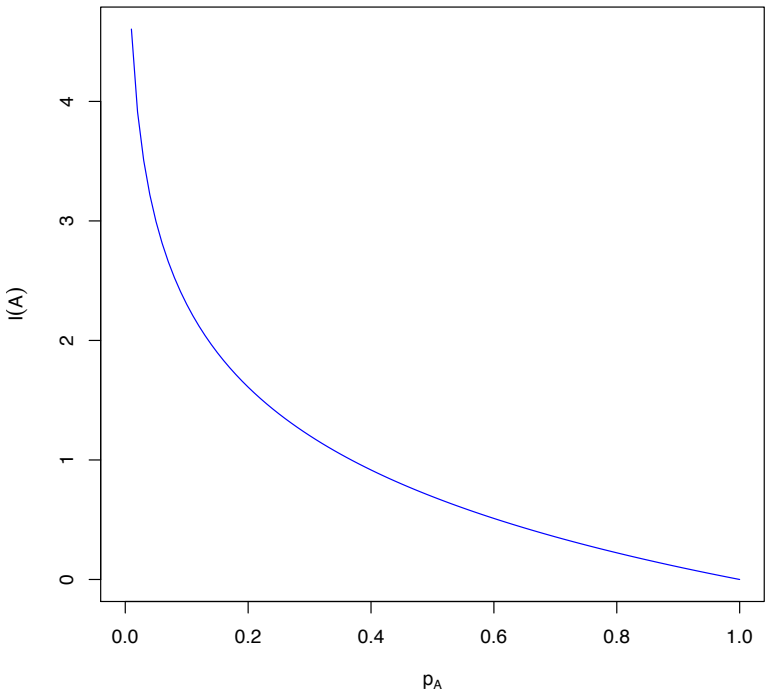


Figure 3.2: Amount of information given the probability of event A.

Note in Figure 3.2 how the amount of information obtained when observing an event A increases as the probability of this event decreases. For instance, consider the *1-mer* case, where each of the outcomes has the same probability of 1/4 of occurring. The amount of information obtained when observing a given *1-mer* is:

$$I(\textit{1-mer}) = -log\left(\frac{1}{4}\right) \approx 0.6 \tag{3.3}$$

On the other hand, consider the case where *3-mers* are used. In this case, the state space is significantly large since for every position in the *3-mer*, there are four different choices. The cardinality of the state space in this case is 64. Again, assuming that all the outcomes have the same probability, the probability of observing any *3-mer* is 1/64 and the amount of information obtained in a single observation is:

$$I(\textit{3-mer}) = -log\left(\frac{1}{64}\right) \approx 1.8 \tag{3.4}$$

Which is larger than then amount of information obtained in a single observation of a *1-mer*.

The average amount of information is usually referred to as entropy. Here, the entropy is the expected amount of information obtained in a single observation. Therefore, the entropy can be expressed as:

$$S = -\sum_{A \in E} p_A \log(p_A) \tag{3.5}$$

However, since all the events were equally likely in the previous examples, their entropies are going to have the same value as the information function for any event.

$$S = -\sum_{A \in E} p_A \log(p_A)$$

$$
\begin{aligned}
&= -\sum_{A \in E} p \log(p) \\
&= -|E| p \log(p) \\
&= -|E| \frac{1}{|E|} \log(p) \\
&= -\log(p) \\
&= I(A)
\end{aligned} \tag{3.6}
$$

In addition, it is of interest to look at the behavior of the entropy function as one increases the *k-mer* size. Assuming again that all events are equally likely, we have that:

$$
\begin{aligned}
I(A) &= -\log(p_A) \\
&= -\log\left(\frac{1}{|E|}\right) \\
&= -\log\left(\frac{1}{|N|^k}\right) \\
&= \log\left(|N|^k\right) \\
&= k \log\left(|N|\right) \\
&\propto k
\end{aligned} \tag{3.7}
$$

Therefore, the amount of information is going to be proportional to the *k-mer* size, thus proving, from an Information Theory standpoint, that in fact one can learn more from the sequence the larger the choice of *k-mer* is. However, using large *k-mers* is usually associated with both higher memory and processing requirements as Figure 3.1 reflects. Thus, a compromise between the amount of biological information and computational requirements is often required.

3.2 Modeling the state sequences

Using the notation introduced in section 2, the hidden state sequence is represented as follows:

$$X = X_1 \cdots X_m \tag{3.8}$$

and the probability of such a path would be:

$$\begin{aligned} P\{X|\Theta\} &= \pi_0(X_1)P(X_1, X_2) \cdots P(X_{m-1}, X_m) \\ &= \pi_0(X_1) \prod_{i=2}^{m} P(X_{i-1}, X_i) \end{aligned} \tag{3.9}$$

Let us consider the following sequence:

$$S = ACTAGACAGATGACA \tag{3.10}$$

Let us assume now that the *k-mer* size chosen to break down the sequence is three. Then, the sequence of states would be:

$$ACT \rightarrow AGA \rightarrow CAG \rightarrow ATG \rightarrow ACA \tag{3.11}$$

Thus, if $X = S$, then

$$X_1 = ACT$$

$$X_2 = AGA$$

$$X_3 = CAG$$

$$X_4 = ATG$$

$$X_5 = ACA$$

and the probability of observing that sequence of states would be:

$$P\{X = S|\Theta\} = \pi_0(ACT)P(ACT|AGA)P(AGA|CAG)P(CAG|ATG)P(ATG|ACA) \tag{3.12}$$

These probabilities would be found in the *transition matrix*. The probability can be computed in the same way for any path once the sequence of states is defined.

3.3 Modeling the symbol sequences

As previously introduced, each DNA sequence present in the NGS dataset is going to be regarded as a sequence of symbols. Using the notation introduced in section 2, the symbol sequence is represented as follows:

$$Y = Y_1 \cdots Y_m \tag{3.13}$$

Considering the model introduced in section 2, the HMM uses the emission probabilities to evaluate how likely is to observe a given symbol given the hidden state. Recall that:

$$P\{Y_n = y|X_n = i\} = e(y|i) \tag{3.14}$$

For instance, if one wants to model the probability that the ith nucleotide is observed to be Adenine given that the hidden state is Guanine, a value between zero and one can be assigned to that emission probability:

$$P\{Y_i = A|X_i = G\} = e(A|G) = 0.05 \tag{3.15}$$

In addition, recall that by the LTP:

$$\sum_{y \in E} e(Y_i = y | X_i = x) = 1, \qquad \forall\, y \in E,\ j = 1, ..., m \tag{3.16}$$

Note that the state space E is shared in this case between the symbols and the states. This is usually preferred for convenience when working with HMM. Nevertheless, in case one wanted to work with larger *k-mers* in order to capture more biological information, it would be convenient to compute the emission probabilities for *k-mers* with k larger than one.

Let us consider *k-mers* $R_1 = R_{11}...R_{1k}$ and $R_2 = R_{21}...R_{2k}$, which both have length k. The emission probability of observing R_2 provided that R_1 is the hidden state would be:

$$P\{R_2 = R_{21}...R_{2k} | R_1 = R_{11}...R_{1k}\} = e(R_{21}...R_{2k} | R_{11}...R_{1k}) \tag{3.17}$$

However, since a SNP database is going to be used to learn the emission probabilities, the method proposed is based on the assumption that the emission probabilities of adjacent nucleotides are independent. This assumption allows one to express the emission probability of observing *k-mer* $R_2 = R_{21}...R_{2k}$ given *k-mer* $R_1 = R_{11}...R_{1k}$ as a product of the different emission probabilities at a single base resolution:

$$\begin{aligned} e(R_2|R_1) &= e(R_{21}|R_{11}) e(R_{22}|R_{12}) \cdots e(R_{2k}|R_{1k}) \\ &= \prod_{i=1}^{k} e(R_{2i}|R_{1i}) \end{aligned} \tag{3.18}$$

where k would be the length of the *k-mer*. For instance, if we observe the symbol $Y_i = ACTAG$ and we know that the corresponding hidden state is $X_i = ACTTG$,

the emission probability would be:

$$P\{Y_i = ACTAG|X_i = ACTTG\} = e(ACTAG|ACTTG) \tag{3.19}$$

Then, assuming independence, this emission probability could be computed in the following way:

$$e(ACTAG|ACTTG) = e(A|A)e(C|C)e(T|T)e(A|T)e(G|G) \tag{3.20}$$

Each single emission probability would be obtained from a matrix learned from the SNP database previously mentioned. The exact way this matrix is constructed is explained in section 3.5.

3.4 Learning the transition matrix P

The dynamics of the *Markov chain* are totally defined by the *transition matrix*, and this can be deduced from the reference *genome* when working with DNA or from the reference *transcriptome* if one was dealing with RNA data. These references are based on multiple sequencing datasets from the same species that the scientific community has put together creating a consensus sequence. Therefore, they can be considered a good foundation for the *transition matrix*. That is, one can deduce the conditional probability of having a *k-mer* X_n after *k-mer* X_{n-1} from the reference. For instance, a *transition matrix* using *1-mers*, thus for state space $E = \{A, C, G, T\}$, could look like:

$$P = \begin{bmatrix} P(A,A) & P(A,C) & P(A,G) & P(A,T) \\ P(C,A) & P(C,C) & P(C,G) & P(C,T) \\ P(G,A) & P(G,C) & P(G,G) & P(G,T) \\ P(T,A) & P(T,C) & P(T,G) & P(T,T) \end{bmatrix} \tag{3.21}$$

A *transition matrix* has to fulfill the properties described in section 2.1. Therefore, the method suggested in this thesis to learn the *transition matrix* is to, based on a *k-mer* size choice, scan the reference of the organism being studied and count the transitions from any *k-mer* to any other *k-mer*, including itself. For instance, the probability:

$$P\{X_n = ACTGT | X_{n-1} = AAGTA\} = P(AAGTA, ACTGT) \tag{3.22}$$

can be estimated from the reference by counting how many times the *5-mer* AAGTA is followed by ACTGT and dividing by the number of times that AAGTA is present in it so that the row corresponding to that *5-mer* adds up to one as any *transition matrix* or *Markov matrix* requires. This process would have to be repeated for all the states in the state space E.

As stated at the beginning of this section, the choice of *k-mer* size has both memory and processing requirements. Table 3.1 illustrates the approximate amount of memory that the *transition matrix* would require for different *k-mer* sizes. Note that if one wanted to store the whole matrix in the random-access memory (RAM), this would require a RAM capacity that few machines have when the *k-mer* size choice gets large. For example, it looks feasible to store a *transition matrix* built based on *6-mers* in a regular machine. If one wanted to store larger *transition matrices*, then it would not probably be feasible in terms of memory requirements to run the algorithm in a regular desktop and something more powerful would be required. Note as well that there are going to be other matrices and variables to store in the RAM, hence the memory available for the *transition matrix* will be very limited depending on the capacity of the machine used.

Transition matrix sizes		
k-mer size	Elements in matrix	Memory required
1	16	150 B
2	256	2.2 KB
3	4,096	33 KB
4	65,536	515 KB
5	1,048,576	8.1 MB
6	16,777,216	129 MB
7	268,435,456	2.1 GB

Table 3.1: Memory required by the *transition matrix* for different *k-mer* sizes.

Nevertheless, storing this matrix in the RAM would be ideal in order to accelerate the lookup process, since this would be an operation repeated continuously in the method proposed. By storing it in the RAM, the lookup process would be much faster rather than having it saved in the hard drive and having the algorithm open that file, look for the number of interest and then close it again every time a *transition probability* was needed. In addition, when working with larger *k-mer* sizes, even the mere fact of storing the matrix in the hard drive could potentially be an issue.

Another issue to take into account, from the computational standpoint, is the time required to build this matrix. However, once the matrix is built for a given organism and *k-mer* size, the user would not need to go through this process again. The perl code developed can be found in Appendix A.1.1 with some comments that will help the reader to better understand the structure.

