Università di Pisa

Master Programme in Computer Science

Fast Packing of Large Genomic Data

Supervisors:
Prof. *Filippo Geraci*
Dott.ssa *Veronica Guerrini*

Candidate:
Gemma Martini

Accademic Year 2021–2022

Un grazie di cuore
a tutti coloro che
mi hanno supportato
in questo periodo
decisivo.

Contents

Introduction

The advent and the subsequent continuous refinement of genome sequencing technologies[3][40][42] have made it possible to have a large sampling of many living organisms, but at the same time have raised the necessity for introducing mechanisms of compression, or at least encoding, able to allow effective storage and transfer of huge collections of genomics data.

In 2010, the 1000 *Genomes Project*[34][35][2] led to the acquisition of the genome of 1000 individuals belonging to the species homo sapiens and from there the available data[13] have increased with exponential speed, due to the combination of significantly lower cost and increased speed of sequencing.

The target for the cost of 1000$ for accurate sequencing a human genome will be matched shortly, allowing researchers and clinitians to perform sequencing more and more often. Unfortunately, the pace at which storage and communication resources are evolving is not enough, and the genomic data centers are being flooded with data; it is the so-called “data deluge”[4].

The Sequence Read Archive(SRA)[i][26][21], which houses a large portion of the World’s public sequencing data, is rapidly expanding (Figure 1) and presently comprises 62 petabases of DNA and RNA sequences.

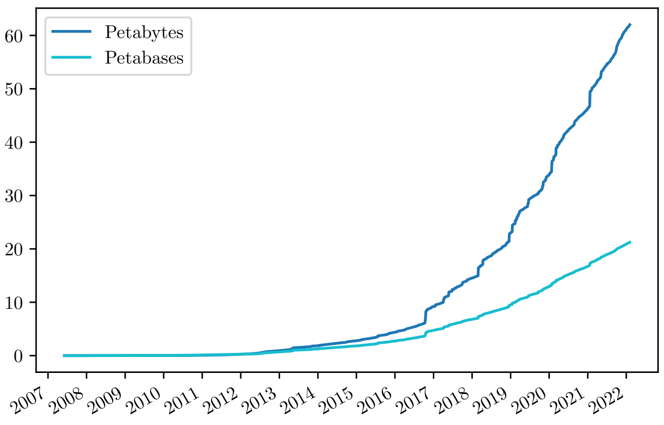


Figure 1: Sequence Read Archive (SRA) growth over the years.

[i]For a quick overview see https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?

This is a problem for both data storage and transmission, preventing team collaboration and long-term data storage, which is required for reproducibility of published results.

Data centers are frequently employed as a solution, but they come at a high expense in terms of storage capacity and transmission bandwidth.

The solution commonly adopted for ease of use and processing speed is to employ general-purpose compressors, such as GZip or Bzip2. This approach, however, does not take into account the structure of the data coming from **N**ext **G**eneration **S**equencing and therefore does not achieve particularly effective results for compressing genomic data. It is in this context that it becomes crucially important to employ research effort in designing tools that can leverage on the features of NGS data to achieve the best performance.

The standard output produced by NGS sequencing machines consists into ASCII-based text files, called FASTQ[16], which can contain up to millions of records each of which represented by 4 lines: one header identifying the record, the second containing the nucleotide sequence, the third acting as a separator, and the fourth representing the confidence with which the individual bases were called by the machinery. Each record is called *read*.

Another common format used for representing genomics sequences is the FASTA file (or, more commonly multiFASTA file), that is ASCII-based file that contains 2 elements for each record: one with an identifier and one with the nucleotide sequence.

In uncompressed form, storing raw, whole-genome, high-coverage sequencing data for a single human can easily surpass $200GB$. The majority of DNA sequencing analysis (for example, in the context of precision medicine) relies on comparing the variations of the sequenced genomes to a known reference genome, to which the raw reads in the FASTQ(A) file are aligned in order to assess these variations.

So far we have explained how to store the entire genome of an individual as a set of reads, but this is not the only way. The sequences of nucleotides identified by the sequencing machinery can be *assembled*, through the process named *assembly*: reads are aligned in order to form the whole genome of an individual. In order to assemble DNA or RNA, every single read needs to be placed in the correct position with respect to the whole helix and this procedure allows the assembled file (in FASTA format) not to store the quality score.

As already presented, these files are commonly compressed with GZIP, a general-purpose compression program that is rather fast and widely used but does not take advantage of file-specific features.

Fast **P**acking of **L**arge **Ge**nomic **D**ata[ii] is the proposed solution to this problem that has been reflected into a tool for *encoding* genomic files, such as FASTQ and FASTA, and explaining its functioning it the main objective of this thesis.

This manuscript is divided into 5 chapters, preceded by a brief warm up (Chapter 1), necessary to understand in detail what is exposed. More in detail, we will discuss topics such as the sequencing

[ii]The source code of the tool is freely available at https://github.com/magemma/FPLGeD_tool

of DNA fragments, we will see what are the most commonly used techniques for this purpose in the past and today, and we will discover what are the basic steps to convert an ensemble of reads into the assembly of the entire genome of an individual.

In the central part of the first chapter (Section 1.2), we will present what has been proposed in the state of the art as a solution to the problem of reducing the space occupied in storing nucleotide sequences. In particular, we will see that compression methods belong to 4 categories and highlight their pros and cons, and then move on to a definition of the data we aim to compress thanks to FGPLGeD (Section 1.3), that are FASTQ(A) files.

The core part of this dissertation begins with Chapter 2, where we present our approach to compressing FASTQ(A) files, through 3 different lossless algorithms, one for each row of the FASTQ file. One may ask why only three algorithms if each sequence stored in a FASTQ file is decoupled into 4 rows and the reason is that the third row (*plus line*) is often a copy of the *header*. For FASTA files, only two algorithms out of the three are used, since these files do not contain information about the *per-base quality* of the stored sequences.

After detailing the algorithms employed for compression, we present in Chapter 3 an experimental evaluation of the performance of FPLGeD on a dataset representing different sequencing scenarios and strategies. For the sake of explanation, Chapter 3 is divided into two parts, the first one describes the data used to test our tool and compares it with the state of the art, the second one shows the significant improvement made by FPLGeD in the compression of nucleotide sequences stored in both FASTQ and FASTA format. In fact, our tool proves to be faster than its competitors and also more performing in terms of compression ratio on long reads generated by third generation sequencing, which is, in our view, the technology that will be more present in the near future.

Conclusions in Chapter 4 summarize the achievements of FPLGeD and discuss new directions about future work on this topic.

Chapter 1

Background

In order to compare algorithms and describe our FPLGeD, we need to introduce some terminology. Usually, the input of a compression algorithm is a *sequence* of *symbols* from a given *alphabet*. In the case of the compression of DNA/RNA sequences, the alphabet is made of characters that come from the four possible *nucleotides* (A, T/U, G, C) (Where thymine ‘T’ is replaced by uracil ‘U’ in RNA sequences).

Definition 1 (Nucleotide)**.** *A **nucleotide** is one of DNA and RNA’s structural components, or building blocks. A nucleotide is made up of a base (one of the chemicals: adenine-A, thymine-T, uracil-U, guanine-G, or cytosine-C), as well as a sugar molecule and a phosphoric acid molecule. Thymine and Uracil are alternative to each other. The nucleotides belong to two groups: the* pyrimidines *C, T, and U each have a single nitrogen-containing ring; and the* purines *A and G with two nitrogen-containing rings.*

1.1 Sequencing and Assembling

The procedure of determining the order of nucleotides (Definition 1) in a strand of DNA is known as *DNA sequencing*. Single-stranded DNA bases form pairings with the bases on the other helix in double-stranded DNA. Hydrogen bonds develop between complimentary bases, resulting in this pairing.

Basically, sequencing technologies allow a DNA sequence to be transformed into a digital file.

The Sanger technique[38] was introduced in 1977 and is regarded as a *first generation sequencing* technology (FGS). It was able to read fragments with a maximum length of 1000 nucleotides, at a quite high cost and with high precision: for example, the first human genome sequencing took 15 years and \$100 millions to sequence human DNA with this technique[22].

Until the introduction of the so called high-throughput next generation sequencing, that opened new perspectives for genome exploration and analysis, the only sequencing technology available to biologists was Sanger’s.

Roche's 454 technology[29] introduced *second generation technologies* (SGS) in 2005, and they were commercialized as technologies capable of creating sequences at a very high throughput and at a significantly lower cost.

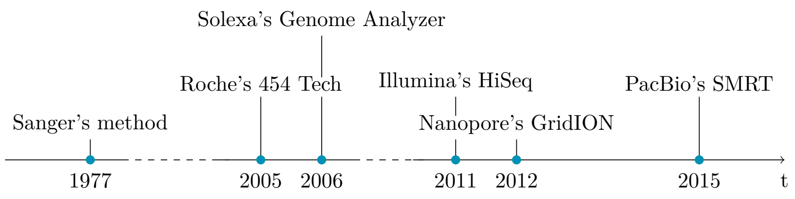


Figure 1.1: A timeline of the major sequencing technologies.

Solexa's Genome Analyzer was launched as a competitor against Roche's technology in 2006. In less than a year, Illumina (a company involved in gene and protein analysis) acquired Solexa and gave birth to SGSs, such as HiSeq and MiSeq.

Nowadays, Illumina's HiSeq 2500/3000/4000 are the most commonly used sequencing techniques, due to their high reliability and low cost.

Let us describe the functioning of Illumina's sequencing machinery.
This sequencing process consists of three phases:

- Sample preparation;
- Cluster generation;
- Sequencing.

