Scuola di dottorato
Università DEGLI STUDI DI MILANO-BICOCCA

Dipartimento di Informatica, Sistemistica e Comunicazione

PhD program in Computer Science – Cycle XXXII

Algorithms for analyzing genetic variability from Next-Generation Sequencing data

Luca Denti
(750058)

Supervisor: *Prof. Paola Bonizzoni*
Co-supervisors: *Dott. Raffaella Rizzi, Dott. Marco Previtali*

Tutor: *Prof. Giuseppe Vizzari*

PhD Coordinator: *Prof. Leonardo Mariani*

ACADEMIC YEAR 2018/2019

To my granddad

Acknowledgements

First of all, I want to thank my mum for her support: she is the one who kept me going and helped me reaching the end of this journey.

I would like to thank my advisors, Paola, Raffaella, and Marco, for their guidance and assistance. I am extremely grateful to Marco who taught me how to approach research sharing with me a bit of its knowledge every day.

Special thanks go to my PhD colleagues and office mates Giulia, who always encouraged me, especially during the second year of my PhD, and Simone, who keeps me young by making me feel like a high school student.

My thanks go also to Gianluca, Stefano, Murray, Luca, and Mauricio for the useful discussions I had with them.

A special thank to Alexander Schönhuth who guided my research activity during my visit at the CWI.

Lastly, I would like to thank Stefano, Omar, and Laura: spending my spare time with you always makes me feel better.

Contents

Chapter 1

Introduction

The foundation of life mainly consists in a microscopic molecule, known as Deoxyribonucleic Acid (DNA). DNA contains the genetic information that is essential for the correct development of any organism. The slightest variation in its structure or in its functioning may lead to lethal diseases, such as tumors. Being able to investigate DNA molecules is of utmost importance for analyzing the reasons behind such disorders and improving the quality of life. Development of DNA sequencing technologies has revolutionized the way this kind of investigation is performed. Such technologies are fast and produce a lot of data that allow to better understand and analyze how DNA works.

Due to the huge amount of sequencing data available, nowadays computer science plays a key role in their analysis. Indeed, analyzing DNA from sequencing data in an efficient and effective way requires the development and application of computational approaches. This is one of the main goal of *bioinformatics*, a research field that combines computer science and biology. Luckily, in many applications, the biological information contained in a DNA molecule can be described as a sequence of nucleotides and it can be represented as a *string* in which each character represents a nucleotide. Strings are a well-known and well-studied notion in computer science and therefore it is possible to exploit the huge literature related to storing and processing strings for improving the analysis of DNA.

Sequencing technologies are those technologies that allow to translate a DNA molecule into a set of strings. Due to technical limitations, these technologies

are not able to produce a single string representing the entire DNA molecule, but they produce fragments, known as *reads*, of its nucleotide sequence. When the fragments come from a DNA molecule, they are called *genomic reads*, or simply reads, whereas when they come from a RNA molecule, they are known as *RNA-Seq reads*. The set of fragments produced by a sequencing technology is known as *sample*.

Currently the most used sequencing technologies are the so-called Next-Generation Sequencing (NGS) technologies. These technologies are cost-effective, efficient, and produce a huge amount of data, ranging from several millions to billions of short read fragments per sequencing experiment. Development of NGS technologies paved the way to numerous bioinformatics analyses and nowadays such technologies are routinely applied to clinical settings [147, 100] and a massive amount of sequencing data is produced every day [120]. Indeed, the analysis of NGS data allows to understand genetic diversity among the individuals of a population and to discover the causes behind diseases and tumors.

Within this context, this thesis focuses on two specific problems arising from the analysis of NGS data: the study of transcript variability due to alternative splicing and the investigation of genetic variability among different individuals due to small variations such as Single Nucleotide Polymorphisms and indels. Regarding both these problems, we investigate two novel computational approaches by devising original strategies and we prove their efficacy by comparing them with the most used state-of-the-art approaches.

The first problem we tackle is the detection of alternative splicing events from RNA-Seq data. Our contribution is an original algorithmic approach that exploits the novel notion of alignment against a splicing graph [11, 34].