3.5 Learning the emission probabilities $e(\cdot|\cdot)$

As discussed in section 3.3, the emission probabilities of *k-mers* with k larger than one will be computed as the product of individual emission probabilities. Therefore, the matrix containing the emission probabilities is going to be the equivalent to a *transition matrix* based on *1-mers*. Note that this is a very different case when compared to the *transition matrix* scenario given that, in that case, we are interested in building matrices with the largest *k-mer* size as possible. The matrix containing the emission probabilities will be referred to as V and will contain only 16 elements:

$$V = \begin{bmatrix} e(A|A) & e(C|A) & e(G|A) & e(T|A) \\ e(A|C) & e(C|C) & e(G|C) & e(T|C) \\ e(A|G) & e(C|G) & e(G|G) & e(T|G) \\ e(A|T) & e(C|T) & e(G|T) & e(T|T) \end{bmatrix} \tag{3.23}$$

For testing purposes, these emission probabilities are going to be learned from a Human Variation database generated with human data by the National Center for Biotechnology Information (NCBI). The common format for variant calling is the so called Variant Call Format (VCF). In such a file, the third and fourth columns contain the information relevant to us, the sequence information of the variants found in the dataset. By counting the frequencies of each variant, the emission probability of a given nucleotide being observed given that the hidden state is the same nucleotide can be found then by subtracting to one the summation all the other possible outcomes:

$$e(i|i) = 1 - \sum_{j \in E \setminus i} e(j|i), \tag{3.24}$$

for all $i \in E$. Therefore, by parsing the training VCF file, the matrix containing the emission probabilities can be built in a relative simple and fast fashion. This matrix would probably be specific to each organism and updated as new variants were discovered. Again, the software suite developed will allow the user to generate his or her own emission probabilities matrix based on whatever SNP database he or she wants to use to train the HMM. The perl code developed can be found in Appendix A.1.2 with some comments that will help the reader to better understand the structure.

4. COMPUTATIONAL APPROACHES

In section 3, the HMM model has been applied to genomic sequencing data. However, it is still not clear how to compute the probabilities that the method presented in this thesis seeks to find. This section will introduce the way these are to be found from a computational standpoint. By taking advantage of *dynamic programming*, the complexity of the algorithms will be significantly reduced when compared to an exhaustive search approach.

First, the forward algorithm will be adapted to allow us to compute the probability of observing a given sequence of symbols given the model Θ. By doing so, the probability of observing a given sequence of symbols $\{Y = y\}$, without knowing the corresponding underlying sequence of states X, can be computed. This will serve as a score for the sequence in that the higher the probability, the more confident one can be about it. In addition, one can compare the scores obtained to that of randomly generated sequences to get a sense of how significant these are.

Second, an adaptation of the well-known Viterbi algorithm will be introduced. This algorithm will allow us to find the optimal path X^*, i.e. the sequence of states that better explain the observed sequence of symbols $\{Y = y\}$. In addition, the probability of this path will also be computed, thus obtaining a second score for the sequence. Again, the higher this probability is, the more confident that one can be about the sequence observed.

Third, another scoring approach is suggested by using the optimal path X^* found through the Viterbi algorithm to compute the conditional probability of observing $\{Y = y\}$ given the optimal path and the model, i.e. Θ.

4.1 Adaptation of the forward algorithm

It is of interest to compute the probability of observing a given sequence of symbols based on a HMM. In other words, to compute the probability:

$$P\{Y = y|\Theta\} \tag{4.1}$$

This problem is usually referred to as the *scoring problem* [17]. In this case, the corresponding underlying sequence of states is not known. However, recall that, as discussed previously, by the LTP:

$$P\{Y = y|\Theta\} = \sum_{x \in \Omega_m} P\{Y = y, X = x|\Theta\} \tag{4.2}$$

Therefore, by considering all the possible underlying sequences of states, one could eventually find the marginal probability of a given observation $\{Y = y|\Theta\}$. Nevertheless, this would not be feasible for somewhat elaborated hidden Markov models. Recall, that there would be as many as $|E|^m$ possible combinations, where $|E|$ is the cardinality of the state space and m the number of elements in a given symbol sequence. Therefore, this number grow exponentially with the length of the observation being studied.

The *forward algorithm* is an algorithm that allows us to deal with this issue. It is based on *dynamic programming*, and can compute the probability of interest in a rather efficient way [9]. This computational approach consists in solving a complex problem by breaking it down into smaller problems that are much simpler, solving these and storing the solutions to finally find the answer to the larger problem by combining all these solutions. In this case, the following recursive variable is defined:

$$\alpha(n,i) = P\{Y_1 = y_1, ..., Y_n = y_n, X_n = i|\Theta\} \tag{4.3}$$

and this variable is recursively computed:

$$\alpha(n,i) = \sum_{s\in E} \alpha(n-1,s)P(s,i)e(y_n|i), \quad n = 2, ..., m \tag{4.4}$$

Therefore, the first three iterations would be:

1. Set $n = 1$ and, for all i in E compute and store all the $\alpha(1,\cdot)$

$$\begin{aligned} \alpha(1,i) &= P\{Y_1 = y_1, X_1 = i|\Theta\} \\ &= \pi_0(i)e(y_1|i) \end{aligned} \tag{4.5}$$

2. Set $n = 2$ and, for all i in E compute and store all the $\alpha(2,\cdot)$

$$\begin{aligned} \alpha(2,i) &= P\{Y_1 = y_1, Y_2 = y_2, X_2 = i|\Theta\} \\ &= \sum_{s\in E} \alpha(1,s)P(s,i)e(y_2|i) \end{aligned} \tag{4.6}$$

3. Set $n = 3$ and, for all i in E compute and store all the $\alpha(3,\cdot)$

$$\begin{aligned} \alpha(3,i) &= P\{Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, X_3 = i|\Theta\} \\ &= \sum_{s\in E} \alpha(2,s)P(s,i)e(y_3|i) \end{aligned} \tag{4.7}$$

Note that there are going to be as many recursive variables $\alpha(n,\cdot)$ as the size of the cardinality of the state space $|E|$. In addition, we would need as many iterations as *k-mers* there are in the symbol sequence of interest. In order to improve the

performance of the algorithm, since each recursive variable will be independent, this calculation could be multi-threaded.

Once the recursions are completed, then the probability of interest can be computed in the following way:

$$P\{Y = y|\Theta\} = \sum_{s \in E} \alpha(m, s) \tag{4.8}$$

As a result, the probability $\{Y = y|\Theta\}$ is obtained with a complexity of $O(m|E|^2)$. Note that the complexity of the algorithm is linear with respect to the number of symbols m in the sequence observed and quadratic with respect to the cardinality of the space state E. This makes a huge difference when compared to the initial guess, which grows exponentially with the length of the sequence of symbols.

4.2 Adaptation of the Viterbi algorithm

It is of interest as well to find the optimal sequence of states that can better explain the observed sequence of symbols. More technically, it is of interest to find:

$$x^* = \arg\max_{x \in \Omega_m} P\{X = x|Y = y, \Theta\} \tag{4.9}$$

Given Bayes' rule,

$$\begin{aligned} x^* &= \arg\max_{x \in \Omega_m} P\{X = x|Y = y, \Theta\} \\ &= \arg\max_{x \in \Omega_m} \frac{P\{X = x, Y = y, \Theta\}}{P\{Y = y, \Theta\}} \\ &= \arg\max_{x \in \Omega_m} \frac{P\{X = x, Y = y|\Theta\}P\{\Theta\}}{P\{Y = y|\Theta\}P\{\Theta\}} \\ &= \arg\max_{x \in \Omega_m} \frac{P\{X = x, Y = y|\Theta\}}{P\{Y = y|\Theta\}} \end{aligned} \tag{4.10}$$

Note that the denominator is going to be a constant that will effect every possible candidate path in the same way. Therefore, maximizing equation 4.9 is the same as maximizing:

$$x^* = \arg\max_{x \in \Omega_m} P\{X = x, Y = y|\Theta\} \tag{4.11}$$

As in section 4.2, enumerating all possible states might be unfeasible depending on the length of the sequence of observations and the cardinality of the space state. Therefore, we can benefit again from using *dynamic programming* to find an efficient way to find the optimal path as well as its conditional probability based on the observations. The well-known *Viterbi algorithm* fits this scenario perfectly [3]. In this case, we define the variable:

$$\gamma(n, i) = \max_{x_1,...,x_{n-1}} P\{Y_1 = y_1, ..., Y_n = y_n, X_1 = x_1, ..., X_{n-1} = x_{n-1}, X_n = i|\Theta\} \tag{4.12}$$

for $n = 2, ..., m$ and we compute it recursively in the following way:

$$\gamma(n, i) = \max_{s \in E} \gamma(n-1, s) P(s, i) e(y_n|i) \tag{4.13}$$

Therefore, the first three iterations would be:

1. Set $n = 1$ and, for all i in E compute and store all the $\gamma(1, \cdot)$

$$\begin{aligned} \gamma(1, i) &= P\{Y_1 = y_1, X_1 = i|\Theta\} \\ &= \pi_0(i) e(y_1|i) \end{aligned} \tag{4.14}$$

2. Set $n = 2$ and, for all i in E compute and store all the $\gamma(2, \cdot)$

$$\begin{aligned}\gamma(2, i) &= \arg\max_{s \in E} P\{Y_1 = y_1, Y_2 = y_2, X_1 = s, X_2 = i|\Theta\} \\ &= \arg\max_{s \in E} \gamma(1, s) P(s, i) e(y_2|i) \end{aligned} \tag{4.15}$$

3. Set $n = 3$ and, for all i in E compute and store all the $\gamma(3, \cdot)$

$$\begin{aligned}\gamma(3, i) &= \arg\max_{s \in E} P\{Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, X_2 = s, X_3 = i|\Theta\} \\ &= \arg\max_{s \in E} \gamma(2, s) P(s, i) e(y_3|i) \end{aligned} \tag{4.16}$$

As in the *forward algorithm* case, note that there are going to be as many recursive variables $\gamma(n, \cdot)$ as the size of the cardinality of the state space $|E|$. In addition, we would need as many iterations as *k-mers* present in the symbol sequence of interest.

Once equation 4.13 is computed, the maximum observation probability can be calculated as:

$$\begin{aligned}P^* &= \max_{x \in \Omega_m} P\{Y = y, X = x|\Theta\} \\ &= \max_{s \in E} \gamma(m, s) \end{aligned} \tag{4.17}$$

Finally, the optimal path x^* can be found by tracing back the states that led tot P^*. As in the *forward algorithm*, the *Viterbi algorithm* has an complexity of $O(m|E|^2)$, thus the time required will be proportional to the length of the sequence of symbols and quadratic to respect the cardinality of the space state.

4.3 Using the optimal path X^* to score an observation

Once the optimal path X^* has been obtained by using the *Viterbi algorithm*, it is possible to compute the following probability as well:

$$P\{Y = y|X^*, \Theta\} \tag{4.18}$$

This is pretty straightforward since this probability is simply the product of the different emission probabilities.

$$P\{Y = y|X^*, \Theta\} = \prod_{n=1}^{m} e(y_n|x_n^*) \tag{4.19}$$

5. RESULTS

In order to evaluate the performance of the algorithms introduced in sections 4.1 and 4.2, a set of data has been generated. Firstly, six files have been generated containing each 250 random sequences of the same length. For instance, the first file contains 250 sequences of length ten. The second file contains 250 sequences of length twenty, and so forth. On the other hand, six files have been generated by randomly drawing sequences from the reference from which the *transition matrix* has been generated. Again, the first file contains 250 sequences of length ten, the second one contains 250 sequences of length twenty, and so forth. In total, the dataset generated to test the methods described in this thesis consist of twelve FASTA files with 250 sequences each. In addition to this dataset, data from [12] have been used as well to test HiMMe with real NGS data.

In this section, two main results are presented. Firstly, all the sequences have been scored based on the HMM model built. The results suggest that the algorithm is able to recognize those sequences that belong to the reference and, in turn, score them higher than those that have been randomly generated. Secondly, the randomly generated sequences are corrected using the algorithm described in section 4.2 to achieve a higher score.

5.1 Scoring sequences

The algorithm described in section 4.1 has been used to score randomly generated sequences as well as sequences derived from the reference used to learn the *transition matrix*. The purpose of this comparison is to see whether the algorithm scores higher those sequences that belong to the reference from which the *transition matrix* was learned. Intuitively, the larger the sequences are, the larger the difference between

scores should be given the amount of information the sequences would contain. In addition, one would expect to encounter more differences between scores as the *k-mer* size increases, since the algorithm should be able to identify more genomic patterns in the sequences.