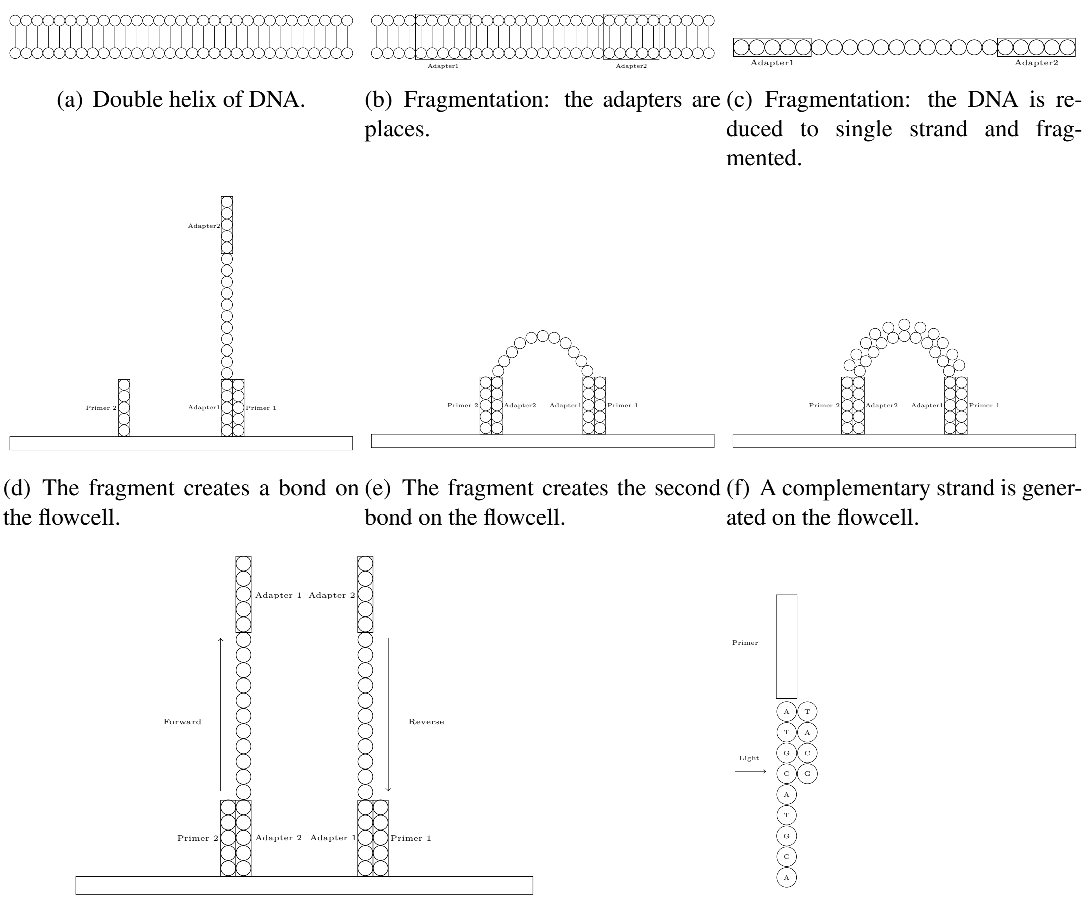
The first phase, called **sample preparation** (Figure 1.2(a), Figure 1.2(b), Figure 1.2(c)), begins with a fragmentation phase that has the aim of splitting the DNA into smaller portions that begin and end with a special sequence (called *adapter*) that matches a special binding site.

Those binding sites (called *primers*) are placed onto a glass panel called *flowcell* that is used to perform the amplification of fragments and their clusterization.

A single fragment creates a bond with the primer corresponding to the first adapter (Figure 1.2(d)) and it is bent to match its second adapter to the second primer on the flowcell (Figure 1.2(e)). A complementary strand is generated (Figure 1.2(f)) and then denatured to form two pairwise-complementary strands (*forward* and *reverse*, as depicted in Figure 1.2(g)).

The process is repeated over and over, resulting in *clonal amplification* from which reverse strands are washed off and all the equal sequences are **clustered**.

The core part of this technology is the **sequencing by synthesis**, where each base is excited using a light source and this leads to the emission of a fluorescent signal that gives birth to the digital read (Figure 1.2(h)).



(a) Double helix of DNA.

(b) Fragmentation: the adapters are places.

(c) Fragmentation: the DNA is reduced to single strand and fragmented.

(d) The fragment creates a bond on the flowcell.

(e) The fragment creates the second bond on the flowcell.

(f) A complementary strand is generated on the flowcell.

(g) The complementary strand is denaturated.

(h) Each base is called thanks to fluorescent technology.

Figure 1.2: Illumina's DNA sequencing.

The second-generation of sequencing technologies, as Illumina, have revolutionized the analysis of DNA and have been the most widely used compared to the first generation of sequencing technologies.

However, the SGS technologies generally require PCR amplification step which is a long procedure in execution time and expensive in sequencing price. To remedy the problems caused by SGS technologies, scientists have developed a new generation of sequencing, called *third generation sequencing* (TGS). The most common sequencing approach of this TGS is the *single molecule sequencing* (SMS), that allows the sequencing of a single cell, without need of any replication. It is important to stress that SMS allows to sequence the RNA of a single cell, and this procedure can be used in order to check if a tissue is healthy or cancerous. The two most famous sequencing platforms in TGS are Pacific Biosciences[i] and the MinION sequencing from Oxford Nanopore technology.

PacBio's sequencer uses the same fluorescent labeling as the other technologies, but instead of executing cycles of amplification nucleotide, it detects the signals in real time.

The MinION[30] from Oxford Nanopore Technologies was released in 2014. It is a *mobile* single-molecule sequencing tool that measures four inches in length and is connected by a USB 3.0 port of a laptop computer (depicted in Figure 1.3). One of MinION's great features (in addition to its portable size) is its speed. A whole genome's sequencing can take less than 60 minutes.

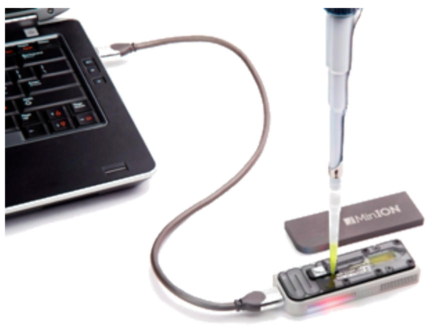


Figure 1.3: Oxford Nanopore's MinION.

So far, we discussed the functioning of various sequencing techniques, but we still need to explain how the reads can be combined into longer sequences.

The process of finding the best alignment of the reads into longer continuous regions is called *assembly*[15 17]. The average microbial genome is about $2m$ basis pairs, but the sequencing technology can output from 100 bases to $5'000$ bases. These reads can be assembled in a hierarchical fashion: we start from reads, we assemble reads into *contigs* and, provided to be able to guess the orientation of contigs, we cal build *scaffolds* (see Figure 1.4).

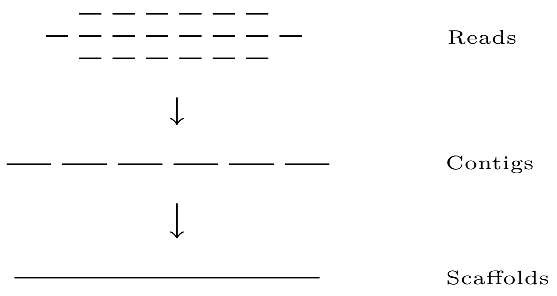


Figure 1.4: The general steps of de novo assembly.

[i]For reference, see https://www.pacb.com/

Under some conditions of the mapping information, scaffolds can be combined to form *chromosomes*.

Among individuals of the same species, it may occur that some parts of their genomes differ. These differences are called *alternate loci*.

Definition 2 (Alternate locus scaffolds)**.** *In a mostly haploid assembly, an* ***alternate locus scaffolds*** *is a sequence that is an alternate representation of a genomic area.*
As a result, different loci are supplied for genomic areas that demonstrate significant population variability and are included in a haploid representation of the genome.

After a brief introduction of bioinformatics fundamentals, we are ready for the next Section in which we present what has been achieved through research in the past years for compressing nucleotide sequences.

1.2 State of the Art

Many compression strategies have been proposed in response to the growing quantity of (re-)sequenced genomes. In general, there are four types of compression algorithms: **naïve bit encoding**, **dictionary-based encoding**, **statistical encoding**, **referential encoding**.
In the following paragraphs, we discuss all these techniques in detail.

1.2.1 Naïve bit encoding

Naïve bit encoding technique[18] [11] [31] exploits fixed-length encodings of two or more symbols in a single byte.
This kind of algorithms are lossless and they exploit regularities in the files (e.g. Nsira[11] identifies and handles palindromic strings) It is evident that using eight bits (or 256 different states) to encode four different bases is a waste of space. With two bits, four bases can be simply encoded (or four states). As a result, bit encoding of four bases into one byte is a straightforward compression technique for DNA sequence data.

An example of naïve bit encoding uses the replacements $A \rightarrow 00$, $C \rightarrow 01$, $G \rightarrow 10$, $T \rightarrow 11$, where each symbol in the input is replaced by two bits. Current CPU designs offer significantly enhanced bit operations, allowing on-the-fly encoding of DNA sequence data with only two bits. In Section 2.4, we will see how FPLGeD uses a simple yet powerful solution for encoding the *nucleotide sequence* such that also the case (upper/lower) can be represented efficiently.

1.2.2 Dictionary-based encoding

Dictionary-based or substitutional compression techniques substitute repeated substrings with references to a dictionary (a collection of previously encountered or predefined common strings) that is created in real time or offline. The state-of-the art pattern-based dictionary approach is represented by Larsson et. al.[25] and Shibata et.al[41].

Dictionary-based encodings are compression algorithms that are largely unaffected by the input data's specific properties. The overall strategy is to use references to a dictionary to replace repeated data elements (in this case, DNA subsequences) in the input. Bookkeeping previously recurring sequences is frequently used to find repetitions. During the decompression process, the dictionary is rebuilt at runtime, therefore, the dictionary does not need to be stored with the compressed data. An example of a dictionary-based algorithm could be this mapping $AAA \rightarrow 1$, $CGT \rightarrow 2$, $TGAG \rightarrow 3$, and so on and so forth. Therefore the sequence $AAATGAGAAACGT$ can be encoded as 1312.

Nowadays, the most famous examples of such technique are Lempel-Ziv-based compression algorithms[46].