The second problem we tackle is the genotyping of a set of known Single Nucleotide Polymorphisms and indels from NGS data. In this area, our contribution is the first alignment-free approach that is able to genotype SNPs, indels, and multi-allelic variants directly from the raw reads, *i.e.* without aligning them [33].

In both these areas, our focus is on the development of bioinformatics tools that combine accurate algorithms with efficient data structures. Efficiency and accuracy are crucial for bioinformatics tools but obtaining them requires to overcome arduous computational challenges. Firstly NGS reads are small fragments

of a donor DNA and being able to extract meaningful knowledge from them is complex since they do not offer a comprehensive view of the entire DNA sequence. Secondly raw NGS datasets may require hundreds of gigabyte of memory to be stored and their analysis is computationally intensive.

In the first part of this thesis, we introduce a novel algorithmic approach for the analysis of alternative splicing from RNA-Seq reads.

Alternative splicing is a regulatory process that increases the protein diversity by allowing a single gene to synthesize multiple proteins, each one with a different functionality. For example, in humans, almost 25,000 protein coding genes can generate more than 90,000 proteins [170]. Alternative splicing alters how the coding portions of a gene are combined together to produce the messenger RNA molecule, known as *transcript*, that contains the information needed for the synthesis of a protein. Alternative splicing plays an important role in many different life aspects, from the correct evolution of an individual [170] to the development of diseases [154]. Therefore, its analysis is of extreme importance for better investigating diseases and their causes.

Current bioinformatics approaches for the analysis of alternative splicing rely on the reconstruction of transcripts from NGS data or on the spliced alignment of NGS reads against a reference genome. Transcript reconstruction consists in identifying the structure and the nucleotide sequence of the transcripts expressed in an RNA-Seq sample. Current approaches for transcript reconstruction can be classified in reference-based [157, 56, 123] and de-novo [51, 143]. The former use the alignments of the reads to the reference genome to identify the transcripts whereas the latter build the transcripts directly from the reads, without aligning them. On the other hand, spliced alignment consists in identifying the most probable (possibly non-contiguous) coding regions along the reference genome from which each RNA-Seq read originates. Examples of spliced aligners are `TopHat2` [77], `STAR` [38], `BBMap` [19], and `HISAT` [75].

Differently from current techniques we investigate an alternative approach based on the alignments of NGS reads against a *splicing graph* [60], *i.e.* a graph representation of the known transcripts of a gene. We were motivated by our computer science interest in solving the open problem of designing an algorithmic approach based on succinct data structures for the approximate matching of a

string against a labeled graph. Aligning strings to a graph is a fascinating problem and, in the last years, it is drawing the attention of more and more researchers [76, 46, 127, 65, 66].

With this goal in mind, we propose an algorithm based on the FM-index data structure and we implemented it into `ASGAL` (Alternative Splicing Graph ALigner). `ASGAL` is a bioinformatics tool that aligns an RNA-Seq sample against the splicing graph of a gene and then detects the alternative splicing events supported by the sample by comparing the alignments with the gene annotation.

Detecting alternative splicing events task is adequate for many transcriptome analyses [163, 68, 146] and, differently from the more complex task of transcript reconstruction, it is computationally feasible. Moreover, by introducing the formalization of spliced alignment to a splicing graph, `ASGAL` performs an alignment step that is tailored to the identification of alternative splicing events. This allows `ASGAL` to detect alternative splicing events that are novel with respect to a gene annotation, *i.e.* events that cannot be described by two or more annotated transcripts. `ASGAL` is the first tool that aligns reads against a splicing graph and that is able to detect novel alternative splicing events even when only a single isoform per gene is supported by the sample.

The results of our experimental evaluation show the usefulness of aligning against a splicing graph and prove the ability of the proposed approach in detecting alternative splicing events.

In the second part of this thesis, we introduce a novel algorithmic approach for genotyping a set of known Single Nucleotide Polymorphisms (SNPs) and indels from NGS data.

Although the DNA sequences of two unrelated individual of the same population are similar, they differ in many aspects. Among the various differences that may occur, SNPs and indels are the most studied ones. A SNP is the substitution of a single nucleotide whereas an indel is the insertion or the deletion of one or more nucleotides.