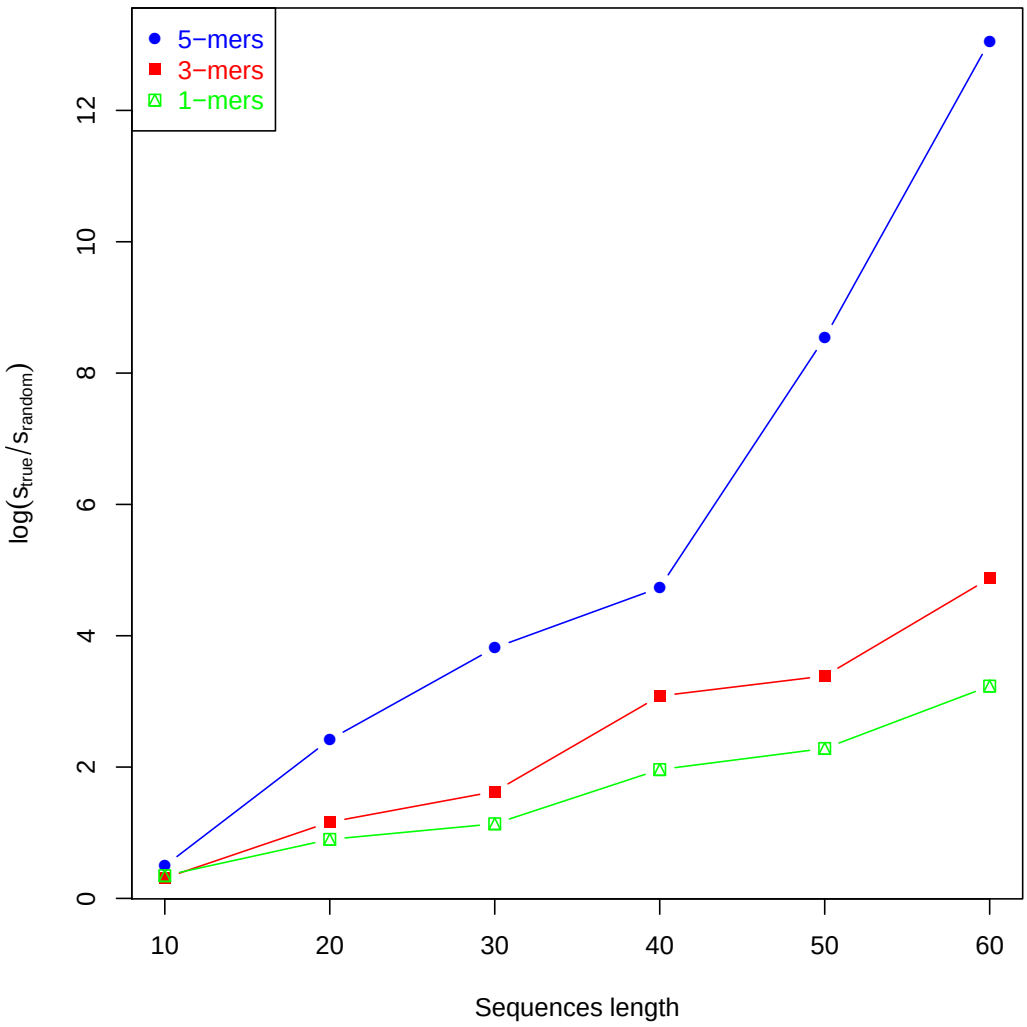


Figure 5.1: Logarithm of the score ratio between the random and true sequences as function of the sequence length.

In Figure 5.1 the logarithm of the ratio of the average scores for different sequence lengths and *k-mer* sizes has been plotted. Note that as the length of the sequences increases, the ratio between true and random sequences increases. Therefore, the hypothesis that the longer the sequences, the larger the discriminant power is, is confirmed. In addition, as the *k-mer* size increases, this ratio seems to be more prominent. That is consistent again with the fact that the larger the *k-mers* are, the more biological information they are able to carry.

In order to statistically determine whether there is a significant difference between the random and true scores obtained, a t-test has been performed for each possible combination.

t-test summary			
Sequence length	p-value (1-mers)	p-value (3-mers)	p-value (5-mers)
10 bp	1e-10	1.1e-07	1.3e-09
20 bp	4.4e-23	1.3e-17	8.6e-03
30 bp	8.9e-17	1.e-13	4.7e-02
40 bp	4.8e-23	2.6e-07	9.9e-05
50 bp	4.5e-15	2.5e-09	0.1
60 bp	9.9e-16	6.3e-10	0.05

Table 5.1: Statistics for each comparison.

All the p-values, except for one, conclude that the differences between score populations were different with a 95% confidence level. As the *k-mer* size increases, the magnitude of the p-values, however, seems to increase. This is due to the fact

that as the *k-mer* size increases, the difference in score magnitude decreases faster than the standard deviation does. Therefore, the t-test loses power as the *k-mer* size increases. However, as depicted in Figure 5.1, the algorithm is capable of scoring higher those sequences that were sampled from the reference and this power increases with the length of the sequences as well as with the *k-mer* size.

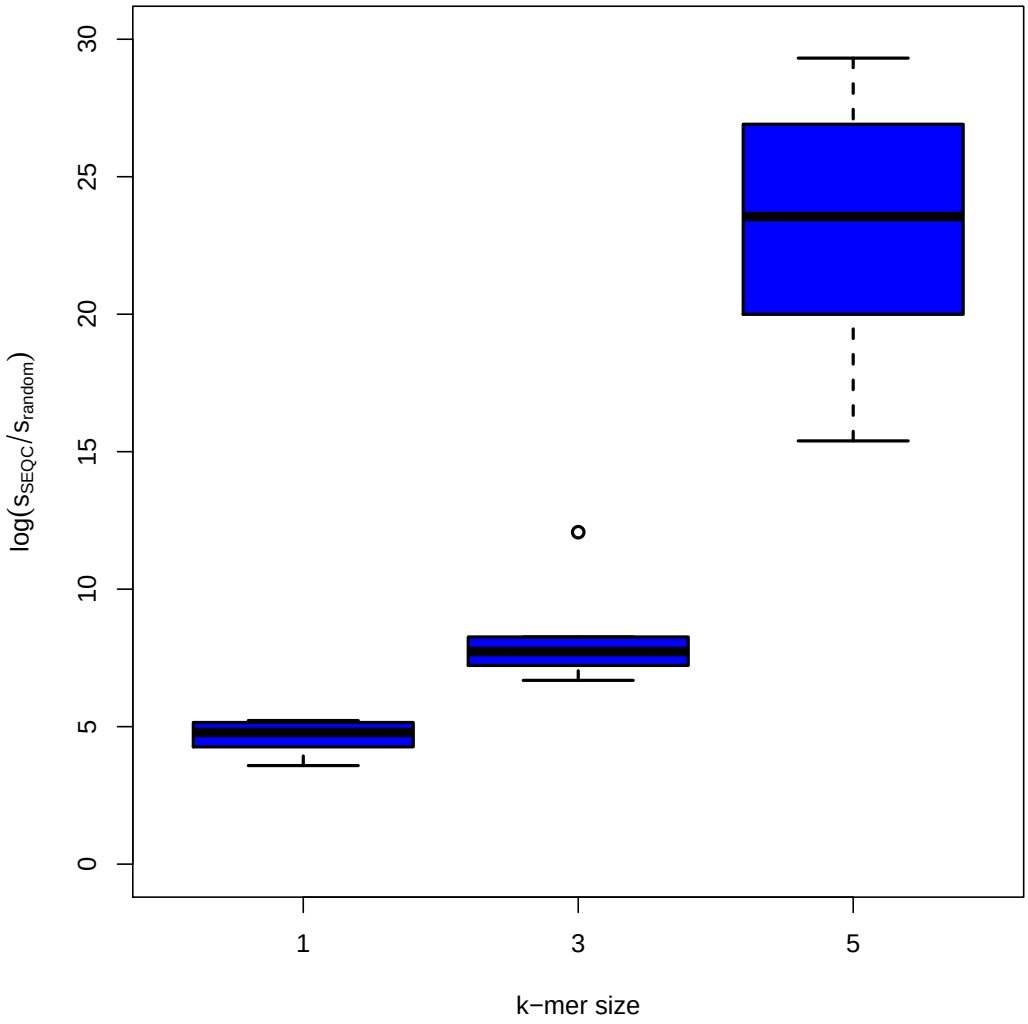


Figure 5.2: Boxplots of the logarithm of the score ratios between the sequences sampled from SEQC and the random sequences for *k-mer* sizes 1, 3 and 5 respectively.

HiMMe has been used as well to score part of the RNA-seq data from [12]. This dataset consists of NGS data that come from six different sequencing centers: the Australian Genome Research Facility, the Beijing Genomics Institute, City of Hope, Cornell, Mayo Clinic and Novartis, the pharmaceutical company. For each sequencing center present in the dataset, 250 sequences of length 100 have been randomly extracted and scored. Note in Figure 5.2 that the sequences extracted from their dataset score significantly higher than the randomly generated ones. Furthermore, the ratio grows as the *k-mer* size used grows as in Figure 5.1, suggesting again that the larger the choice of *k-mer* is, the more genetic information and patterns the algorithm is able to identify in the data. The code used to generate the random sequences, learn the *transition matrix*, learn the emission probabilities and score FASTA files, can be found in Appendix A.1.

5.2 Correcting sequences

The algorithm described in section 4.2 has been applied to the random sequences. As a result, these sequences have been corrected so that the joint probability

$$P\{X = x, Y = y|\Theta\} \tag{5.1}$$

is maximized. The results are illustrated in Figure 5.3. Note that for *1-mers*, the score of the corrected versions does not improve significantly compared to that of the original random reads. However, for *3-mers* and *5-mers*, the algorithm is capable of improving the scores significantly. For instance, when using *3-mers*, there is a twofold increase when the length of the sequence is above forty nucleotides. When using *5-mers*, the improvement is even more prominent, reaching an eightfold increase when the length of the sequence is fifty nucleotides.

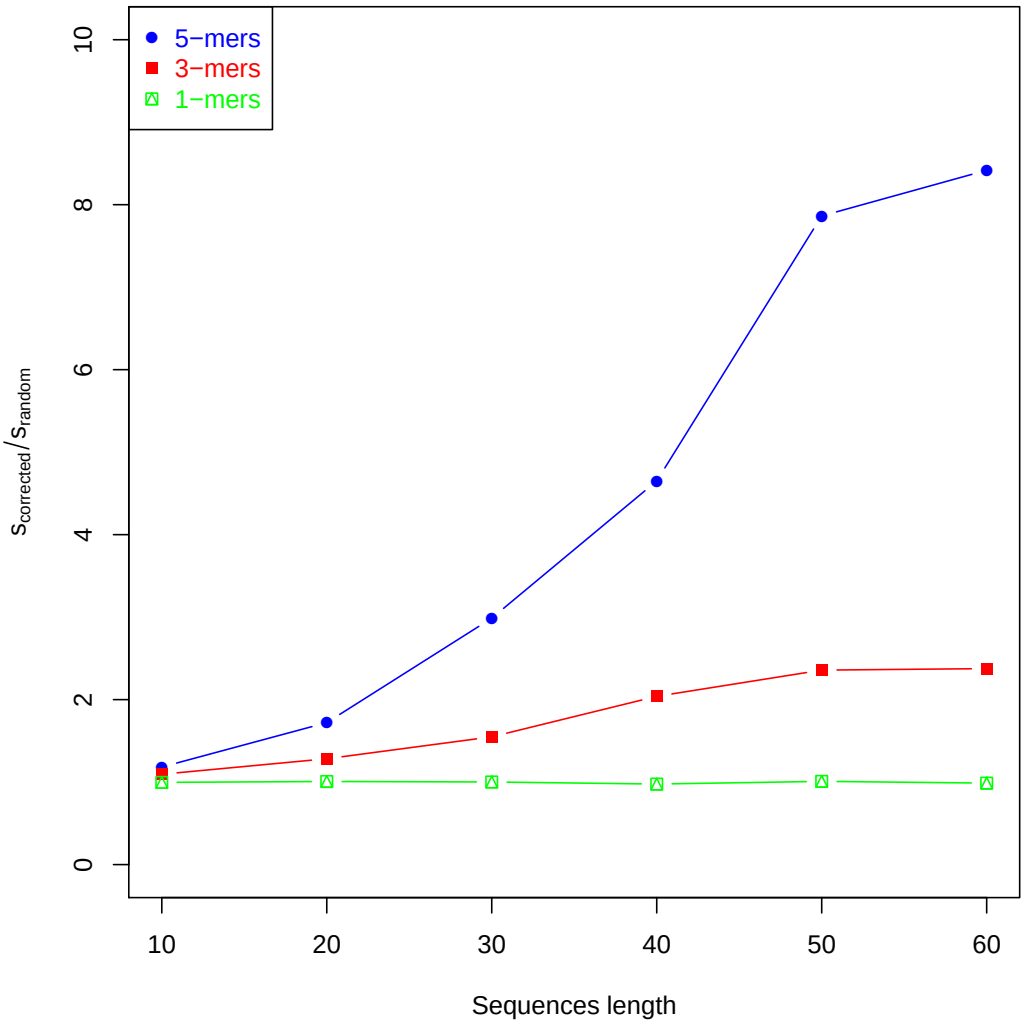


Figure 5.3: Score ratio between the corrected and original sequences as function of the sequence length.