FPLGeD makes use of dictionary-based technique for encoding the *quality string*, namely a string that has the same length of the read and stores in each character the confidence of reading that particular nucleotide.

1.2.3 Statistical encoding

Statistical or **entropy encoding** techniques use the input to generate a probabilistic model. This model predicts the following symbols in the sequence based on partial matches of subsets of the input. High compression rates are attainable if the model consistently predicts a high probability for the following symbol, i.e. if the forecast is accurate.

Cleary and Witten's solution[14] belongs to this group, applying arithmetic coding and Markov models to the initial nucleotides sequence. The work of Duc Cao et.al[8] outperforms the competitors in compression ratio, keeping a practical encoding time by means of a panel of experts that suggest the following nucleotide in the sequence, based on the previous occurrences.

Statistical algorithms generate a statistical model of the incoming data, which is usually expressed as a probabilistic or prefix tree data structure. Shorter codes are used for sub-sequences having a greater frequency in the genome. As a result, statistical compression methods may be thought of as a subset of dictionary-based schemes that combine recurrence detection and reference encoding into a single algorithm.

The rate of compression is determined by the model's quality as well as the presence of observable patterns in the input. Huffman encoding[19] is one of the most widely used and well-understood statistical encodings. It employs a variable-length code table constructed from calculated probability for each conceivable symbol's occurrence. Leaf nodes correspond to symbols, while edges are labeled with probabilities and the resulting codes, forming a binary tree.

The resultant Huffman code table must be stored in addition to the compressed stream, and so must be factored into the compression ratio calculation. This storage expense can be reduced by sharing the same code table across many streams. Large alphabets with an uneven distribution of utilized characters benefit Huffman encoding in general[1].

More involved applications of statistical encoding[8 5 43] exploit statistical properties and repetition within sequences.

1.2.4 Reference-based encoding

Referential or **reference-based encoding** is a technique recently emerged. These methods replace large substrings of the input with references to another string, similarly to dictionary-based approaches. These references, on the other hand, point to external sequences that are not part of the input data to be compressed. Furthermore, whereas dictionaries are often enlarged throughout the compression process, the reference is usually static.

The fundamental concept behind reference-based encoding is to record just the differences between a genome sequence and a reference sequence, with the discrepancies being located using absolute or relative coordinates. Various encoding methods can be used to encode these sites and their differential versions into binary strings, as studied by P. Baldi et. al. in their *Data Structures and Compression Algorithms for Genomic Sequence Data*[7].

Referential compression strategies are a straightforward way to read compression since aligning reads to a reference genome is the first step in most analytic processes.

While storing a sequence ‘s’ of N characters ($N + 1$ if we consider the terminating character \0) takes $(N + 1) \cdot 8$ bits, we represent the i-th sequence s_i by its address a_i in the reference genome and save several bits. Let us explain the procedure a bit more in detail.

The length of the i-th sequence l_i must additionally be given if the length of each sequence is not set and known in advance or included in the file header. If the match is not perfect, differences in the genomic sequence must be recorded, along with their address and kind. Let us suppose to have a short sequence with a beginning point matching location 1500 in the reference genome, a length of 25 nucleotides, and a C substitution in position 3 may be recorded as $(1500, 25, 3C)$.

This structure, is easily compressed using tools like DNAzip[12].

As mentioned in the previous paragraphs, FPLGeD is a genomic data compression tool structured in 3 sub-procedures using bit encoding and dictionary mapping techniques, depending on the type of data under consideration. In the section below, we present the file types that FPLGeD deals with compressing and highlight their characteristics.

1.3 The input data

The aforementioned genome information is stored in mainly two formats: FASTA and FASTQ.
In this section, we explain more in detail the format of such files.

Definition 3 (FASTA file)**.** *In bioinformatics and biochemistry, the* ***FASTA format*** *(shown in Figure 1.5) is a text-based format for representing either nucleotide sequences or amino acid (protein) sequences, in which nucleotides or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences.*
The structure of FASTA files is the following:

- ***Line 1 (Header)*** *begins with a ‘>’ character and is followed by a sequence identifier and an optional description of the sequence itself;*

- ***Line 2 (Sequence)** is the raw sequence letters.*

In this study, we concentrate on sequences of nucleotides. The reason why we chose to formulate the problem in such a way as to compress only sequences of nucleotides and not of amino acids is that the first type of file is far more widespread and is in a sense to be considered as another way to express amino acids and proteins: in fact it is from DNA that proteins and amino acids are obtained, thanks to the translation of mRNA.

A **sequence** was not written on a single line in the original format; instead, it was represented as a series of lines, each of which was no more than 120 characters long and frequently no more than 80 characters long. Most users at the time relied on Digital Equipment Corporation (DEC) VT220 (or comparable) terminals that could show 80 or 132 characters per line, hence this was most likely to allow for pre-allocation of fixed line sizes in software. In 80-character modes, most people favored the larger font, so it became fashionable to use 80 characters or less (typically 70) in FASTA lines. A regular printed page is also 70 to 80 characters wide (depending on the font). As a result, 80 characters became the standard.

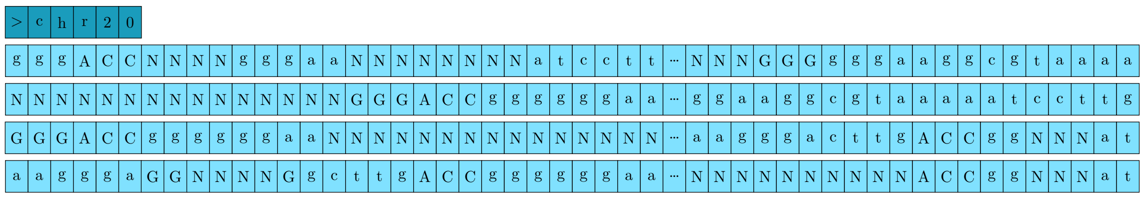
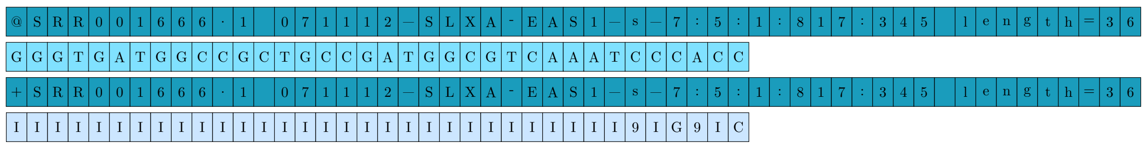

> c h r 2 0
g g g A C C N N N N g g g a a N N N N N N N N a t c c t t ⋯ N N N G G G g g g a a g g c g t a a a a
N N N N N N N N N N N N N N N G G G A C C g g g g g a a ⋯ g g a a g g c g t a a a a a t c c t t g
G G G A C C g g g g g g a a N N N N N N N N N N N N N N N ⋯ a a g g g a c t t g A C C g g N N N a t
a a g g g a G G N N N N G g c t t g A C C g g g g g a a ⋯ N N N N N N N N N N A C C g g N N N a t

Figure 1.5: In dark blue, the **header** of the FASTA file and in light-blue, the **sequence** of nucleotides split in multiple lines of 80 characters each.

FASTA files are possibly made of multiple sequences, and this enables for storing an entire genome on a single file. In terms of file format, we get that the two lines with the structure shown above are repeated multiple times across the file.

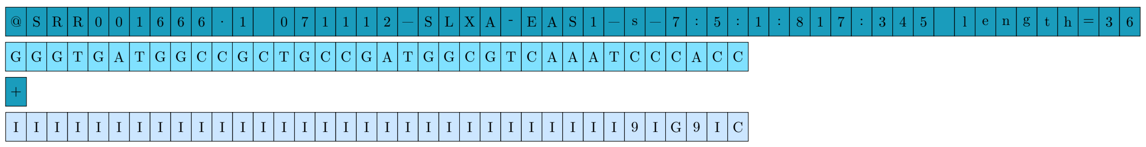
Definition 4 (FASTQ file)**.** ***FASTQ format** (shown in Figure 1.6) is a text-based format for storing both a genomic sequence 's' and its corresponding quality scores. Both the sequence letter and quality score are each encoded with a single ASCII character for brevity.*
A FASTQ file uses four lines per sequence, as shown in Figure 1.6 and explained below.

- ***Line 1 (Header)** begins with a '@' character and is followed by a sequence identifier and an optional description (like a FASTA title line);*
- ***Line 2 (Sequence)** is the raw sequence letters;*
- ***Line 3 (Plus)** begins with a '+' character and is optionally followed by the same sequence identifier (and any description) again;*
- ***Line 4 (Quality)** encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.*



```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
IIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9IC
```

(a) FASTQ with “plus line”



```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9IC
```

(b) FASTQ with empty “plus line”

Figure 1.6: In dark blue, the **header** of the FASTQ file and the **plus** line; in light-blue, the **sequence** of nucleotides; then, in a lighter shade of blue, the **quality** values.

As already stressed for FASTA files, in a FASTQ file are stored multiple reads, which means that the four lines with the structure shown above are repeated multiple times across the file.

Chapter 2

FPLGeD: Fast Packing of Large Genomic Data

In this Chapter, we explain in detail the functioning of FPLGeD (**F**ast **P**acking of **L**arge **Ge**nomic **D**ata) compressor, after introducing some preliminary definitions on the subject.

2.1 Preliminaries

When we talk about compression, we refer to two macro areas: *lossless compression* where it is always possible to reconstruct the complete original input from the compressed output, as opposed to *lossy compression* where a limited degree of divergence between input and output is possible.
In biomedical applications every single base (nucleotide) is important, therefore from now on we will focus only on lossless compression tools, and we will define the performance in terms of space saving as

Definition 5 (Compression Ratio)**.** *We term **compression ratio** the ratio between the original size and the compressed size:*

$$\text{COMPRESSION RATIO} = \frac{s_{original}}{s_{compressed}}$$

We say that a compression scheme allows *random access*, if arbitrary positions of the input stream can be accessed without decompressing the whole stream. Splitting the input sequence into fixed-size blocks, for example, can permit random access.