Discovery and characterization of these kind of genomic variations are the main goals of *genome-wide association studies* (GWAS). Through an in-depth analysis of the variations occurring among the DNA sequences of different individuals, GWAS aim to understand genetic risks factors for diseases that are

common in the population under analysis [18].

Moreover, SNPs and indels are closely related to alternative splicing. Indeed, single nucleotide changes and indels affect gene splicing and can alter or inhibit gene function [62]. For example, SNPs close to the boundary between the coding and non-coding regions of a gene are one of the major cause behind alternative splicing [7, 81, 72].

Therefore, being able to analyze SNPs and indels from the data available, *i.e.* NGS data, is essential for a better understanding of genetic diversity and for discovering the causes behind diseases and tumors.

Standard pipelines for variant discovery and genotyping [101, 86] include read alignment, a computationally expensive procedure that is too time consuming for typical clinical applications. When variant discovery is not desired, it is possible to directly genotype a set of known variants completely avoiding the alignment step. Genotyping variants that are already known is a crucial task in clinical settings where it is necessary to know the genotype at specific loci that are already established to be of medical relevance. Luckily, many GWAS have been established in the last years [25, 26, 28, 45] and many datasets of known variants are available nowadays.

In this context, we propose an alignment-free approach for genotyping a set of known variants. We were motivated by the lack of accurate alignment-free approaches for genotyping indels and multi-allelic variants, *i.e.* variants for which more that two alleles have been observed in the studied population. Indeed, all the alignment-free approaches available in the literature restrict to the genotyping of single-allelic SNPs [118, 139, 152].

Therefore we devised a novel alignment-free algorithmic approach and we implemented it in a bioinformatic tool, called `MALVA`. Our approach combines an accurate way for characterizing the alleles of a variant with the use of an efficient data structure, *i.e.* a Bloom filter. More precisely, `MALVA` characterizes each allele of a variant with a set of k-mers, *i.e.* strings of length k, that are stored in a Bloom filter. Then it exploits a fast k-mer counting step to associate to each allele a weight. Finally, these weights are used for genotyping the variant. To do so, we investigated how the classic probabilistic framework used for genotyping single-allelic variants can be extended to multi-allelic variants. `MALVA` is the first

alignment-free approach that is able to genotype SNPs, indels, and multi-allelic variants.

Thanks to its alignment-free strategy, `MALVA` requires one order of magnitude less time than alignment-based pipelines to genotype a donor individual while achieving similar accuracy. Remarkably, on indels, `MALVA` provides even better results than the most widely adopted approaches.

Outline This thesis is organized as follows. In Chapter 2 we introduce the basic concepts necessary to understand the contents of the thesis. We present the computer science notions and the biological and bioinformatics concepts used through the thesis. In Chapter 3 we present the first contribution of this thesis, namely `ASGAL`, describing its algorithmic approach, its implementation, and the experimental evaluation we performed to validate it. In Chapter 4 we describe `MALVA`, the second contribution of this thesis. Similarly to the previous chapter, we describe the algorithmic approach of `MALVA` and the experiments we performed to evaluate it. Finally, in Chapter 5, we summarize the results and describe future developments of this work.

Chapter 2

Preliminaries

In this chapter we will introduce the basic concepts necessary to understand the main content of this thesis. This chapter is organized as follows. We first introduce the definitions of the computer science concepts that will be used through the rest of the thesis. Then we introduce several biological notions. We devote the final part of this chapter to bioinformatics.

We note that we defer the definition of some concepts to the next chapters, especially when they are closely related to the contribution there presented.

2.1 Strings and graphs

Strings are one of the most important and most used notion in computer science.

Let Σ be an *alphabet*, *i.e.* a finite ordered set of symbols, also called characters. A *string* of length n over alphabet Σ is a sequence of n symbols of Σ, *i.e.* $S = s_0 s_1 \dots s_{n-1}$ with $s_i \in \Sigma \; \forall 0 \leq i < n$.