5.3 Speed assessment

In order to increase the efficiency, since both algorithms share some common steps, they have been implemented together.

Time elapsed running HiMMe			
Sequence length	*1-mers*	*3-mers*	*5-mers*
10 bp	< 1.0 s	3.0 s	721.0 s
20 bp	< 1.0 s	7.5 s	2057.0 s
30 bp	< 1.0 s	13.0 s	3497.0 s
40 bp	< 1.0 s	16.5 s	4860.0 s
50 bp	< 1.0 s	20.5 s	5340.0 s
60 bp	< 1.0 s	25.5 s	7100.0 s

Table 5.2: Time elapsed when running HiMMe on the generated dataset.

Note in Table 5.2 how the sequence length, as explained before, has a somewhat linear relationship with the time elapsed. However, the *k-mer* choice ends up having a huge impact on it and drastically affects the processing time. Recall that the complexity of the algorithms introduced in section 4 grows quadratically with the cardinality of the space state. In addition, the latter grows exponentially with the *k-mer* size.

Average speed HiMMe		
1-mers	*3-mers*	*5-mers*
10,000 bp/s	646 bp/s	2.42 bp/s

Table 5.3: Average number of nucleotides processed per second by HiMMe.

In Table 5.3 the average number of nucleotides processed by HiMMe is summarized. As the *k-mer* size gets larger, more memory is required to store not only the

transition matrix but also all the recursive variables involved in the algorithm and the search space grows exponentially. Combined, this has a huge impact in terms of performance.

6. CONCLUSIONS

The sequence scoring method proposed in this thesis is capable of scoring higher those sequences that were used to construct the hidden Markov model than those that were randomly generated. In other words, it is able to identify those patterns in the sequencing data that are more likely to happen based on previous knowledge about the organism being studied. This method can be applied in multiple cases when analyzing NGS data.

One application could be to evaluate NGS reads quality. Although FASTQ files do contain such information, this would serve as another metric to quantify how confident one can be about the reads that the sequencing machine produced. A threshold could be used to determine which reads are kept for the downstream analysis for instance.

Another application that the scoring method proposed here could have, would be to evaluate the reliability of assemblies, either genomes or transcriptomes. One could generate different assemblies with different parameters and pick the one with the highest score. Also, by using these scores as a benchmark, this method would allow researchers in the field of assemblers to assess the performance of their algorithms and improve based on that.

Finally, in a metagenomics scenario, where one wants to identify the organisms present in a given environment, this tool could be used to classify the reads obtained into the different organisms present in the genetic material recovered. Different models should be learned for the species being studied and the algorithm would allow the researcher to find the model, and thus organism, that better explains the observed reads. These are only some of the applications that the method proposed

in this thesis has.

As for the repairing algorithm proposed, it is capable of identifying potential errors and repairing them. The scores obtained once the sequences are repaired are higher, thus improving the reliability of the data. This has potentially a plethora of applications as well. For instance, one could run the algorithm on NGS data, before the downstream analysis is performed, to ensure that the dataset used in the analysis is as reliable as possible. The same could be done with assemblies to correct errors induced throughout the whole process, specially in rather complex organisms with a lot of polymorphism in their genes.

A clear advantage of the method proposed in this thesis is that there is no alignment involved and that, in turn, it could be faster than some of the state of the art scoring and error-correcting tools. In a nutshell, the HMM model, with the appropriate *k-mer* choice, seems to be able to capture the biological information and detect potential errors in NGS data. In this thesis, two applications of this stochastic model have been described and implemented leading, in both cases, to positive results.

REFERENCES

[1] Cinlar, E. (2013). *Introduction to stochastic processes*. Courier Corportation.

[2] Crick, F. (1970). Central dogma of molecular biology. *Nature*, 227(5258):561–563.

[3] Forney Jr., G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

[4] Illumina (2012). An introduction to next-generation sequencing technology. Retrieved from: http://www.illumina.com/.

[5] Jones, P. A. and Takai, D. (2001). The role of DNA methylation in mammalian epigenetics. *Science*, 293(5532):1068–1070.

[6] Li, E. (2002). Chromatin modification and epigenetic reprogramming in mammalian development. *Nature Reviews Genetics*, 3(9):662–673.

[7] Mardis, E. R. (2011). A decade's perspective on DNA sequencing technology. *Nature*, 470(7333):198–203.

[8] Marioni, J. C., Mason, C. E., Mane, S. M., Stephens, M., and Gilad, Y. (2008). RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research*, 18(9):1509–1517.

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

[10] Sanger, F., Nicklen, S., and Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467.

[11] Schones, D. E., Cui, K., Cuddapah, S., Roh, T.-Y., Barski, A., et al. (2008). Dynamic regulation of nucleosome positioning in the human genome. *Cell*, 132(5):887–898.

[12] SEQC/MAQC-III Consortium (2014). A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the sequencing quality control consortium. *Nature Biotechnology*, 32(9):903–914.

[13] Shendure, J., Mitra, R. D., Varma, C., and Church, G. M. (2004). Advanced sequencing technologies: methods and goals. *Nature Reviews Genetics*, 5(5):335–344.

[14] Watson, J. D. and Crick, F. H. (1953a). Molecular structure of nucleic acids. *Nature*, 171(4356):737–738.

[15] Watson, J. D. and Crick, F. H. (1953b). The structure of DNA. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 18, pages 123–131. Cold Spring Harbor Laboratory Press.

[16] Wetterstrand, K. (2012). DNA sequencing costs: data from the NHGRI genome sequencing program (GSP). Retrieved from: http://www.genome.gov/sequencingcosts/.

[17] Yoon, B. J. (2009). Hidden Markov models and their applications in biological sequence analysis. *Current Genomics*, 10(6):402–415.

APPENDIX A

A.1 Code

A.1.1 Learning the transition matrix

```
#!/usr/bin/env bash
# ----------------------------------------------------------------------------
##The MIT License (MIT)
##
##Copyright (c) 2016 Jordi Abante
##
##Permission is hereby granted, free of charge, to any person obtaining a copy
##of this software and associated documentation files (the "Software"), to deal
##in the Software without restriction, including without limitation the rights
##to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
##copies of the Software, and to permit persons to whom the Software is
##furnished to do so, subject to the following conditions:
##
##The above copyright notice and this permission notice shall be included in all
##copies or substantial portions of the Software.
##
##THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
##IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
##FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
##AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
##LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
##OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
##SOFTWARE.
# ----------------------------------------------------------------------------
shopt -s extglob

abspath_script="$(readlink -f -e "$0")"
script_absdir="$(dirname "$abspath_script")"
script_name="$(basename "$0" .sh)"

# Find perl scripts
```

```
perl_script="${script_absdir}/perl/${script_name}.pl"

if [ $# -eq 0 ]
    then
        cat "$script_absdir/${script_name}_help.txt"
        exit 1
fi

TEMP=$(getopt -o hd:t:k: -l help,outdir:,threads:,kmer_size: -n "$script_name.sh" --
    "$@")

if [ $? -ne 0 ]
then
  echo "Terminating..." >&2
  exit -1
fi

eval set -- "$TEMP"

# Defaults
outdir="$PWD"
threads=2
kmer_size=1

# Options
while true
do
  case "$1" in
    -h|--help)
      cat "$script_absdir"/${script_name}_help.txt
      exit
      ;;
    -d|--outdir)
      outdir="$2"
      shift 2
      ;;
    -t|--threads)
      threads="$2"
      shift 2
```

```
      ;;
    -k|--kmer_size)
      kmer_size="$2"
      shift 2
      ;;
    --)
      shift
      break
      ;;
    *)
      echo "$script_name.sh: Internal error!"
      exit -1
      ;;
  esac
done

# Print LICENSE
cat "${script_absdir}/../../LICENSE"

# Inputs
input="$1"

# Output
input_basename="$(basename "$input")"
prefix="${input_basename%%.*}"
tempfile="${outdir}/${prefix}"
outfile="${outdir}/${prefix}_tm${kmer_size}.txt.gz"

# Output directory
mkdir -p "$outdir"

# Count number of entries in FASTA
n_entries="$(zcat -f "$input" | grep "^>" | wc -l)"

# Run
zcat -f "$input" | "$perl_script" "$input" "$kmer_size" "$n_entries" "$outfile"
```

```
#!/usr/bin/env perl
# ------------------------------------------------------------------------
```

```
3 ##The MIT License (MIT)
4 ##
5 ##Copyright (c) 2016 Jordi Abante
6 ##
7 ##Permission is hereby granted, free of charge, to any person obtaining a copy
8 ##of this software and associated documentation files (the "Software"), to deal
9 ##in the Software without restriction, including without limitation the rights
10 ##to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
11 ##copies of the Software, and to permit persons to whom the Software is
12 ##furnished to do so, subject to the following conditions:
13 ##
14 ##The above copyright notice and this permission notice shall be included in all
15 ##copies or substantial portions of the Software.
16 ##
17 ##THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
18 ##IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
19 ##FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
20 ##AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
21 ##LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
22 ##OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
23 ##SOFTWARE.
24 #------------------------------------------------------------------------------
25
26 # Libraries
27 use strict;
28 use Algorithm::Combinatorics qw(combinations variations_with_repetition);
29
30 # Read arguments
31 my $scriptname = $0;              # Get script name
32 my $fasta_file = @ARGV[0];        # Get target FASTA file name
33 my $kmer_size= @ARGV[1];          # Get user k-mer size
34 my $n_entries=@ARGV[2];           # Number of entries in the FASTA file
35 my $outfile= @ARGV[3];            # Output file
36
37 # Variables
38 my $FASTA;                        # Fasta file handler
39 my $dim=1;                        # Dimension P (based on k-mer size)
40 my @markov_matrix=();             # 3D array [assembly][row][column]
41 my $n_proc=0;                     # Number of entries processed
```

```
my $n_mem=0;                    # Number of entries stored in RAM
my $n_limit=2500;               # Limit for number of entries in RAM

# Time stamps
my $st_time=0;                  # Start time
my $end_time=0;                 # End time
my $current_time=0;             # Current time
my $elapsed_time=0;             # Time elapsed

# Hashes
my %fasta_hash=();              # Hash containing sequence info of each sample
my %transition_hash=();         # Hash containing duplets combinations

################################## Main ########################################

# Read in fasta file
$st_time = localtime;
print STDERR "${st_time}: FASTA file: ${fasta_file}\n";

# Markov matrices
$current_time = localtime;
print STDERR "${current_time}: Initializing Markov matrix ...\n";
initialize();
$current_time = localtime;
print STDERR "${current_time}: Learning Markov matrix ...\n";
fill_markov_matrix();

# Print stuff
$current_time = localtime;
print STDERR "${current_time}: Saving Markov matrix in ${outfile}...\n";
print_markov_matrices();

################################## Subs ########################################
## Fill the markov matrix
sub fill_markov_matrix
{
    # Process in chunks of n_limit
 while($n_proc<$n_entries)
    {
```

```
81          # Read n_lim entries
82          read_fasta();
83
84          # Get chromosomes from fasta
85          my @entries=keys %fasta_hash;
86          foreach my $entry (@entries)
87          {
88              for(my $i=0;$i<=(scalar @{$fasta_hash{$entry}})-2*$kmer_size;$i++)
89              {
90                  # Get nucleotides of the iteration
91                  my $seq_1=@{$fasta_hash{$entry}}[$i];
92                  my $seq_2=@{$fasta_hash{$entry}}[$i+$kmer_size];
93                  for(my $j=1;$j<$kmer_size;$j++)
94                  {
95                      $seq_1=$seq_1.@{$fasta_hash{$entry}}[$i+$j];
96                      $seq_2=$seq_2.@{$fasta_hash{$entry}}[$i+$kmer_size+$j];
97                  }
98                  # Get codon index form markov_matrix
99                  my $row=$transition_hash{$seq_1};
100                 my $col=$transition_hash{$seq_2};
101                 # Fill markov_matrix
102                 $markov_matrix[$row][$col]+=1;
103             }
104             # Get rid of that entry
105             delete $fasta_hash{$entry};
106             $n_mem--;
107             # Update progress
108             $n_proc++;
109             my $perc=$n_proc/$n_entries*100;
110             printf STDERR "\rCurrent progress: %.2f%", $perc;
111         }
112     }
113     printf STDERR "\n";
114     # Scale matrix
115     my $sum;
116     for(my $i=0;$i<=$dim;$i++)
117     {
118         $sum=0;
119         # Count frequencies per row
```

```
120         for(my $j=0;$j<=$dim;$j++)
121         {
122             $sum+=$markov_matrix[$i][$j];
123         }
124         # Normalize each row
125         if($sum!=0)
126         {
127             for(my $j=0;$j<=$dim;$j++)
128             {
129                 $markov_matrix[$i][$j]/=$sum;
130             }
131         }
132     }
133 }
134
135 ## Initialize markov_matrices
136 sub initialize
137 {
138  # Nucleotides taken into consideration
139  my @nucleotides=('A','C','G','T');
140  for(my $i=1;$i<=$kmer_size;$i++)
141  {
142   $dim*=(scalar @nucleotides);
143  }
144  # Because we start with dim=0 in the loops
145  $dim-=1;
146  # Get all possible combinations of kmer_size nucleotides
147  my @permutations=variations_with_repetition(\@nucleotides,$kmer_size);
148  # Codify numerically each possible permutation
149  my $i=0;
150  foreach my $combination (@permutations)
151  {
152   my $length=$kmer_size-1;
153   my $sequence = join('', @{$combination}[0..$length]);
154   $transition_hash{$sequence}=$i;
155   $i++;
156  }
157  # Initialize markov_matrix
158  for(my $i=0;$i<=$dim;$i++)
```

```
 {
     for(my $j=0;$j<=$dim;$j++)
  {
   $markov_matrix[$i][$j]=0;
  }
 }
}

## Read in fasta file
sub read_fasta
{
    my $entry;
 while((my $line = <STDIN>) and ($n_mem<=$n_limit))
 {
  chomp($line);
  if( $line =~ />/)
  {
            $entry=substr($line,1); # Get rid of leading ">" character
            $n_mem++;
  }
  else
  {
      my @array = split //, $line;
      push @{$fasta_hash{$entry}},@array;
  }
 }
}

## Print fasta
sub print_fasta
{
 # Print output
 foreach my $key (sort keys %fasta_hash)
 {
  my @entry=@{$fasta_hash{$key}};
  print ">$key\n";
  foreach my $nuc (@entry)
  {
      print "$nuc";
```

```
198   }
199   print "\n";
200  }
201 }
202
203 ## Print markov_matrices
204 sub print_markov_matrices
205 {
206     open(OUT, "|gzip -c > ${outfile}")            # $$ is our process id
207         or die "Can't open file '${outfile}' $!";
208  # Reverse hash
209  my %reverse_hash = reverse %transition_hash;
210     # Print to OUT
211  print OUT "\t";
212  for(my $j=0;$j<=$dim;$j++)
213  {
214      my $key = $reverse_hash{$j};
215      print OUT "$key\t";
216  }
217  print OUT "\n";
218  for(my $i=0;$i<=$dim;$i++)
219  {
220      my $key = $reverse_hash{$i};
221      print OUT "$key\t";
222      for(my $j=0;$j<=$dim;$j++)
223      {
224   printf OUT "%.5f\t", $markov_matrix[$i][$j];
225      }
226      print OUT "\n";
227  }
228     close OUT;
229 }
230 ###############################################################################
```