Definition 6 (Run Length Encoding)**.** ***Run Length Encoding** (or **RLE**) is one of the most basic forms of compression. RLE is a fundamental data compression method that turns a string of identical values into a code that includes the character and the length of the run. The greater the number of identical values, the more values may be compressed.*

Example:
A first trivial example of the functioning of RLE is the translation of AAA into $3A$, where the 3 is the *run* (indicating that this specific character was stored 3 times in the original data), and the A is the run-value (indicating, that the repeated character is A).
Let us move to a more involved example: s = `BBBBAAOPPOOOOP`.
$RLE(s)$ = 4B 2A 1O 2P 4O 1P, allowing s to be stored using 12 characters instead of 14. In this case, we saved 2 characters compared to the original string. At decoding time, we read from the encoded data the run and then the run-value. Subsequently, we store the run-value for run-times until we decoded and by that restored the original data.

- 4B $\rightarrow$ BBBB
- 2A $\rightarrow$ AA
- 1O $\rightarrow$ O
- 2P $\rightarrow$ PP
- 4O $\rightarrow$ OOOO
- 1P $\rightarrow$ P

After this procedure, we get back to the initial string: BBBBAAOPPOOOOP.

From this example, we can glimpse one of the downsides of RLE: two characters in a sequence (like the AA or the PP) never create compression, as the encoded data is of the same size as the original data. Moreover, single characters (like the first O and the last P in the example above), that occupy only one byte in the original data, also get an additional run during the encoding-process. In the latter case, the encoded data becomes twice the size of the original.

Conversely, it is easy to observe that a sequence consisting only of multiple repetitions of the same character exploits this techniques at its best.

In order to make a wise use of this encoding technique, in Section 2.5 we will present a mixed approach that combines RLE encoding with a plain encoding. In particular, in Section 2.5.2 and Section 2.5.4, we will use RLE only if the number of repeated characters reaches a certain threshold.

2.2 FPLGeD

The problem that FPLGeD solves is reducing the disk space occupied by files that store possibly large sequences of nucleotides without loss of information, by means of highly optimized encoding techniques.

In particular, our tool allows the encoding of two different kinds of files, FASTA and FASTQ (Definition 3 and Definition 4), and such procedure is held by two or three sub-tasks respectively.

Chapter 3

Experimental Results

In the previous chapter, we described in detail the operation of our FPLGeD; it will be in this second part of the discussion that we will comment on the experimental results obtained on a large dataset of genomic data and compare them with the theoretical performance.

We employed the state-of-the-art competitors, GZip (the most frequently used FASTQ compressor[32]), and SPRING14[10] for the evaluation. We decided not to include other good compressors like FaStore[37] and Minicom[28] because SPRING14 outperforms them both in terms of time and compression ratio[32].

This chapter is divided into three parts: the first serves as an introduction to the data under consideration and is concerned with describing and documenting the dataset. In the second section (Section 3.2) we report on the performance of the compared methods on FASTQ files, while the last section (Section 3.3) is left to the analysis and comparison of our algorithm on FASTA files.

3.1 Dataset

The performance of FPLGeD was tested and compared on a variety of publicly available sequence datasets mostly downloaded from NCBI[i] [39].

Let us first describe the dataset:

- 4 representative samples (i.e. the biggest) from a wide collection of 61 runs of small RNA-seq (miRNAs) described in[44] downloaded from NCBI trace (study accession number SRP1993503[ii]). The samples consist of the same human brain tissue, in various concentrations, prepared with 4 different protocols and sequenced with an Illumina HiSeq-3000. In particular
    - `Clontech5_Sample5_batch1.fastq` was prepared using the Clontech SMARTer smRNA-Seq Kit for Illumina (Clontech);

[i]NCBI home page: https://www.ncbi.nlm.nih.gov/assembly
[ii]https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?study=SRP199350

- `Illumina4_batch1.fastq` was prepared with the Illumina TruSeq Small RNA Library Prep Kit (TruSeq);
- `NEB12_batch1.fastq` was prepared using the New England BioLabs Next Multiplex small RNA kit (NEB)[iii] ;
- `NEXTFlex_batch2.fastq` was prepared with the Bioo Scientific NEXTflex Illumina Small RNA Sequencing Kit v3 (NEXTflex)[iv];

◗ `ERR194147` is a paired-end sequencing of the genome of the individual *NA*12878 (i.e. the mother) belonging to the *CEPH/UTAH PEDIGREE* shown in Figure 3.1)[33]. Sequencing was performed using Illumina HiSeq 2000[27 9] and consists of three files:

- `ERR194147_1.fastq` contains the read1;
- `ERR194147_2.fastq` contains the read2;
- `ERR194147` contains reads for which the pair is not present;

◗ `SRR10407349.fastq` contains reads of non-coding RNA of *homo sapiens*. The reads present in this file are obtained using Illumina MiSeq system;

◗ `SRR13632831.fastq` is a FASTQ file containing the RNA sequencing of a fish named brook trout (*Salvelinus fontinalis*) and are obtained making use of Illumina HiSeq 2500;

◗ `SRR17333571.fastq` contains the RNA reads of a plant, the Chinese ginseng (*panax schinseng*). SUch reads are obtained through Illumina NovaSeq 6000 technology;

◗ `SRR8311266.fastq` stores the same data of `SRR10407349.fastq`, but this time the reads come from Illumina HiSeq-2500 technology;

◗ `nanopore-NA12878.fq` contains the whole genome sequencing of sample *NA*12878, where the sequencing was performed using Oxford Nanopore Technologies (ONT) MinION long-read sequencer[6 20]. With respect to Illumina sequencing (second generation technology), the length of the reads is much higher, although Illumina sequencing is more accurate in terms of quality of the calls;

◗ `pacbio-k3-NA12878.fq` store the whole sequencing of the same individual *NA*12878, but this sequencing is performed using PacBio technology[36]. Similarly to Nanopore sequencing, PacBio belongs to third generation sequencers and produces longer, although less accurate, reads with respect to Illumina technology.

[iii]For further readings on NEB, https://international.neb.com/tools-and-resources/interactive-tools

[iv]For reference, see https://perkinelmer-appliedgenomics.com/home/products/library-preparation-kits/small-rna-library-prep/nextflex-small-rna-seq-kit-v3/

- `GCF_000001405.26_GRCh38_genomic.fna` contains the reference genome, assembled at chromosomic level of a human being (*homo sapiens*). This build has a total of 207 alternate-locus-containing regions associated with a total of 473 alternate locus scaffolds[v];

- `GCF_000001635.27_GRCm39_genomic.fna` stores the reference genome, assembled at chromosomic level, of zebrafish (*danio rerio*). Let us stress that the zebrafish is widely used in bioinformatics and in genomic research mainly because they have a reasonably similar genomic structure to humans (they share 70% of genes with us) and the 84% of genes known to be associated with human disease have a zebrafish counterpart.
  Moreover, they are cheaper to maintain with respect to mice. In this file as well, there are stored also the alternate loci for a total of 607 alternate-locus-containing regions associated with a total of 1917 alternate locus scaffolds;

- `GCF_000002035.6_GRCz11_genomic.fna` contains the reference genome, assembled at chromosomic level, of house mouse (*mus musculus*). Just like the other two reference genomes, alternate loci are present, although in the small amount of 3 and with 102 alternate locus scaffolds;

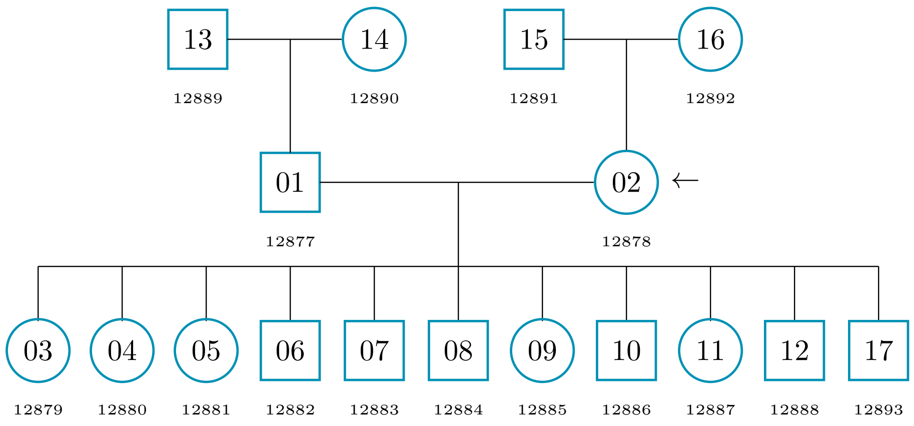


Figure 3.1: The family tree of CEPH Family 1463. In this figure, males are represented as squares and females as circles. Individual *NA*12878 (mother) is marked with the symbol ←.

As far as the technical details are concerned, let us group the files into Table 3.2 and Table 3.1.

[v]The human genome reference sequence was formerly represented as a single consensus sequence known as the *golden route*[23]. Several chromosomal regions have such a high degree of diversity that a single nucleotide cannot effectively represent them[24 47 45]. As a result, the GRC began to give alternate sequences for chosen variation regions by including alternate locus scaffolds (for the definition, see (Definition 2).

Sample	Name	Size(KB)	Type	Method	Instrument	N_{reads}	R_{length}
Human	np-NA12878	183 238 556	DNA	ONT	MinION	83 237 108	/
	pb-k3-NA12878	7 165 020	DNA	PacBio	SMRT	2 618 188	/
	ERR194147_1	207 494 172	DNA	Illumina	HiSeq2000	3 149 060 436	101
	ERR194147_2	207 494 172	DNA	Illumina	HiSeq2000	3 149 060 436	101
	ERR194147	2 173 144	DNA	Illumina	HiSeq2000	32 963 184	101
H. brain	Ct5_S5_b1	9 872 180	DNA	Clontech	HiSeq3000	238 508 524	51
	Ill4_batch1	1 225 876	DNA	Illumina	HiSeq3000	31 672 308	51
	NEB12_batch1	7 930 268	DNA	NEB	HiSeq3000	204 889 280	51
	NEXTF_batch2	12 349 932	DNA	NEXTFlex	HiSeq3000	319 073 916	51
Human	SRR10407349	480 116	RNA	Illumina	MiSeq	9 449 128	65
Human	SRR8311266	4 945 096	RNA	Illumina	HiSeq2500	81063812	51
B. trout	SRR13632831	23 965 044	RNA	Illumina	HiSeq2500	203 675 076	202
Ginseng	SRR17333571	36 457 676	RNA	Illumina	NovaSeq6000	216 678 820	302

Table 3.1: The FASTQ dataset split into groups.