Given a string S, $|S|$ denotes the length of the string, $S[i]$ with $0 \leq i < n$ denotes the i-th symbol of S, $S[i, j]$ with $0 \leq i \leq j \leq n-1$ denotes the *substring* of S starting at position i included and ending at position j included, $S[0, i]$ denotes the prefix ending at position i, and $S[i, n-1]$ denotes the suffix starting at position i. If $i > j$, $S[i, j]$ is the *empty string*, also denoted as ϵ.

A *rotation* (or *cyclic shift*) of a string S is the string obtained by concatenating a suffix of S with the remaining prefix. Formally, the i-th rotation of the

n-long string S is the string $S' = S[i, n-1] \cdot S[0, i-1]$, where $\cdot$ is the operator of *string concatenation*.

For example, given $\Sigma = \{\mathtt{I}, \mathtt{M}, \mathtt{S}\}$ and $S = \mathtt{MISSISSIPPI}$, $|S| = 11$, $S[1] = \mathtt{I}$, $S[2, 5] = \mathtt{SSIS}$ is a substring of S, $S[0, 3] = \mathtt{MISS}$ is a prefix, $S[4, 10] = \mathtt{ISSIPPI}$ is a suffix, and $S[4, 10] \cdot S[0, 3] = \mathtt{ISSIPPIMISS}$ is the 4-th rotation.

The *edit distance* is a metric to quantify the dissimilarity between two strings. It is defined as the minimum number of operations needed to transform one string into the other one. The allowed *(edit) operations* are: substitution of a character in the first string with a character of the second string, insertion of a character in the first string, and deletion of a character from the first string. Given two string X and Y, we denote the edit distance between them as $d_E(X, Y)$.

Another fundamental notion in computer science is the notion of *graph* that can be informally described as a structure that represents a set of elements and how they relate to each other. A *directed graph* G is a pair (V, E) where V is a finite set of vertices and $E \subseteq V \times V$ is a finite set of edges, *i.e.* ordered pairs of vertices. A graph is said *undirected* when its edges are unordered pairs of vertices. A vertex v is a *source* if it has no incoming edges whereas it is a *sink* if it has no outgoing edges. A *path* of length k from a vertex s to a vertex t is a sequence $\langle v_0, v_1, \ldots, v_k \rangle$ of $k+1$ vertices such that $v_0 = s$, $v_k = t$, and $(v_{i-1}, v_i) \in E$ for $0 < i \leq k$. The *length* of the path is the number of edges in the path. A *subpath* of a path is a contiguous subsequence of its vertices. A path $\langle v_0, v_1, \ldots, v_k \rangle$ is said a *cycle* if $v_0 = v_k$ and it contains at least one edge. A graph with no cycles is said *acyclic*, *cyclic* otherwise. A *tree* is an undirected acyclic graph (V, E) where $|E| = |V| - 1$.

Finally, we introduce the notion of *hypertext*. A hypertext is a directed graph (V, E) where each vertex is labeled by a string, also called *label of the vertex*. The label of vertex $v \in V$ is denoted by $\mathtt{seq}(v)$. A hypertext is said *acyclic* (*cyclic*) when the underlying graph is acyclic (cyclic).

2.2 Bit vectors

A bit vector B of length n is an array of n binary values, that is, $\forall i\, 1 \leq i \leq n, B[i] \in \{0, 1\}$. Alternatively, a bit vector of length n can be defined as a

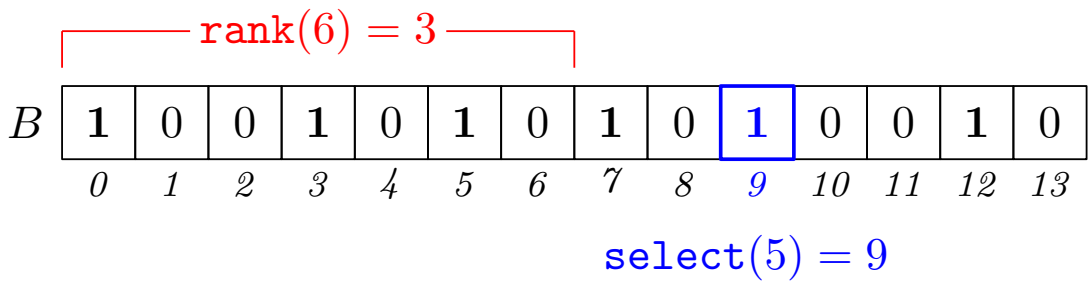


Figure 2.1: Example of `rank` and `select` functions over a bit vector B.