A.1.2 Learning the emission probabilities

```
1 #!/usr/bin/env bash
2 # ------------------------------------------------------------------------------
3 ##The MIT License (MIT)
```

```
##
##Copyright (c) 2016 Jordi Abante
##
##Permission is hereby granted, free of charge, to any person obtaining a copy
##of this software and associated documentation files (the "Software"), to deal
##in the Software without restriction, including without limitation the rights
##to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
##copies of the Software, and to permit persons to whom the Software is
##furnished to do so, subject to the following conditions:
##
##The above copyright notice and this permission notice shall be included in all
##copies or substantial portions of the Software.
##
##THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
##IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
##FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
##AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
##LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
##OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
##SOFTWARE.
# ------------------------------------------------------------------------------
shopt -s extglob

abspath_script="$(readlink -f -e "$0")"
script_absdir="$(dirname "$abspath_script")"
script_name="$(basename "$0" .sh)"

# Find perl scripts
perl_script="${script_absdir}/perl/${script_name}.pl"

if [ $# -eq 0 ]
    then
        cat "$script_absdir/${script_name}_help.txt"
        exit 1
fi

TEMP=$(getopt -o hd:t:b: -l help,outdir:,threads:,bases: -n "$script_name.sh" -- "$@
    ")

```

```
42 if [ $? -ne 0 ]
43 then
44   echo "Terminating..." >&2
45   exit -1
46 fi
47
48 eval set -- "$TEMP"
49
50 # Defaults
51 outdir="$PWD"
52 threads=2
53 bases=10000000
54
55 # Options
56 while true
57 do
58   case "$1" in
59     -h|--help)
60       cat "$script_absdir"/${script_name}_help.txt
61       exit
62       ;;
63     -d|--outdir)
64       outdir="$2"
65       shift 2
66       ;;
67     -t|--threads)
68       threads="$2"
69       shift 2
70       ;;
71     -b|--bases)
72       bases="$2"
73       shift 2
74       ;;
75     --)
76       shift
77       break
78       ;;
79     *)
80       echo "$script_name.sh: Internal error!"
```

```
      exit -1
      ;;
  esac
done

# Print LICENSE
cat "${script_absdir}/../../LICENSE"

# Inputs
input="$1"

# Output
input_basename="$(basename "$input")"
prefix="${input_basename%%.*}"
tempfile="${outdir}/${prefix}"
outfile="${outdir}/${prefix}_ep${bases}.txt.gz"

# Output directory
mkdir -p "$outdir"

# Count number of entries in FASTA
n_entries="$(zcat -f "$input" | grep -v "^#" | wc -l)"

# Run
zcat -f "$input" | "$perl_script" "$input" "$n_entries" "$bases" "$outfile"
```

```
#!/usr/bin/env perl
# ------------------------------------------------------------------------
##The MIT License (MIT)
##
##Copyright (c) 2016 Jordi Abante
##
##Permission is hereby granted, free of charge, to any person obtaining a copy
##of this software and associated documentation files (the "Software"), to deal
##in the Software without restriction, including without limitation the rights
##to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
##copies of the Software, and to permit persons to whom the Software is
##furnished to do so, subject to the following conditions:
##
```

```
##The above copyright notice and this permission notice shall be included in all
##copies or substantial portions of the Software.
##
##THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
##IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
##FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
##AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
##LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
##OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
##SOFTWARE.
# ------------------------------------------------------------------------------

# Libraries
use strict;
use Algorithm::Combinatorics qw(combinations variations_with_repetition);

# Read arguments
my $scriptname = $0;              # Get script name
my $vcf_file = @ARGV[0];          # Get target FASTA file name
my $n_entries=@ARGV[1];           # Number of entries in the FASTA file
my $n_bases=@ARGV[2];             # Number of bases
my $outfile=@ARGV[3];             # Output file

# Variables
my $VCF;                          # Fasta file handler
my $dim=1;                        # Dimension P (based on k-mer size)
my @emission_matrix=();           # 3D array [assembly][row][column]
my $n_proc=0;                     # Number of entries processed
my $n_mem=0;                      # Number of entries stored in RAM
my $n_limit=2000;                 # Limit for number of entries in RAM
my $kmer_size=1;


# Time stamps
my $st_time=0;                    # Start time
my $end_time=0;                   # End time
my $current_time=0;               # Current time
my $elapsed_time=0;               # Time elapsed

```

```
# Hashes
my %vcf_hash=();                # Hash containing sequence info of each sample
my %transition_hash=();         # Hash containing duplets combinations

################################ Main ########################################

# Read in fasta file
$st_time = localtime;
print STDERR "${st_time}: VCF file: ${vcf_file}\n";
# Initialize matrix
$current_time = localtime;
print STDERR "${current_time}: Initializing emission matrix ...\n";
initialize();
# Count frequencies
$current_time = localtime;
print STDERR "${current_time}: Learning emission matrix $n_bases...\n";
fill_emission_matrix();
# Print stuff
$current_time = localtime;
print STDERR "${current_time}: Saving emission matrix in ${outfile}...\n";
print_emission_matrices();

################################ Subs ########################################
## Fill the emission matrix
sub fill_emission_matrix
{
    # Process in chunks of n_limit
 while($n_proc<$n_entries)
    {
        # Read n_lim entries
        read_vcf();
        # Get new entries
        my @entries=keys %vcf_hash;
        # Loop through the entries
        foreach my $entry (@entries)
        {
            # Get variants
            my @ref=@{$vcf_hash{$entry}{'ref'}};
            my @alt=@{$vcf_hash{$entry}{'alt'}};
```

```
92              # Get index form emission_matrix
93              my $row=$transition_hash{@ref[0]};
94              my $col=$transition_hash{@alt[0]};
95              # Fill emission_matrix
96              $emission_matrix[$row][$col]+=1;
97              # Get rid of that entry
98              delete $vcf_hash{$entry};
99              # Update progress
100             $n_mem--;
101             $n_proc++;
102             my $perc=$n_proc/$n_entries*100;
103             printf STDERR "\rCurrent progress: %.2f%", $perc;
104         }
105     }
106     printf STDERR "\n";
107     # Add e(i|i) and scale matrix
108     for(my $i=0;$i<=$dim;$i++)
109     {
110         my $sum=0;
111         $emission_matrix[$i][$i]=$n_bases/4;
112         # Count frequencies per row
113         for(my $j=0;$j<=$dim;$j++)
114         {
115             $sum+=$emission_matrix[$i][$j];
116         }
117         # Normalize each row
118         if($sum!=0)
119         {
120             for(my $j=0;$j<=$dim;$j++)
121             {
122                 $emission_matrix[$i][$j]/=$sum;
123             }
124         }
125     }
126 }
127
128 ## Initialize emission_matrices
129 sub initialize
130 {
```

```
131 # Nucleotides taken into consideration
132 my @nucleotides=('A','C','G','T');
133 for(my $i=1;$i<=$kmer_size;$i++)
134 {
135  $dim*=(scalar @nucleotides);
136 }
137 # Because we start with dim=0 in the loops
138 $dim-=1;
139 # Get all possible combinations of kmer_size nucleotides
140 my @permutations=variations_with_repetition(\@nucleotides,$kmer_size);
141 # Codify numerically each possible permutation
142 my $i=0;
143 foreach my $combination (@permutations)
144 {
145  my $length=$kmer_size-1;
146  my $sequence = join('', @{$combination}[0..$length]);
147  $transition_hash{$sequence}=$i;
148  $i++;
149 }
150 # Initialize emission_matrix
151 for(my $i=0;$i<=$dim;$i++)
152 {
153     for(my $j=0;$j<=$dim;$j++)
154  {
155   $emission_matrix[$i][$j]=0;
156  }
157 }
158}
159
160## Read in vcf file
161sub read_vcf
162{
163    my $entry;
164 while(($n_mem<$n_limit) and (my $line = <STDIN>))
165 {
166  chomp($line);
167  if( $line =~ /#/)
168  {
169            # nothing
```

```
170   }
171   else
172   {
173       my @array = split('\t',$line);
174             my $id = $array[2];
175             my @ref = split(//,$array[3]);
176             my @alt = split(//,$array[4]);
177             if((scalar @ref eq 1) and (scalar @alt eq 1))
178             {
179                 $vcf_hash{$id}{'ref'}=[@ref];
180                 $vcf_hash{$id}{'alt'}=[@alt];
181                 $n_mem++;
182             }
183             else
184             {
185                 $n_entries--;
186             }
187   }
188  }
189 }
190
191 ## Print emission_matrices
192 sub print_emission_matrices
193 {
194     open(OUT, "|gzip -c > ${outfile}")            # $$ is our process id
195         or die "Can't open file '${outfile}' $!";
196  # Reverse hash
197  my %reverse_hash = reverse %transition_hash;
198     # Print to OUT
199  print OUT "\t";
200  for(my $j=0;$j<=$dim;$j++)
201  {
202      my $key = $reverse_hash{$j};
203      print OUT "$key\t";
204  }
205  print OUT "\n";
206  for(my $i=0;$i<=$dim;$i++)
207  {
208      my $key = $reverse_hash{$i};
```

```
209       print OUT "$key\t";
210       for(my $j=0;$j<=$dim;$j++)
211       {
212     printf OUT "%.5f\t", $emission_matrix[$i][$j];
213       }
214       print OUT "\n";
215  }
216      close OUT;
217 }
218 ##############################################################################
```