Notice that in Table 3.1 the read length in the case of Nanopore and PacBio sequencing is not present since they are variable.

Sample	Name	Size(KB)	Total seq. length	$N_{chromosomes}$
Human	GCF_000001405.26_GRCh38_genomic	3 134 124	3 099 734 149	24
Zebrafish	GCF_000001635.27_GRCm39_genomic	2 664 292	1 373 454 788	25
Mouse	GCF_000002035.6_GRCz11_genomic	1 640 076	2 728 222 451	22

Table 3.2: FASTA dataset stats.

In the following section, we describe the experimental results of our FPLGeD tool against two main competitors: the common, general-purpose GZip compressor and SPRING[10], a reference-free compressor tailored for FASTA and FASTQ files.

The experimental results are divided into two main categories (FASTA and FASTQ) and this is due to the fact that the structure of the two file types and their sizes are very different and treating them simultaneously would hamper the data visualization.

All tests were performed on a machine with a Intel Quad-Core i7-3770, 3.40*GHz* CPU, 8*GiB* of memory, and running Ubuntu 20.04 Linux as operating system.

3.2 Performance on FASTQ files

The performance of FPLGeD on FASTQ files is evaluated from three points of view: the encoding time, the encoded size and the decoding time. In Table 3.3, we can see the compression performance of FPLGeD against GZip (the most commonly used FASTQ compressor in practice) and the well-known SPRING compressor.

Name	Size(KB)	FPLGeD(KB)	GZip(KB)	SPRING(KB)
nanopore-NA12878	183 238 556	82 008 924	82 523 720	82 453 876
pacbio-k3-NA12878	7 165 020	3 173 716	3 059 020	2 360 456
ERR194147_1	207 494 172	85 632 576	50 227 020	33 406 176
ERR194147_2	207 494 172	87 268 872	51 434 560	34 331 324
ERR194147	2 173 144	909 876	702 908	428 720
Clontech5_Sample5_batch1	9 872 180	3 609 852	1 716 276	942 316
Illumina4_batch1	1 225 876	446 248	163 916	83 252
NEB12_batch1	7 930 268	2 770 784	980 516	487 292
NEXTFlex_batch2	12 349 932	3 794 016	1 523 728	766 096
SRR10407349	480 116	145 808	93 440	44 192
SRR8311266	4 945 096	1 142 368	698 900	292 052
SRR13632831	23 965 044	8 737 000	7 811 356	5 177 244
SRR17333571	36 457 676	11 773 932	6 925 276	4 023 616

Table 3.3: The FASTQ dataset size performance.

Name	Size(KB)	FPLGeD ratio	Gzip ratio	SPRING ratio
nanopore-NA12878	183 238 556	2.23×	2.22×	2.22×
pacbio-k3-NA12878	7 165 020	2.25×	2.34×	3.04×
ERR194147_1	207 494 172	2.40×	4.13×	6.21×
ERR194147_2	207 494 172	2.38×	4.03×	6.04×
ERR194147	2 173 144	2.39×	3.09×	5.07×
Clontech5_Sample5_batch1	9 872 180	2.73×	5.75×	10.48×
Illumina4_batch1	1 225 876	2.75×	7.48×	14.72×
NEB12_batch1	7 930 268	2.86×	8.09×	16.27×
NEXTFlex_batch2	12 349 932	3.26×	8.11×	16.12×
SRR10407349	480 116	3.29×	5.14×	10.86×
SRR8311266	4 945 096	4.33×	7.08×	16.93×
SRR13632831	23 965 044	2.74×	3.07×	4.63×
SRR17333571	36 457 676	3.10×	5.26×	9.06×
average	//	2.82×	5.06×	9.36×

Table 3.4: The FASTQ dataset compression ratios.

As table Table 3.4 shows, FPLGeD compression ratio has an average of 2.42×, which is slightly worse than Gzip (5.06×) and three times worse than SPRING (9.36×). The same data of Table 3.3, is presented as bar-plot in Figure 3.2, where on the x axis there are the FASTQ file names, sorted in lexicographic order and on the y axis there are the sizes of original files and their compressed versions measured in KBs.

In particular, we observe that on average files packed with FPLGeD are twice as large as the same compressed with GZip and SPRING. A notable exception is for third generation sequencing datasets in which the high entropy of quality scores causes bwt-based compression of both GZip and SPRING to become inefficient.

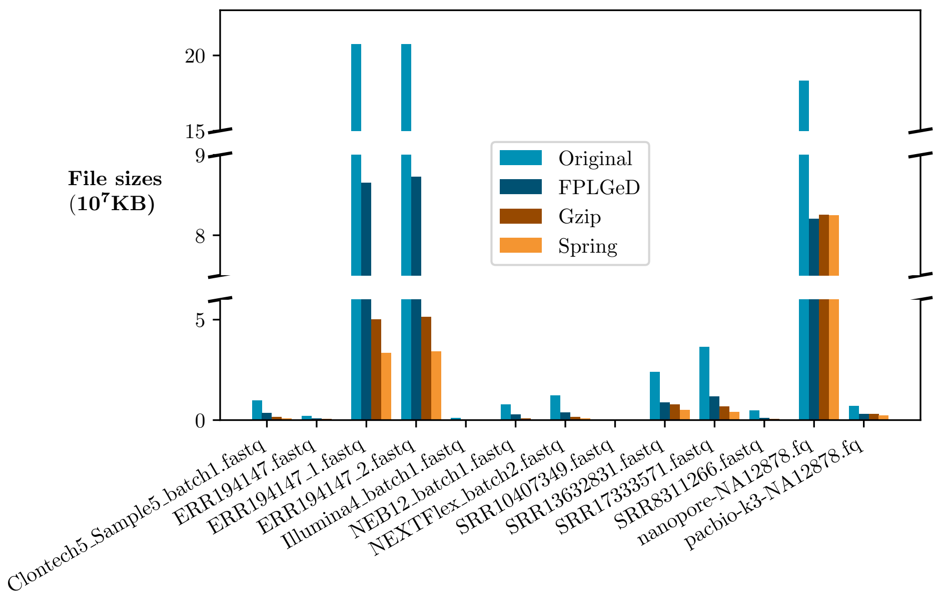


Figure 3.2: Comparison between original size, FPLGeD encoding, GZip compressed size and SPRING encoding size.

This result is not to be interpreted as a failure of the procedure, because FPLGeD applies only an encoding function to the various parts of the file, without performing a real compression, unlike competitors.

The choice to perform only an encoding, as already said, derives from the satisfaction of the specifications defined in the introduction: to develop a tool able to reduce the size of files containing DNA or RNA that allows fast random access and that is very light in terms of compression and decompression speed.

The real advantage of FPLGeD is that, if further space reduction is required, a simple compression algorithm (such as GZip) can be applied to the result to obtain a very small file in a rather low overall compression time.

In particular, FPLGeD does not alter the entropy of the file and performs a size reduction such that applying GZip to the encoded file takes a very short time.

In order to evaluate the compression and decompression speed of FPLGeD, we look at Figure 3.3 and Figure 3.4 that represent respectively the encoding times of the three methods analyzed, and the fastest of the two.

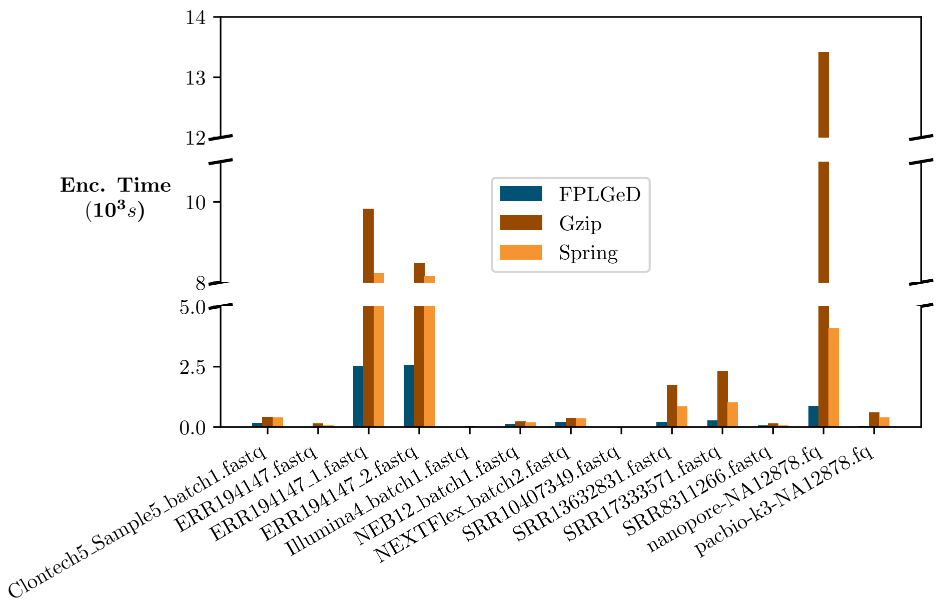


Figure 3.3: Encoding times of FPLGeD, Gzip and SPRING.