mapping from values in the range $\{0, \ldots, n-1\}$ to values in the set $\{0, 1\}$, *i.e.* from the positions of the array to the values stored in each position. The basic function is the `access` function that returns the element stored at a certain position. Jacobson [64] noticed that bit vectors are fundamental to support various data structures and he introduced two different functions, the `rank` and the `select` functions. See Figure 2.1 for an example. We will now define the two functions following the notation used in [165].

The `rank` function takes a position and returns the number of 1s up to that position, excluded. More formally:

$$\texttt{rank}_B(i) = \sum_{0 \leq j < i} B[j] \quad with \quad 0 \leq i < n$$

The `select` function, given an integer i, returns the position of the i-th 1 in the bit vector. More formally:

$$\texttt{select}_B(i) = \max\{j < n | \texttt{rank}_B(j) \leq i\} \quad with \quad 0 < i \leq \texttt{rank}_B(n)$$

The naive implementation of these functions consists in scanning the bit vector but, in the worst case, this requires $\mathcal{O}(n)$ operations. The two functions can be supported in $\mathcal{O}(1)$ using $o(n)$ additional bits. For a review of the different implementations proposed in the literature, we refer the reader to [52].

2.3 Bloom filters

A Bloom filter is a probabilistic space-efficient data structure that represents a set of elements and allows approximate membership queries. Bloom filters

were first introduced as a more space efficient alternative to conventional hashing techniques [14]. They are usually represented as the union of a bit vector of length m and a set of h hash functions $\{\mathtt{H}_1, \ldots, \mathtt{H}_h\}$, each one mapping one element of the universe to an integer in $\{0, \ldots, m-1\}$. Using these data structures, the addition of an element e to the set is performed by setting to 1 the bit vector's cells in positions $\{\mathtt{H}_1(e), \ldots, \mathtt{H}_h(e)\}$, while testing if an element is in the set boils down to checking whether the same positions are all set to 1. See Figure 2.2 for an example.

Due to collisions of the hash functions, an element can be reported as present in the set even though it is absent, resulting in a false positive. However, the result of a query can never be a false negative.

The false positive rate of a Bloom filter of a set of n elements, with h hash functions, and an array of m bits is $(1-(1-1/m)^{hn})^h$ [111] and it can be approximated as $(1-e^{\frac{hn}{m}})^h$ [160]. Therefore the more elements are added to the Bloom filter, the higher the false positive rate is and by increasing the size of the Bloom filter, it is possible to lower such rate. However as reported in [15, 22], this analysis underestimates the real false positive rate of a Bloom filter, but it still represents a good approximation, especially when considering large Bloom filters. Due to their simplicity and efficiency, Bloom filters have been applied in multiple areas of computer science, from cryptography [36, 79] to bioinformatics [102, 21].

2.4 Pattern matching

Let T and P be two strings, called *text* and *pattern*, respectively. The *pattern matching* problem consists in finding and/or locating the occurrences of the pattern P in the text T. "Finding the occurrences" means identifying if the pattern occurs in the text whereas "locating the occurrences" means reporting all the positions of T at which P occurs. For example, if $T = \mathtt{MISSISSIPPI}$, then the pattern $P = \mathtt{IS}$ occurs in T at positions 1 and 4.

The pattern matching problem is one of the fundamental problem in computer science [55] and it finds application in many different areas [144], from word processors to bioinformatics. Many algorithms were proposed to solve the pattern matching problem. To name a few: the comparison-based Boyer-Moore