A.1.3 HiMMe algorithm

```
1 #!/usr/bin/env bash
2 # ----------------------------------------------------------------------------
3 ##The MIT License (MIT)
4 ##
5 ##Copyright (c) 2016 Jordi Abante
6 ##
7 ##Permission is hereby granted, free of charge, to any person obtaining a copy
8 ##of this software and associated documentation files (the "Software"), to deal
9 ##in the Software without restriction, including without limitation the rights
10 ##to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
11 ##copies of the Software, and to permit persons to whom the Software is
12 ##furnished to do so, subject to the following conditions:
13 ##
14 ##The above copyright notice and this permission notice shall be included in all
15 ##copies or substantial portions of the Software.
16 ##
17 ##THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
18 ##IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
19 ##FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
20 ##AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
21 ##LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
22 ##OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
23 ##SOFTWARE.
24 # ----------------------------------------------------------------------------
25 shopt -s extglob
26
```

```
abspath_script="$(readlink -f -e "$0")"
script_absdir="$(dirname "$abspath_script")"
script_name="$(basename "$0" .sh)"

# Find perl scripts
perl_script="${script_absdir}/perl/${script_name}.pl"

if [ $# -eq 0 ]
    then
        cat "$script_absdir/${script_name}_help.txt"
        exit 1
fi

TEMP=$(getopt -o hd:t:k: -l help,outdir:,threads:,kmer_size: -n "$script_name.sh" --
    "$@")

if [ $? -ne 0 ]
then
  echo "Terminating..." >&2
  exit -1
fi

eval set -- "$TEMP"

# Defaults
outdir="$PWD"
threads=2
kmer_size=1

# Options
while true
do
  case "$1" in
    -h|--help)
      cat "$script_absdir"/${script_name}_help.txt
      exit
      ;;
    -d|--outdir)
      outdir="$2"
```

```
65        shift 2
66        ;;
67    -t|--threads)
68        threads="$2"
69        shift 2
70        ;;
71    -k|--kmer_size)
72        kmer_size="$2"
73        shift 2
74        ;;
75    --)
76        shift
77        break
78        ;;
79    *)
80        echo "$script_name.sh: Internal error!"
81        exit -1
82        ;;
83  esac
84 done
85
86 # Inputs
87 tm_file="$1"
88 ep_file="$2"
89 fasta_file="$3"
90
91 # Output
92 fasta_basename="$(basename "$fasta_file")"
93 prefix="${fasta_basename%%.*}"
94 outfile_sum="${outdir}/${prefix}_summary_${kmer_size}.txt"
95 outfile_fasta="${outdir}/${prefix}_corrected_${kmer_size}.fa"
96
97 # Output directory
98 mkdir -p "$outdir"
99
100 # Count number of entries in FASTA
101 n_entries="$(zcat -f "$fasta_file" | grep "^>" | wc -l)"
102
103 # Run
```

```
zcat -f "$fasta_file" | "$perl_script" "$tm_file" "$ep_file" \
    "$fasta_file" "$kmer_size" \
    "$n_entries" "$outfile_sum" "$outfile_fasta"
```

```
#!/usr/bin/env perl
# ------------------------------------------------------------------------------
##The MIT License (MIT)
##
##Copyright (c) 2016 Jordi Abante
##
##Permission is hereby granted, free of charge, to any person obtaining a copy
##of this software and associated documentation files (the "Software"), to deal
##in the Software without restriction, including without limitation the rights
##to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
##copies of the Software, and to permit persons to whom the Software is
##furnished to do so, subject to the following conditions:
##
##The above copyright notice and this permission notice shall be included in all
##copies or substantial portions of the Software.
##
##THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
##IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
##FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
##AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
##LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
##OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
##SOFTWARE.
# ------------------------------------------------------------------------------

# Libraries
use strict;
use Algorithm::Combinatorics qw(combinations variations_with_repetition);
use POSIX;

# Read arguments
my $scriptname = $0;            # Get script name
my $tm_file = @ARGV[0];         # Transition matrix file name
my $ep_file = @ARGV[1];         # Emission probabilities file name
my $fasta_file = @ARGV[2];      # Get target FASTA file name
```

```
36 my $kmer_size= @ARGV[3];          # Get user k-mer size
37 my $n_entries=@ARGV[4];          # Number of entries in the FASTA file
38 my $outfile_sum= @ARGV[5];       # Output file
39 my $outfile_fasta= @ARGV[6];     # Output file
40
41 # For testing purposes
42 my $tm_outfile="out/tm_test.txt.gz";  # File to check the transition matrix
43 my $ep_outfile="out/ep_test.txt.gz";  # File to check the emission probs
44
45 # Handlers
46 my $FASTA;                       # Fasta file handler
47 my $TM;                          # Transition matrix handler
48 my $EP;                          # Emission probabilities handler
49
50 # Variables
51 my $dim=1;                       # Dimension P (based on k-mer size)
52 my @markov_matrix=();            # Transition Matrix [row][column]
53 my @emission_matrix=();          # Emission Matrix [row][column]
54 my $n_proc=0;                    # Number of entries processed
55 my $n_mem=0;                     # Number of entries stored in RAM
56 my $n_limit=250;                  # Limit for number of entries in RAM
57
58 # Time stamps
59 my $st_time=0;                   # Start time
60 my $end_time=0;                  # End time
61 my $current_time=0;              # Current time
62 my $elapsed_time=0;              # Time elapsed
63
64 # Hashes
65 my %fasta_hash=();               # Hash containing sequence info of each sample
66 my %transition_hash=();          # Hash containing transition matrix
67 my %emission_hash=();            # Hash containing emission probs
68 my %emission_all_hash=();        # Hash containing all comb emission probs
69 my %score_hash_1=();             # Hash containing scores iter n-1
70 my %gamma_1=();                  # Recursive variable gamma for opt. path -1
71 my %score_hash_2=();             # Hash containing scores iter n
72 my %gamma=();                    # Recursive variable gamma for opt. path curr.
73 my %results_hash=();             # Hash containing scores of all sequences
74 my %viterbi_out=();              # Hash containing Viterbi's output
```

```
75 my %template_hash=();           # Hash generated based on all the combinations
76
77 ################################## Main ########################################
78
79 ## Read input files
80 $st_time = localtime;
81 print STDERR "${st_time}: ${n_entries} entries to process...\n";
82 print STDERR "${st_time}: Reading in transition matrix file: ${tm_file}...\n";
83 read_tm();
84 #print_markov_matrix();
85 $current_time = localtime;
86 print STDERR "${current_time}: Reading in emission probabilities file: ${ep_file
       }...\n";
87 read_ep();
88 #print_emission_matrix();
89 $current_time = localtime;
90 print STDERR "${current_time}: FASTA file: ${fasta_file}\n";
91
92 ## Initialize state hash and emission all hash
93 $current_time = localtime;
94 print STDERR "${current_time}: Initializing...\n";
95 initialize();
96
97 ## Run algorithm
98 $current_time = localtime;
99 print STDERR "${current_time}: Computing scores & optimal path...\n";
100 run_algorithm();
101
102 # Save stuff
103 $current_time = localtime;
104 print STDERR "${current_time}: Saving in ${outfile_sum} & ${outfile_fasta}...\n";
105 save_results();
106
107 ################################## Subs ########################################
108 ## Run forward and Viterbi algorithms
109 sub run_algorithm
110 {
111     # Process in chunks of n_limit
112  while($n_proc<$n_entries)
```

```
113     {
114         # Read n_lim entries
115         read_fasta();
116         # Get chromosomes from fasta
117         my @entries=keys %fasta_hash;
118         foreach my $entry (@entries)
119         {
120             my $n_iter=floor((scalar @{$fasta_hash{$entry}})/$kmer_size)-1;
121             my $pos=1;
122             # Loop through the sequence
123             for(my $i=0;$i<=$n_iter*$kmer_size;$i+=$kmer_size)
124             {
125                 # Get states in the iteration
126                 my $seq_1=@{$fasta_hash{$entry}}[$i];
127                 my $seq_2=@{$fasta_hash{$entry}}[$i+$kmer_size];
128                 for(my $j=1;$j<$kmer_size;$j++)
129                 {
130                     $seq_1=$seq_1.@{$fasta_hash{$entry}}[$i+$j];
131                     $seq_2=$seq_2.@{$fasta_hash{$entry}}[$i+$kmer_size+$j];
132                 }
133                 # Get all possible hidden states
134                 %score_hash_2=%template_hash;
135                 # In case it's the first iteration, initialize alpha
136                 if($i eq 0)
137                 {
138                     # Get all possible hidden states
139                     %score_hash_1=%template_hash;
140                     # Compute score for each hidden state
141                     foreach my $hidden_state (keys %score_hash_1)
142                     {
143                         my $pi_0=1/(4**$kmer_size);
144                         my $emission=$emission_all_hash{$hidden_state}{$seq_1};
145                         my $score=$pi_0*$emission;
146                         $score_hash_1{$hidden_state}=$score;
147                         $gamma{$pos}{$hidden_state}{score}=$score;
148                         $gamma{$pos}{$hidden_state}{seq}="";
149                     }
150                     $pos++;
151                 }
```

```
152                 if(length($seq_2) eq $kmer_size)
153                 {
154                     # Consider every possible hidden state
155                     foreach my $hidden_state_2 (keys %score_hash_2)
156                     {
157                         my $emission=$emission_all_hash{$hidden_state_2}{$seq_2};
158                         my $sum=0;
159                         foreach my $hidden_state_1 (keys %score_hash_1)
160                         {
161                             # Forward algorithm
162                             my $row=$transition_hash{$hidden_state_1};
163                             my $col=$transition_hash{$hidden_state_2};
164                             my $score=$score_hash_1{$hidden_state_1}*
165                                 $markov_matrix[$row][$col]*$emission;
166                             $sum+=$score;
167                             # Viterbi algorithm
168                             my $score=$gamma{$pos-1}{$hidden_state_1}{score}*
169                                 $markov_matrix[$row][$col]*$emission;
170                             if($score > $gamma{$pos}{$hidden_state_2}{score})
171                             {
172                                 $gamma{$pos}{$hidden_state_2}{score}=$score;
173                                 $gamma{$pos}{$hidden_state_2}{seq}=$hidden_state_1;
174                             }
175                         }
176                         $score_hash_2{$hidden_state_2}=$sum;
177                     }
178                     $pos++;
179                 }
180                 else
181                 {
182                     %score_hash_2=%score_hash_1;
183                 }
184                 %score_hash_1=%score_hash_2;
185             }
186             ## Traceback gamma
187             $pos--;
188             foreach my $hidden_state (keys %{$gamma{$pos}})
189             {
190                 if($gamma{$pos}{$hidden_state}{score} > $viterbi_out{$entry}{$pos}{
```

```
    score})
191                 {
192                     $viterbi_out{$entry}{$pos}{score}=$gamma{$pos}{$hidden_state}{
    score};
193                     $viterbi_out{$entry}{$pos}{seq}=$hidden_state;
194                 }
195             }
196             $pos--;
197             for(my $i=$pos;$i>=1;$i--)
198             {
199                 my $seq=$viterbi_out{$entry}{$i+1}{seq};
200                 my $seq_1=$gamma{$i+1}{$seq}{seq};
201                 $viterbi_out{$entry}{$i}{seq}=$seq_1;
202             }
203             %gamma=();
204             # Total score P(X=x|HMM)
205             my $total_score=0;
206             foreach my $kmer (keys %score_hash_1)
207             {
208                 $total_score+=$score_hash_1{$kmer};
209             }
210             # Store in results hash
211             $results_hash{$entry}=$total_score;
212             # CLean score hash
213             %score_hash_1=();
214             %score_hash_2=();
215             # Get rid of that entry
216             delete $fasta_hash{$entry};
217             $n_mem--;
218             # Update progress
219             $n_proc++;
220             my $perc=$n_proc/$n_entries*100;
221             printf STDERR "\rCurrent progress: %.2f%", $perc;
222         }
223     }
224     print STDERR "\n";
225 }
226 
227 ## Initialize stuff
```

```
sub initialize
{
    # Get possible permutations of lenght k
    my @nucleotides=('A','C','T','G');
    my @permutations=variations_with_repetition(\@nucleotides,$kmer_size);
    # Get all possible hidden states
    foreach my $string_1 (@permutations)
    {
        my $length=$kmer_size-1;
        my $sequence_1 = join('', @{$string_1}[0..$length]);
        $template_hash{$sequence_1}=1;
        foreach my $string_2 (@permutations)
        {
            my $length=$kmer_size-1;
            my $sequence_2 = join('', @{$string_2}[0..$length]);
            my $emission=1;
            for(my $j=0;$j<$kmer_size;$j++)
            {
                my @state_1=split(//,$sequence_1);
                my @state_2=split(//,$sequence_2);
                my $row=$emission_hash{$state_1[$j]};
                my $col=$emission_hash{$state_2[$j]};
                $emission*=$emission_matrix[$row][$col];
            }
            $emission_all_hash{$sequence_1}{$sequence_2}=$emission;
        }
    }
}
## Save results
sub save_results
{
    my $sum=0;
    my $n=0;
    # Open output file
    open(OUT,">$outfile_sum") or die "Can't open file '${outfile_sum}' $!";
    # Loop through all sequences
    foreach my $key (sort keys %results_hash)
    {
        my $string = sprintf('%.3e', $results_hash{$key});
```

```
267         printf OUT "$key\t$string";
268         print OUT "\n";
269         $sum+=$results_hash{$key};
270         $n++;
271     }
272     my $mean=$sum/$n;
273     my $string = sprintf('%.3e', $mean);
274     print OUT "Mean\t$string\n";
275     # Close handler
276     close OUT;
277     # Print corrected sequences
278     open(OUT,">$outfile_fasta") or die "Can't open file '${outfile_fasta}' $!";
279     foreach my $entry (sort keys %results_hash)
280     {
281         print OUT ">${entry}\n";
282         foreach my $pos (sort {$a<=>$b} keys %{$viterbi_out{$entry}})
283         {
284             print OUT "$viterbi_out{$entry}{$pos}{seq}";
285         }
286         print OUT "\n";
287     }
288     close OUT;
289 }
290
291 ## Read in fasta file
292 sub read_fasta
293 {
294     my $entry;
295     $n_mem=0;
296  while((my $line = <STDIN>) and ($n_mem<$n_limit))
297  {
298   chomp($line);
299   if( $line =~ />/)
300   {
301             $entry=substr($line,1); # Get rid of leading ">" character
302   }
303   else
304   {
305       my @array = split //, $line;
```

```
306         push @{$fasta_hash{$entry}},@array;
307                 $n_mem++;
308   }
309  }
310 }
311
312 ## Print fasta
313 sub print_fasta
314 {
315  # Print output
316  foreach my $key (sort keys %fasta_hash)
317  {
318   my @entry=@{$fasta_hash{$key}};
319   print ">$key\n";
320   foreach my $nuc (@entry)
321   {
322       print "$nuc";
323   }
324   print "\n";
325  }
326 }
327 ## Read in transition matrix
328 sub read_tm
329 {
330     my @nucleotides=('A','C','T','G');
331  for(my $i=1;$i<=$kmer_size;$i++)
332  {
333   $dim*=(scalar @nucleotides);
334  }
335  # Because we start with dim=0 in the loops
336  $dim-=1;
337     # Open file
338     open($TM, "gunzip -c ${tm_file} | ")
339         or die "Can't open file '${tm_file}' $!";
340     my $i=0;
341     # Loop through each row
342     while(my $line=<$TM>)
343     {
344         if($i gt 0)
```

```
345         {
346             chomp($line);
347             my @row=split(/\t/,$line);
348             my $j=0;
349             foreach my $col (@row)
350             {
351                 if($j eq 0)
352                 {
353                     $transition_hash{$col}=$i-1;
354                 }
355                 else
356                 {
357                     $markov_matrix[$i-1][$j-1]=$col;
358                 }
359                 $j++;
360             }
361         }
362         $i++;
363     }
364     close $TM;
365 }
366 
367 ## Print transition matrix
368 sub print_markov_matrix
369 {
370     open(OUT, "|gzip -c > ${tm_outfile}")            # $$ is our process id
371         or die "Can't open file '${tm_outfile}' $!";
372  # Reverse hash
373  my %reverse_hash = reverse %transition_hash;
374     # Print to OUT
375  print OUT "\t";
376  for(my $j=0;$j<=$dim;$j++)
377  {
378      my $key = $reverse_hash{$j};
379      print OUT "$key\t";
380  }
381  print OUT "\n";
382  for(my $i=0;$i<=$dim;$i++)
383  {
```

```
384     my $key = $reverse_hash{$i};
385     print OUT "$key\t";
386     for(my $j=0;$j<=$dim;$j++)
387     {
388   printf OUT "%.5f\t", $markov_matrix[$i][$j];
389     }
390     print OUT "\n";
391 }
392    close OUT;
393}
394
395## Read in emission probabilities
396sub read_ep
397{
398    # Open file
399    open($EP, "gunzip -c ${ep_file} | ")
400        or die "Can't open file '${ep_file}' $!";
401    my $i=0;
402    # Loop through each row
403    while(my $line=<$EP>)
404    {
405        if($i gt 0)
406        {
407            chomp($line);
408            my @row=split(/\t/,$line);
409            my $j=0;
410            foreach my $col (@row)
411            {
412                if($j eq 0)
413                {
414                    $emission_hash{$col}=$i-1;
415                }
416                else
417                {
418                    $emission_matrix[$i-1][$j-1]=$col;
419                }
420                $j++;
421            }
422        }
```

```
423         $i++;
424     }
425     close $EP;
426 }
427 
428 ## Print emission probabilities
429 sub print_emission_matrix
430 {
431     open(OUT, "|gzip -c > ${ep_outfile}")            # $$ is our process id
432         or die "Can't open file '${ep_outfile}' $!";
433  # Reverse hash
434  my %reverse_hash = reverse %emission_hash;
435     # Print to OUT
436  print OUT "\t";
437  for(my $j=0;$j<=3;$j++)
438  {
439      my $key = $reverse_hash{$j};
440      print OUT "$key\t";
441  }
442  print OUT "\n";
443  for(my $i=0;$i<=3;$i++)
444  {
445      my $key = $reverse_hash{$i};
446      print OUT "$key\t";
447      for(my $j=0;$j<=3;$j++)
448      {
449   printf OUT "%.5f\t", $emission_matrix[$i][$j];
450      }
451      print OUT "\n";
452  }
453     close OUT;
454 }
455 
456 ################################################################################
```