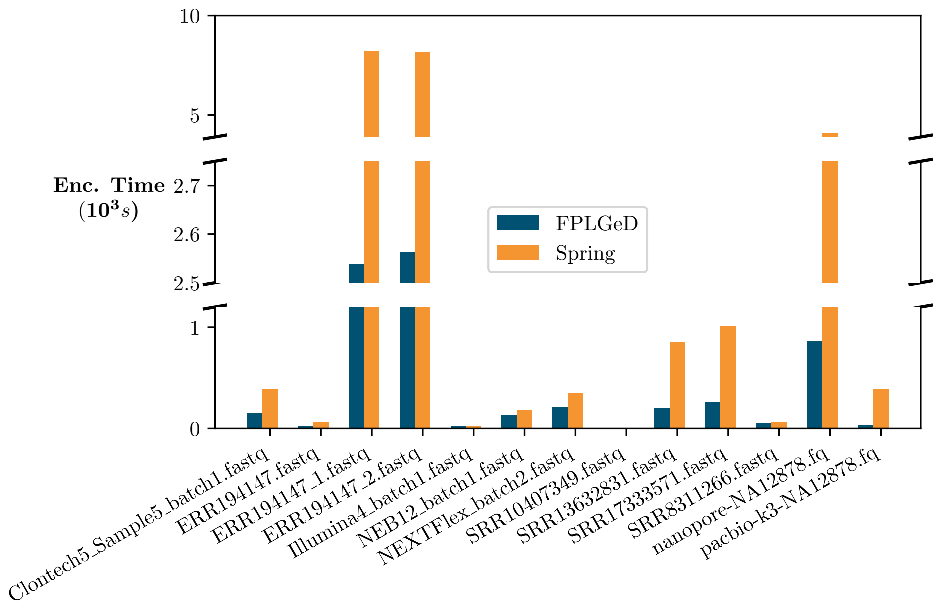


Figure 3.4: Encoding times of FPLGeD and SPRING.

The time consumed by FPLGeD in encoding is on average 0.18× the time spent by GZip compression, and 0.28× the time used by SPRING. Moreover, in the case of third generation sequencing (aka the most widely used and transferred nowadays), FPLGeD is more than 4 times faster than its competitors. For example, FPLGeD requires 14 minutes for compressing the 174GB `nanopore-NA128878.fq` dataset, which outperforms SPRING (68 minutes). Let us go back on Figure 3.2 and examine the compression ratio of FPLGeD on `nanopore-NA128878.fq`. Not only our tool is faster than its competitors, but it is also better in terms of space occupancy.

In Chapter 2 we described FPLGeD as an algorithm that combines three different procedures to encode the three different parts that constitute a FASTQ file. In Figure 3.5 we see how the relationships between DNA sequence and quality score changes. If initially, they were two strings of equal length, it is clear from the experimental results that the ability of compressing the nucleotide sequence is far greater than the compression ratio of quality. This is consistent with what already shown in the theorems regarding the compression ratios of sequence encoding and quality encoding (Theorem 2.4.4, Theorem 2.5.2, Theorem 2.5.6, Theorem 2.5.9, Theorem 2.5.13, Theorem 2.5.16).

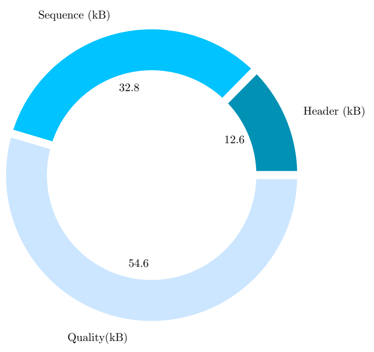


Figure 3.5: Partial encoded sizes on average across the three parts of a FASTQ file (`hdr`, `seq`, `qlt`).

Regarding decoding time, we see that SPRING is up to 4 times slower than FPLGeD and Gzip (Figure 3.6). When comparing with GZip, we see that our algorithm is slower by a factor of 1.5 on average, but in the best cases (reads sequenced with Nanopore and Pacbio, the third generation sequencers), the performance of FPLGeD improves by up to a factor of 4 that of GZip (Figure 3.7).

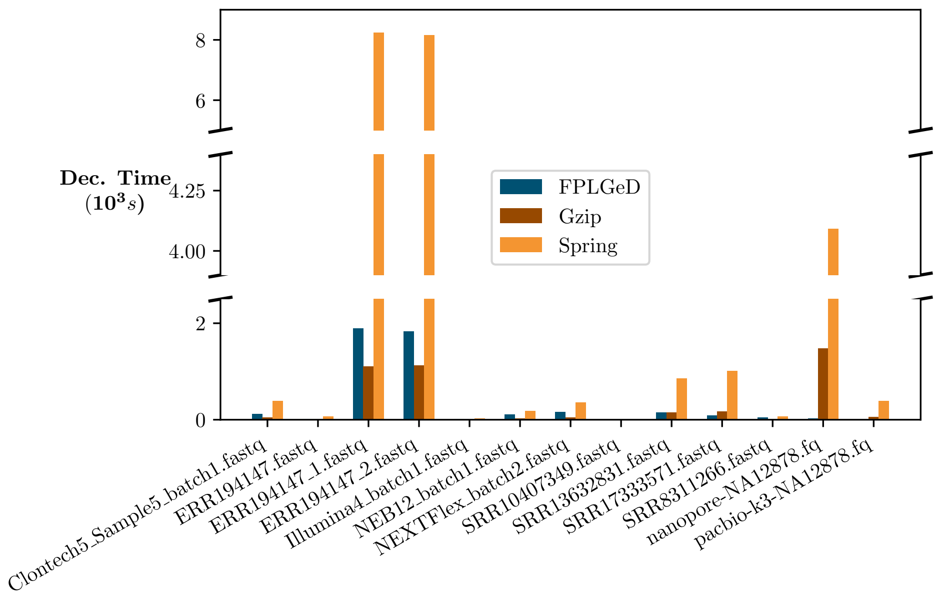


Figure 3.6: Decoding times of FPLGeD encoding, GZip, and SPRING.

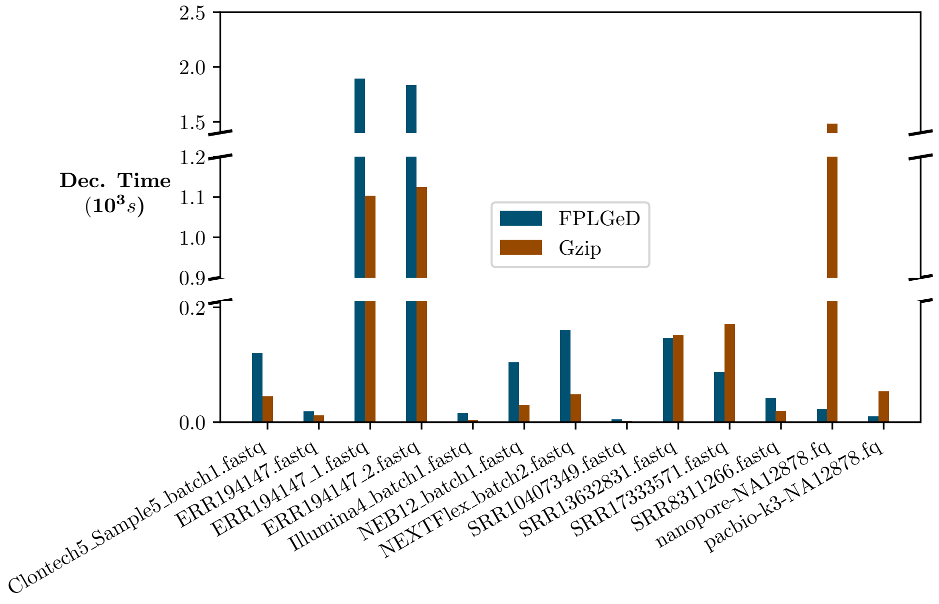
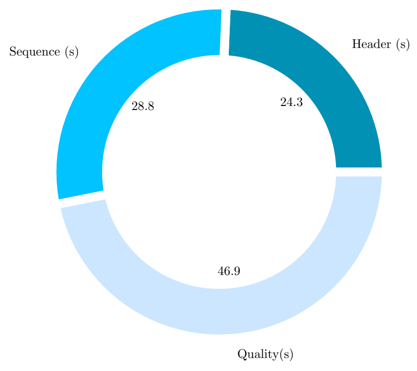


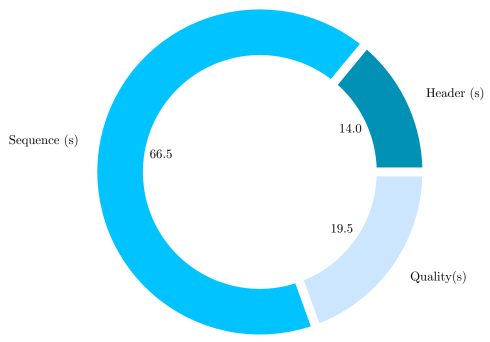
Figure 3.7: Decoding times of FPLGeD encoding, and GZip.

Before concluding the section on processing FASTQ files, we would like to show the percentage breakdown of encoding and decoding times across the 3 sub-procedures (`hdr`, `seq`, `qlt`).

Figure 3.8 shows that at encoding time the most expensive procedure in terms of time is the encoding of the quality score (`qlt`), while at decoding time it is more expensive to decode the sequence (it takes the 66% of the total time.



(a) Encoding



(b) Decoding

Figure 3.8: Encoding/decoding time distribution across the three procedures (`hdr`, `seq`, `hdr`).

3.3 Performance on FASTA files

In Section 3.1, we presented the FASTA files as the assembled genomes of three different species: *human*, *zebrafish*, *mice*, which sizes (plain and compressed/encoded) are shown in Table 3.5 and in Figure 3.9.

Sample	Name	Size(KB)	FPLGeD(KB)	GZip(KB)	SPRING(KB)
Human	`GCF_000001405.26_GRCh38_genomic`	3 134 124	967 328	898 548	664 220
Zebrafish	`GCF_000001635.27_GRCm39_genomic`	2 664 292	817 772	794 120	567 912
Mouse	`GCF_000002035.6_GRCz11_genomic`	1 640 076	512 884	500 976	360 840

Table 3.5: FASTA dataset plain and encoded sizes.

Sample	Name	Size(KB)	FPLGeD ratio	GZip ratio	SPRING ratio
Human	`GCF_000001405.26_GRCh38_genomic`	3 134 124	3.24×	4.72×	3.49×
Zebrafish	`GCF_000001635.27_GRCm39_genomic`	2 664 292	3.20×	4.55×	3.27×
Mouse	`GCF_000002035.6_GRCz11_genomic`	1 640 076	3.26×	4.69×	3.36×
//	**average**	//	3.23×	3.37×	4.65×

Table 3.6: FASTA dataset compression ratios.