A.1.4 Generation of random sequences

```
1 #!/usr/bin/env bash
2 shopt -s extglob
```

```
3
4 abspath_script="$(readlink -f -e "$0")"
5 script_absdir="$(dirname "$abspath_script")"
6 script_name="$(basename "$0" .sh)"
7
8 TEMP=$(getopt -o hl:r -l help,length:,rna -n "$script_name.sh" -- "$@")
9
10 if [ $? -ne 0 ]
11 then
12   echo "Terminating..." >&2
13   exit -1
14 fi
15
16 eval set -- "$TEMP"
17
18 # Defaults
19 length=10
20 bases="ACTG"
21
22 while true
23 do
24   case "$1" in
25     -h|--help)
26       cat "$script_absdir"/${script_name}_help.txt
27       exit
28       ;;
29     -l|--length)
30       length="$2"
31       shift 2
32       ;;
33     -r|--rna)
34       bases="ACUG"
35       shift
36       ;;
37     --)
38       shift
39       break
40       ;;
41     *)
```

```
      echo "$script_name.sh: Internal error!"
      exit -1
      ;;
  esac
done

# Run
cat /dev/urandom | tr -dc "$bases" | fold -w "$length" | head -n 1
```

```
#!/usr/bin/env bash
shopt -s extglob

abspath_script="$(readlink -f -e "$0")"
script_absdir="$(dirname "$abspath_script")"
script_name="$(basename "$0" .sh)"

if [ $# -eq 0 ]
    then
        cat "$script_absdir/${script_name}_help.txt"
        exit 1
fi

TEMP=$(getopt -o hd:t:c:l:p: -l help,outdir:,threads:,chr:,length:,prefix: -n "
    $script_name.sh" -- "$@")

if [ $? -ne 0 ]
then
  echo "Terminating..." >&2
  exit -1
fi

eval set -- "$TEMP"

# Defaults
outdir="$PWD"
chr=5
length=2000
prefix="simulated_genome"
bases="ACTG"
```

```
threads=2

# Options
while true
do
  case "$1" in
    -h|--help)
      cat "$script_absdir"/${script_name}_help.txt
      exit
      ;;
    -d|--outdir)
      outdir="$2"
      shift 2
      ;;
    -t|--threads)
      threads="$2"
      shift 2
      ;;
    -c|--chr)
      chr="$2"
      shift 2
      ;;
    -l|--length)
      length="$2"
      shift 2
      ;;
    -p|--prefix)
      prefix="$2"
      shift 2
      ;;
    --)
      shift
      break
      ;;
    *)
      echo "$script_name.sh:Internal error!"
      exit -1
      ;;
  esac
```

```
done

# Start time
start_time="$(date +"%s%3N")"

# Temp and output
outfile="${outdir}/${prefix}.fa"
tempfile="${outdir}/${prefix}"
export outfile
export tempfile
export length

# Outdir
mkdir -p "$outdir"

# Generate a file for each chromosome
seq 1 "$chr" | xargs -I {} --max-proc "$threads" bash -c \
    'echo ">random_seq_{}" >> '${tempfile}_{}.tmp' && random_sequence_generator.sh -
    l '$length' >> '${tempfile}_{}.tmp''

# Concatenate all chromosomes and filter
seq 1 "$chr" | xargs -I {} --max-proc 1 bash -c \
    'cat '${tempfile}_{}.tmp' >> '$outfile''

# Remove temp file
rm -f ${tempfile}*tmp*

# Time elapsed
end_time="$(date +"%s%3N")"
echo "Time elapsed: $(( $end_time - $start_time )) ms"
```

A.1.5 Scoring results analysis

```
### t-test
## 1-mers
read.table('random_seq_10_summary_1.txt')->r_10_1
read.table('random_seq_20_summary_1.txt')->r_20_1
read.table('random_seq_30_summary_1.txt')->r_30_1
read.table('random_seq_40_summary_1.txt')->r_40_1
```

```
read.table('random_seq_50_summary_1.txt')->r_50_1
read.table('random_seq_60_summary_1.txt')->r_60_1

read.table('true_seq_10_summary_1.txt')->t_10_1
read.table('true_seq_20_summary_1.txt')->t_20_1
read.table('true_seq_30_summary_1.txt')->t_30_1
read.table('true_seq_40_summary_1.txt')->t_40_1
read.table('true_seq_50_summary_1.txt')->t_50_1
read.table('true_seq_60_summary_1.txt')->t_60_1

stats_10_1=t.test(r_10_1[1:250,2],t_10_1[1:250,2],alternative='less')
stats_20_1=t.test(r_20_1[1:250,2],t_20_1[1:250,2],alternative='less')
stats_30_1=t.test(r_30_1[1:250,2],t_30_1[1:250,2],alternative='less')
stats_40_1=t.test(r_40_1[1:250,2],t_40_1[1:250,2],alternative='less')
stats_50_1=t.test(r_50_1[1:250,2],t_50_1[1:250,2],alternative='less')
stats_60_1=t.test(r_60_1[1:250,2],t_60_1[1:250,2],alternative='less')

## 3-mers
read.table('random_seq_10_summary_3.txt')->r_10_3
read.table('random_seq_20_summary_3.txt')->r_20_3
read.table('random_seq_30_summary_3.txt')->r_30_3
read.table('random_seq_40_summary_3.txt')->r_40_3
read.table('random_seq_50_summary_3.txt')->r_50_3
read.table('random_seq_60_summary_3.txt')->r_60_3

read.table('true_seq_10_summary_3.txt')->t_10_3
read.table('true_seq_20_summary_3.txt')->t_20_3
read.table('true_seq_30_summary_3.txt')->t_30_3
read.table('true_seq_40_summary_3.txt')->t_40_3
read.table('true_seq_50_summary_3.txt')->t_50_3
read.table('true_seq_60_summary_3.txt')->t_60_3

stats_10_3=t.test(r_10_3[1:250,2],t_10_3[1:250,2],alternative='less')
stats_20_3=t.test(r_20_3[1:250,2],t_20_3[1:250,2],alternative='less')
stats_30_3=t.test(r_30_3[1:250,2],t_30_3[1:250,2],alternative='less')
stats_40_3=t.test(r_40_3[1:250,2],t_40_3[1:250,2],alternative='less')
stats_50_3=t.test(r_50_3[1:250,2],t_50_3[1:250,2],alternative='less')
stats_60_3=t.test(r_60_3[1:250,2],t_60_3[1:250,2],alternative='less')

```

```
## 5-mers
read.table('random_seq_10_summary_5.txt')->r_10_5
read.table('random_seq_20_summary_5.txt')->r_20_5
read.table('random_seq_30_summary_5.txt')->r_30_5
read.table('random_seq_40_summary_5.txt')->r_40_5
read.table('random_seq_50_summary_5.txt')->r_50_5
read.table('random_seq_60_summary_5.txt')->r_60_5

read.table('true_seq_10_summary_5.txt')->t_10_5
read.table('true_seq_20_summary_5.txt')->t_20_5
read.table('true_seq_30_summary_5.txt')->t_30_5
read.table('true_seq_40_summary_5.txt')->t_40_5
read.table('true_seq_50_summary_5.txt')->t_50_5
read.table('true_seq_60_summary_5.txt')->t_60_5

stats_10_5=t.test(r_10_5[1:250,2],t_10_5[1:250,2],alternative='less')
stats_20_5=t.test(r_20_5[1:250,2],t_20_5[1:250,2],alternative='less')
stats_30_5=t.test(r_30_5[1:250,2],t_30_5[1:250,2],alternative='less')
stats_40_5=t.test(r_40_5[1:250,2],t_40_5[1:250,2],alternative='less')
stats_50_5=t.test(r_50_5[1:250,2],t_50_5[1:250,2],alternative='less')
stats_60_5=t.test(r_60_5[1:250,2],t_60_5[1:250,2],alternative='less')

### Summary
out=as.data.frame(cbind(c(stats_10_1$p.value,stats_20_1$p.value,
                          stats_30_1$p.value,stats_40_1$p.value,
                          stats_50_1$p.value,stats_60_1$p.value),
                        c(stats_10_3$p.value,stats_20_3$p.value,
                          stats_30_3$p.value,stats_40_3$p.value,
                          stats_50_3$p.value,stats_60_3$p.value),
                        c(stats_10_5$p.value,stats_20_5$p.value,
                          stats_30_5$p.value,stats_40_5$p.value,
                          stats_50_5$p.value,stats_60_5$p.value)))
colnames(out)=c('1-mers','3-mers','5-mers')
write.table(out,file='pvalues.txt',quote=F,sep='\t',row.names=F)

### Plots
read.table('scores_1.txt')->s1
read.table('scores_3.txt')->s3
read.table('scores_5.txt')->s5
```

```
colnames(s1)=c('random','true')
colnames(s3)=c('random','true')
colnames(s5)=c('random','true')

x=seq(10,100,10)

# Ratios
s1_ratios=log(s1[,2]/s1[,1])
s3_ratios=log(s3[,2]/s3[,1])
s5_ratios=log(s5[,2]/s5[,1])

# Plot
pdf("scoring.pdf")
par(mar=c(5, 5, 1, 1))
plot(x,s5_ratios,pch=16,type="b",col="blue",xlab="Sequences length",
     ylab=expression(log(s[true]/s[random])))
lines(x,s3_ratios,pch=15,type="b",col="red",xlab="",ylab="")
lines(x,s1_ratios,pch=14,type="b",col="green",xlab="",ylab="")
legend("topleft",legend=c("5-mers","3-mers","1-mers"),
                 text.col=c("blue","red","green"),
                 pch=c(16,15,14),
                 col=c("blue","red","green"))

dev.off()
```

A.1.6 SEQC results analysis

```
#!/usr/bin/env Rscript

# Read in seqc scores
read.table('seqc_scores.txt',sep=" ")->seqc

# Random scores for sequence length 100
random=c(5.442e-61,2.21893034881624e-60,2.254e-61)

ratios_1=cbind(log(seqc[,1]/random[1]),rep(1,6))
ratios_3=cbind(log(seqc[,2]/random[2]),rep(3,6))
ratios_5=cbind(log(seqc[,3]/random[3]),rep(5,6))

```

```
seqc_df=as.data.frame(rbind(ratios_1,ratios_3,ratios_5))
colnames(seqc_df)=c("Ratio","kmer")
# Plot
pdf("seqc_scoring_1.pdf")
par(mar=c(5, 5, 1, 1))
boxplot(Ratio ~ kmer,data=seqc_df,lwd = 2, col='blue',ylim=c(0,30),
        xlab="k-mer size",ylab = expression(log(s[SEQC]/s[random])))
dev.off()
```

A.1.7 Viterbi results analysis

```
# Before
read.table('before/random_seq_10_summary_1.txt')->b_10_1
read.table('before/random_seq_10_summary_3.txt')->b_10_3
read.table('before/random_seq_10_summary_5.txt')->b_10_5
read.table('before/random_seq_20_summary_1.txt')->b_20_1
read.table('before/random_seq_20_summary_3.txt')->b_20_3
read.table('before/random_seq_20_summary_5.txt')->b_20_5
read.table('before/random_seq_30_summary_1.txt')->b_30_1
read.table('before/random_seq_30_summary_3.txt')->b_30_3
read.table('before/random_seq_30_summary_5.txt')->b_30_5
read.table('before/random_seq_40_summary_1.txt')->b_40_1
read.table('before/random_seq_40_summary_3.txt')->b_40_3
read.table('before/random_seq_40_summary_5.txt')->b_40_5
read.table('before/random_seq_50_summary_1.txt')->b_50_1
read.table('before/random_seq_50_summary_3.txt')->b_50_3
read.table('before/random_seq_50_summary_5.txt')->b_50_5
read.table('before/random_seq_60_summary_1.txt')->b_60_1
read.table('before/random_seq_60_summary_3.txt')->b_60_3
read.table('before/random_seq_60_summary_5.txt')->b_60_5

# After
read.table('after/random_seq_10_corrected_1_summary_1.txt')->a_10_1
read.table('after/random_seq_10_corrected_3_summary_3.txt')->a_10_3
read.table('after/random_seq_10_corrected_5_summary_5.txt')->a_10_5
read.table('after/random_seq_20_corrected_1_summary_1.txt')->a_20_1
read.table('after/random_seq_20_corrected_3_summary_3.txt')->a_20_3
read.table('after/random_seq_20_corrected_5_summary_5.txt')->a_20_5
read.table('after/random_seq_30_corrected_1_summary_1.txt')->a_30_1
```

```
read.table('after/random_seq_30_corrected_3_summary_3.txt')->a_30_3
read.table('after/random_seq_30_corrected_5_summary_5.txt')->a_30_5
read.table('after/random_seq_40_corrected_1_summary_1.txt')->a_40_1
read.table('after/random_seq_40_corrected_3_summary_3.txt')->a_40_3
read.table('after/random_seq_40_corrected_5_summary_5.txt')->a_40_5
read.table('after/random_seq_50_corrected_1_summary_1.txt')->a_50_1
read.table('after/random_seq_50_corrected_3_summary_3.txt')->a_50_3
read.table('after/random_seq_50_corrected_5_summary_5.txt')->a_50_5
read.table('after/random_seq_60_corrected_1_summary_1.txt')->a_60_1
read.table('after/random_seq_60_corrected_3_summary_3.txt')->a_60_3
read.table('after/random_seq_60_corrected_5_summary_5.txt')->a_60_5

# Difference means
diff_1=c(a_10_1[nrow(a_10_1),1]-b_10_1[nrow(b_10_1),1],
         a_20_1[nrow(a_20_1),1]-b_20_1[nrow(b_20_1),1],
         a_30_1[nrow(a_30_1),1]-b_30_1[nrow(b_30_1),1],
         a_40_1[nrow(a_40_1),1]-b_40_1[nrow(b_40_1),1],
         a_50_1[nrow(a_50_1),1]-b_50_1[nrow(b_50_1),1],
         a_60_1[nrow(a_60_1),1]-b_60_1[nrow(b_60_1),1])
diff_3=c(a_10_3[nrow(a_10_3),1]-b_10_3[nrow(b_10_3),1],
         a_20_3[nrow(a_20_3),1]-b_20_3[nrow(b_20_3),1],
         a_30_3[nrow(a_30_3),1]-b_30_3[nrow(b_30_3),1],
         a_40_3[nrow(a_40_3),1]-b_40_3[nrow(b_40_3),1],
         a_50_3[nrow(a_50_3),1]-b_50_3[nrow(b_50_3),1],
         a_60_3[nrow(a_60_3),1]-b_60_3[nrow(b_60_3),1])
diff_5=c(a_10_5[nrow(a_10_5),1]-b_10_5[nrow(b_10_5),1],
         a_20_5[nrow(a_20_5),1]-b_20_5[nrow(b_20_5),1],
         a_30_5[nrow(a_30_5),1]-b_30_5[nrow(b_30_5),1],
         a_40_5[nrow(a_40_5),1]-b_40_5[nrow(b_40_5),1],
         a_50_5[nrow(a_50_5),1]-b_50_5[nrow(b_50_5),1],
         a_60_5[nrow(a_60_5),1]-b_60_5[nrow(b_60_5),1])
# Ratios means
ratio_1=c(a_10_1[nrow(a_10_1),1]/b_10_1[nrow(b_10_1),1],
          a_20_1[nrow(a_20_1),1]/b_20_1[nrow(b_20_1),1],
          a_30_1[nrow(a_30_1),1]/b_30_1[nrow(b_30_1),1],
          a_40_1[nrow(a_40_1),1]/b_40_1[nrow(b_40_1),1],
          a_50_1[nrow(a_50_1),1]/b_50_1[nrow(b_50_1),1],
          a_60_1[nrow(a_60_1),1]/b_60_1[nrow(b_60_1),1])
ratio_3=c(a_10_3[nrow(a_10_3),1]/b_10_3[nrow(b_10_3),1],
```

```
            a_20_3[nrow(a_20_3),1]/b_20_3[nrow(b_20_3),1],
            a_30_3[nrow(a_30_3),1]/b_30_3[nrow(b_30_3),1],
            a_40_3[nrow(a_40_3),1]/b_40_3[nrow(b_40_3),1],
            a_50_3[nrow(a_50_3),1]/b_50_3[nrow(b_50_3),1],
            a_60_3[nrow(a_60_3),1]/b_60_3[nrow(b_60_3),1])
ratio_5=c(a_10_5[nrow(a_10_5),1]/b_10_5[nrow(b_10_5),1],
            a_20_5[nrow(a_20_5),1]/b_20_5[nrow(b_20_5),1],
            a_30_5[nrow(a_30_5),1]/b_30_5[nrow(b_30_5),1],
            a_40_5[nrow(a_40_5),1]/b_40_5[nrow(b_40_5),1],
            a_50_5[nrow(a_50_5),1]/b_50_5[nrow(b_50_5),1],
            a_60_5[nrow(a_60_5),1]/b_60_5[nrow(b_60_5),1])
# Plot
pdf("viterbi.pdf")
x=seq(10,60,10)
par(mar=c(5, 5, 1, 1))
plot(x,ratio_5,pch=16,type="b",col="blue",xlab="Sequences length",
            ylab=expression(s[corrected]/s[random]))
lines(x,ratio_3,pch=15,type="b",col="red",xlab="",ylab="")
lines(x,ratio_1,pch=14,type="b",col="green",xlab="",ylab="")
legend("topleft",legend=c("5-mers","3-mers","1-mers"),
                          text.col=c("blue","red","green"),pch=c(16,15,14),col=c("blue"
    ,"red","green"))

dev.off()
```