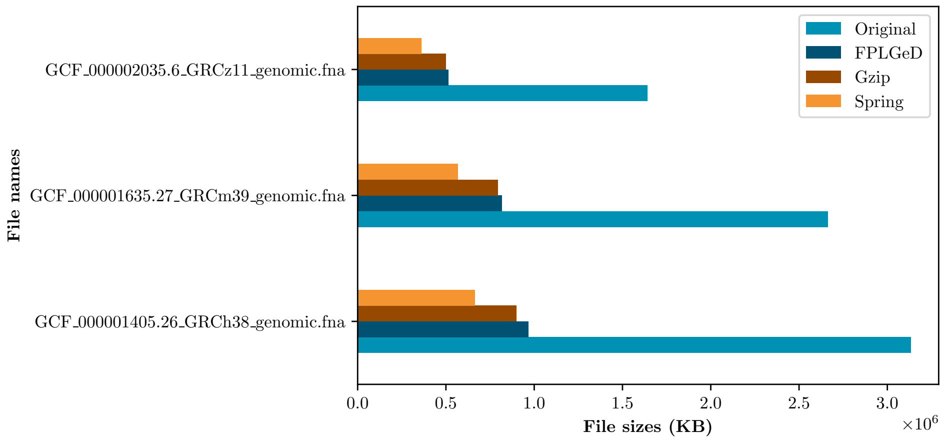


Figure 3.9: Comparison between original size, FPLGeD encoding, GZip compressed size and SPRING encoding size.

In the case of FASTA we note that FPLGeD is still less performing in terms of compression ratio when compared to SPRING and GZip, but this time the gap is much smaller. In particular, FPLGeD gets on average a compression ratio of 3.23×, GZip 3.37× and SPRING 4.65×.

The discrepancy in results, if we compare the compression ratio of FPLGeD on FASTQ and FASTA, is originated by the fact that the encoding of the string containing the qualities is the part that degrades performance the most. As explained in Definition 3, FASTA files usually represent the entire genome of a species and do not contain the quality scores, but only header and sequence. Moreover, when compared to the size of the header, the sequence is several orders of magnitude larger (at least 4) and therefore, in the average case, the conditions are met to get the best out of the sequence encoding algorithm.

As far as encoding/compression times are concerned, FPLGeD performs much better than its competitors in the case of FASTA files, as can be seen from Figure 3.10. On average, the time necessary to encode a FASTA file using FPLGeD is 1/30 of the time needed by GZip and 1/35 the time needed by SPRING. For example, encoding the whole human reference genome (`GCF_000001405.26_GRCh38_genomic.fna`, 3GB) by means of FPLGeD takes 11 seconds, GZip needs 5 minutes, and SPRING 7 minutes.

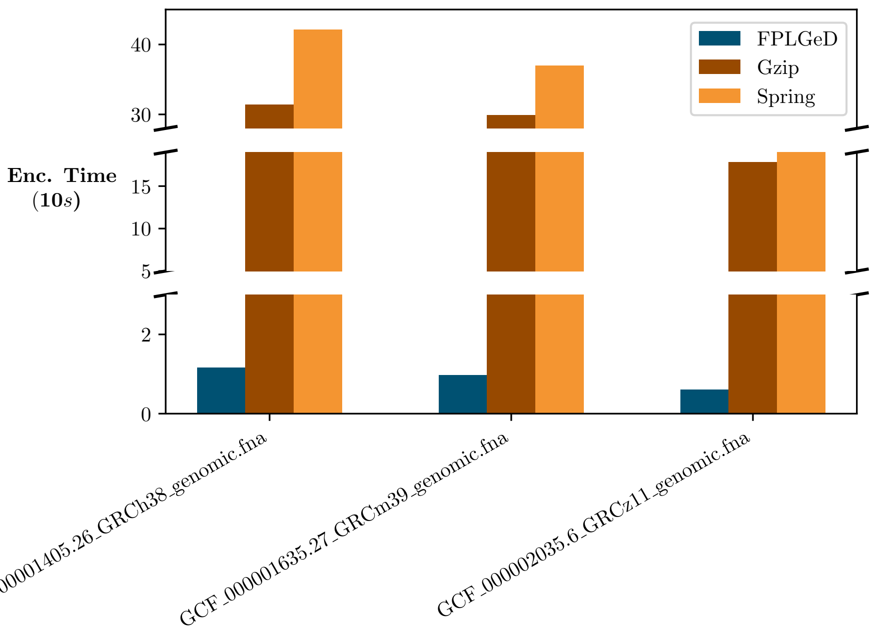


Figure 3.10: Comparison between the encoding times of FPLGeD, GZip, and SPRING.

Decompression of the previously encoded sequence is even faster than encoding. In this case in fact, taking as an example the human genome, we see that FPLGeD takes 3.4 seconds, Gunzip 18 and SPRING 7 minutes (exactly the time taken for compression and this is due to the nature of the algorithm). To get a visualisation of this, look at Figure 3.11.

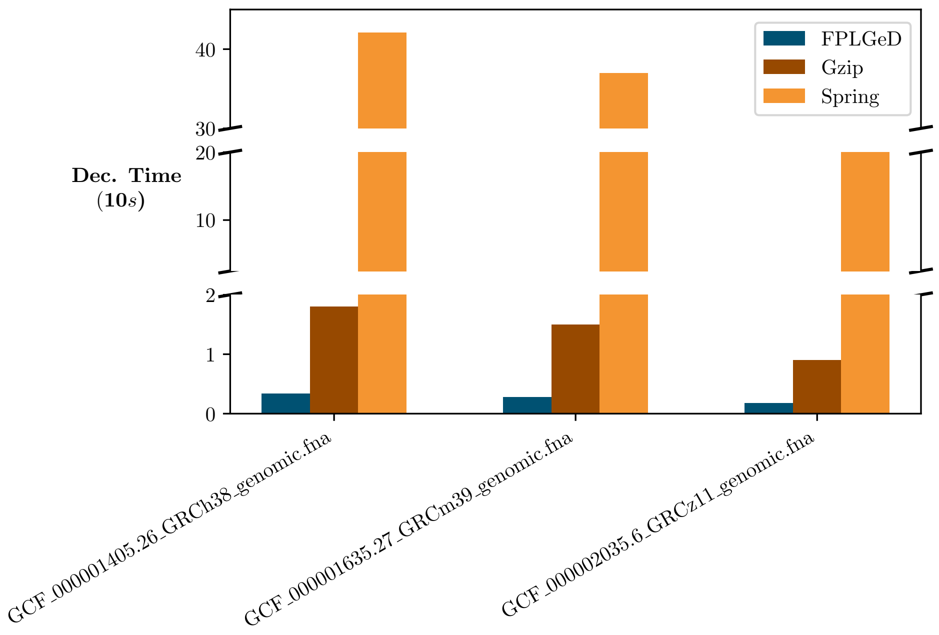


Figure 3.11: Comparison between the decoding times of FPLGeD, GZip, and SPRING.

Chapter 4

Conclusions

This thesis addressed the problem of designing and testing an algorithm able to reduce the size of files containing genomic information, specifically FASTA and FASTQ file types.

The proposed tool, is based on the FPLGeD algorithm that consists in three routines: one for encoding the headers of the sequences, one for encoding the nucleotide sequences and the last one (required only for FASTQ files) deals with encoding the quality score.

From a theoretical point of view, FPLGeD offers an excellent compromise between encoding/decoding speed and compression ratio, ensuring also the possibility to make random accesses to a sequence without the need to entirely decode it. It is also worth noting that the proposed tool does not alter the entropy of the file and, thanks to its extreme speed, is suitable to be used in combination with any compression software, such as GZip.

Following an accurate experimental analysis, carried out by comparing FPLGeD with other algorithms frequently used to compress FASTA(Q) files, we found that the algorithm confirms in practice the characteristics highlighted by a theoretical analysis, offering a good reduction in terms of space and taking very little time to be executed.

FPLGeD outperforms its competitors on third generation sequencing data that is not only the most widely used nowadays, but it also represents the direction towards which all the modern sequencers are focusing on.

Given the effectiveness of this tool, it is proposed as a future work to investigate the I/O bottleneck, in particular by creating a library capable of loading the entire contents of the file into memory.

Furthermore, we see it worth trying to combine this encoding method with a real compression mechanism, in order to decrease the entropy of the file, thus obtaining a higher compression ratio, while keeping a reasonable compression/decompression speed.

Bibliography

[1] A. Al-Okaily, B. Almarri, S. A. Yami, and C.-H. Huang, *Toward a better compression for DNA sequences using huffman encoding*, Journal of Computational Biology, 24 (2017), pp. 280–288.

[2] A. Auton, G. R. Abecasis, D. M. Altshuler, R. M. Durbin, G. R. Abecasis, D. R. Bentley, A. Chakravarti, A. G. Clark, P. Donnelly, and e. a. Evan E. Eichler, *A global reference for human genetic variation*, Nature, 526 (2015), pp. 68–74.

[3] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, J. M. Boutell, J. Bryant, R. J. Carter, R. K. Cheetham, and A. J. C. et. al., *Accurate whole human genome sequencing using reversible terminator chemistry*, Nature, 456 (2008), pp. 53–59.

[4] B. Berger, J. Peng, and M. Singh, *Computational solutions for omics data*, Nature Reviews Genetics, 14 (2013), pp. 333–346.

[5] V. Bhola, A. S. Bopardikar, R. Narayanan, K. Lee, and T. Ahn, *No-reference compression of genomic data stored in FASTQ format*, in 2011 IEEE International Conference on Bioinformatics and Biomedicine, IEEE, Nov. 2011.

[6] R. Bowden, R. W. Davies, A. Heger, A. T. Pagnamenta, M. de Cesare, L. E. Oikkonen, D. Parkes, C. Freeman, F. Dhalla, S. Y. Patel, N. Popitsch, C. L. C. Ip, H. E. Roberts, S. Salatino, H. Lockstone, G. Lunter, J. C. Taylor, D. Buck, M. A. Simpson, and P. Donnelly, *Sequencing of human genomes with nanopore technology*, Nature Communications, 10 (2019).

[7] M. C. Brandon, D. C. Wallace, and P. Baldi, *Data structures and compression algorithms for genomic sequence data*, Bioinformatics, 25 (2009), pp. 1731–1738.

[8] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, *A simple statistical algorithm for biological sequence compression*, in 2007 Data Compression Conference (DCC'07), IEEE, 2007.

[9] J. G. Caporaso, C. L. Lauber, W. A. Walters, D. Berg-Lyons, J. Huntley, N. Fierer, S. M. Owens, J. Betley, L. Fraser, M. Bauer, N. Gormley, J. A. Gilbert, G. Smith, and R. Knight, *Ultra-high-throughput microbial community analysis on the illumina HiSeq and MiSeq platforms*, The ISME Journal, 6 (2012), pp. 1621–1624.

[10] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman, *SPRING: a next-generation compressor for FASTQ data*, Bioinformatics, 35 (2018), pp. 2674–2676.

[11] L. Chen, S. Lu, and J. Ram, *Compressed pattern matching in DNA sequences*, in Proceedings. 2004 IEEE Computational Systems Bioinformatics Conference, 2004. CSB 2004., IEEE, 2004.

[12] S. Christley, Y. Lu, C. Li, and X. Xie, *Human genomes as email attachments*, Bioinformatics, 25 (2008), pp. 274–275.

[13] L. Clarke, S. Fairley, X. Zheng-Bradley, I. Streeter, E. Perry, E. Lowy, A.-M. Tassé, and P. Flicek, *The international genome sample resource (IGSR): A worldwide collection of genome variation incorporating the 1000 genomes project data*, Nucleic Acids Research, 45 (2016), pp. D854–D859.

[14] J. Cleary and I. Witten, *Data compression using adaptive coding and partial string matching*, IEEE Transactions on Communications, 32 (1984), pp. 396–402.

[15] A. Clum, *Genome assembly*, in Methods in Molecular Biology, Springer New York, 2018, pp. 141–153.

[16] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, *The sanger FASTQ file format for sequences with quality scores, and the solexa/illumina FASTQ variants*, Nucleic Acids Research, 38 (2009), pp. 1767–1771.

[17] R. Ekblom and J. B. W. Wolf, *A field guide to whole-genome sequencing, assembly and annotation*, Evolutionary Applications, 7 (2014), pp. 1026–1042.

[18] S. Grumbach and F. Tahi, *A new challenge for compression algorithms: Genetic sequences*, Information Processing & Management, 30 (1994), pp. 875–886.

[19] D. A. Huffman, *A method for the construction of minimum-redundancy codes*, Resonance, 11 (2006), pp. 91–99.

[20] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, S. Malla, H. Marriott, T. Nieto, J. O'Grady, H. E. Olsen, B. S. Pedersen, A. Rhie, H. Richardson, A. R. Quinlan, T. P. Snutch, L. Tee, B. Paten, A. M. Phillippy, J. T. Simpson, N. J. Loman, and M. Loose, *Nanopore sequencing and assembly of a human genome with ultra-long reads*, Nature Biotechnology, 36 (2018), pp. 338–345.

[21] K. Katz, O. Shutov, R. Lapoint, M. Kimelman, J. R. Brister, and C. O'Sullivan, *The sequence read archive: a decade more of explosive growth*, Nucleic Acids Research, 50 (2021), pp. D387–D390.

[22] M. Kchouk, J. F. Gibrat, and M. Elloumi, *Generations of sequencing technologies: From first to next generation*, Biology and Medicine, 09 (2017).

[23] W. J. Kent and D. Haussler, *Assembly of the working draft of the human genome with GigAssembler*, Genome Research, 11 (2001), pp. 1541–1548.

[24] J. M. Kidd, G. M. Cooper, W. F. Donahue, H. S. Hayden, N. Sampas, T. Graves, N. Hansen, B. Teague, C. Alkan, F. Antonacci, E. Haugen, T. Zerr, N. A. Yamada, P. Tsang, T. L. Newman, E. Tüzün, Z. Cheng, H. M. Ebling, N. Tusneem, R. David, W. Gillett, K. A. Phelps, M. Weaver, D. Saranga, A. Brand, W. Tao, E. Gustafson, K. McKernan, L. Chen, M. Malig, J. D. Smith, J. M. Korn, S. A. McCarroll, D. A. Altshuler, D. A. Peiffer, M. Dorschner, J. Stamatoyannopoulos, D. Schwartz, D. A. Nickerson, J. C. Mullikin, R. K. Wilson, L. Bruhn, M. V. Olson, R. Kaul, D. R. Smith, and E. E. Eichler, *Mapping and sequencing of structural variation from eight human genomes*, Nature, 453 (2008), pp. 56–64.

[25] N. Larsson and A. Moffat, *Offline dictionary-based compression*, in Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096), 1999, pp. 296–305.

[26] R. Leinonen, H. Sugawara, and M. S. et. al., *The sequence read archive*, Nucleic Acids Research, 39 (2010), pp. D19–D21.

[27] L. Liu, N. Hu, B. Wang, M. Chen, J. Wang, Z. Tian, Y. He, and D. Lin, *A brief utilization report on the illumina hiseq 2000 sequencer*, Mycology, 2 (2011), pp. 169–191.

[28] Y. Liu, Z. Yu, M. E. Dinger, and J. Li, *Index suffix–prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression*, Bioinformatics, 35 (2018), pp. 2066–2074.

[29] M. L. Metzker, *Sequencing technologies — the next generation*, Nature Reviews Genetics, 11 (2009), pp. 31–46.

[30] A. S. Mikheyev and M. M. Y. Tin, *A first look at the oxford nanopore MinION sequencer*, Molecular Ecology Resources, 14 (2014), pp. 1097–1102.

[31] N. B. Nsira, T. Lecroq, and M. Elloumi, *A fast boyer-moore type pattern matching algorithm for highly similar sequences*, International Journal of Data Mining and Bioinformatics, 13 (2015), p. 266.

[32] I. Numanagić, J. K. Bonfield, F. Hach, J. Voges, J. Ostermann, C. Alberti, M. Mattavelli, and S. C. Sahinalp, *Comparison of high-throughput sequencing data compression tools*, Nature Methods, 13 (2016), pp. 1005–1008.

[33] I. Numanagić, S. Malikic, M. Ford, X. Qin, L. Toji, M. Radovich, T. Skaar, V. Pratt, B. Berger, S. Scherer, and C. Sahinalp, *Allelic decomposition and exact genotyping of highly polymorphic and structurally variant genes*, Nature Communications, 9 (2018).

[34] L. Peltonen, *A map of human genome variation from population-scale sequencing*, Nature, 467 (2010), pp. 1061–1073.

[35] ——, *An integrated map of genetic variation from 1, 092 human genomes*, Nature, 491 (2012), pp. 56–65.

[36] A. Rhoads and K. F. Au, *PacBio sequencing and its applications*, Genomics, Proteomics & Bioinformatics, 13 (2015), pp. 278–289.

[37] Ł. Roguski, I. Ochoa, M. Hernaez, and S. Deorowicz, *FaStore: a space-saving solution for raw sequencing data*, Bioinformatics, 34 (2018), pp. 2748–2756.

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

[39] C. L. Schoch, S. Ciufo, M. Domrachev, C. L. Hotton, S. Kannan, R. Khovanskaya, D. Leipe, R. Mcveigh, K. O'Neill, B. Robbertse, S. Sharma, V. Soussov, J. P. Sullivan, L. Sun, S. Turner, and I. Karsch-Mizrachi, *NCBI taxonomy: a comprehensive update on curation, resources and tools*, Database, 2020 (2020).

[40] S. C. Schuster, *Next-generation sequencing transforms today's biology*, Nature Methods, 5 (2007), pp. 16–18.

[41] Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa, *A boyer—moore type algorithm for compressed pattern matching*, in Combinatorial Pattern Matching, Springer Berlin Heidelberg, 2000, pp. 181–194.

[42] L. J. Steinbock and A. Radenovic, *The emergence of nanopores in next-generation sequencing*, Nanotechnology, 26 (2015), p. 074003.

[43] R. Wan, V. N. Anh, and K. Asai, *Transformations for the compression of FASTQ quality scores of next-generation sequencing data*, Bioinformatics, 28 (2011), pp. 628–635.

[44] C. Wright, A. Rajpurohit, E. E. Burke, C. Williams, L. Collado-Torres, M. Kimos, N. J. Brandon, A. J. Cross, A. E. Jaffe, D. R. Weinberger, et al., *Comprehensive assessment of multiple biases in small rna sequencing reveals significant differences in the performance of widely used methods*, BMC genomics, 20 (2019), pp. 1–21.

[45] B. Yuan, P. Liu, A. Gupta, C. R. Beck, A. Tejomurtula, I. M. Campbell, T. Gambin, A. D. Simmons, M. A. Withers, R. A. Harris, J. Rogers, D. C. Schwartz, and J. R. Lupski, *Comparative genomic analyses of the human NPHP1 locus reveal complex genomic architecture and its regional evolution in primates*, PLOS Genetics, 11 (2015), p. e1005686.

[46] J. Ziv and A. Lempel, *A universal algorithm for sequential data compression*, IEEE Transactions on Information Theory, 23 (1977), pp. 337–343.

[47] M. C. Zody, Z. Jiang, H.-C. Fung, F. Antonacci, L. W. Hillier, M. F. Cardone, T. A. Graves, J. M. Kidd, Z. Cheng, A. Abouelleil, L. Chen, J. Wallis, J. Glasscock, R. K. Wilson, A. D. Reily, J. Duckworth, M. Ventura, J. Hardy, W. C. Warren, and E. E. Eichler, *Evolutionary toggling of the MAPT 17q21.31 inversion region*, Nature Genetics, 40 (2008), pp. 1076–1083.