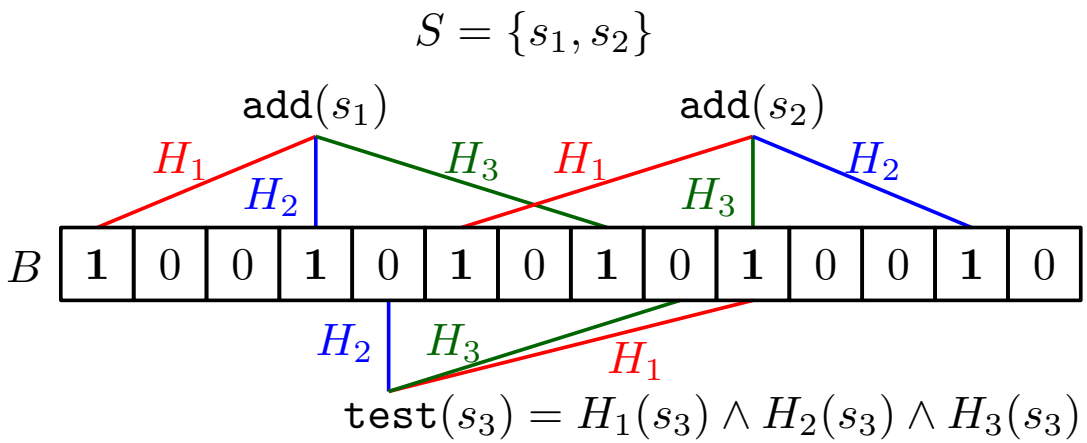


Figure 2.2: Example of Bloom filter using a bit vector B of size 14 and three hash functions $\{H_1, H_2, H_3\}$. The elements s_1 and s_2 of set S are added to the Bloom filter, whereas an element $s_3 \notin S$ is tested for membership.

algorithm [16], the bit-oriented *Shift-And* algorithm [8], and the randomized Karp-Rabin algorithm [71]. In this thesis we will focus our attention on index-based approaches.

These approaches rely on building an *index* of the input text and then searching the pattern over it. Differently from the aforementioned approaches, indexes allow to process the text only once and search for multiple patterns without processing the text multiple times. This is especially advantageous when the text is static, *i.e.* it does not change over time. For example, a reference genome in bioinformatics is a static text and for this reason index-based algorithms find many application in this research field.

In the first part of this thesis, we will tackle the problem of matching a pattern against a hypertext, that is a generalization of the classical pattern matching problem. Our contribution is an index-based approach that relies on the FM-Index and the algorithm proposed in [117] for computing Maximal Exact Matches (`MEMs`). For this reason, we will now describe two indices, namely the suffix array [93] and the FM-Index [43], the notion of Maximal Exact Match, and the approach proposed in [117] for computing the set of `MEMs` between two strings via a FM-Index.

2.4.1 Suffix arrays

Suffix arrays are efficient and compact data structures for storing and querying strings. They were introduced by Manber and Myers [93] to lower the space complexity of suffix trees [173], a tree-shaped data structure that stores all the suffixes of a string. For a detailed explanation of suffix trees, we refer the reader to [55].

Let T be a n-long string obtained as the concatenation of a $n-1$-long string built over the alphabet Σ and a special character \$ that does not belong to Σ and that is lexicographically smaller than any character in Σ. The suffix array of T, denoted by $\texttt{SA}_T$, is the ordered list of the starting positions of all its suffixes, lexicographically ordered. Formally, $\texttt{SA}_T$ is the permutation of the interval $[0, n-1]$ such that $\texttt{SA}_T[i] = j$ if and only if $T[j, n-1]$ is the i-th smaller suffix in the lexicographic order. Figure 2.3 shows an example of the suffix array of string `MISSISSIPPI$`.

The suffix array requires $\mathcal{O}(n \log n)$ bits of space and it allows to search for a m-long pattern P in $\mathcal{O}(m \log n)$ via binary search. All the *occ* occurrences of P can then be reported in an additional $\mathcal{O}(occ)$ time. The result of the binary search is an interval $[i, j]$ in the suffix array, called P-interval, that contains the starting positions of all the $occ = j - i + 1$ occurrences of P in T. For example, the `ISSI`-interval in the suffix array shown in Figure 2.3 is $[3, 4]$ and it refers to all the suffixes of P that have `ISSI`as prefix.

The time complexity for querying a suffix array can be lowered to $\mathcal{O}(m+\log n)$ by enhancing the suffix array with the so-called Longest Common Prefix (LCP) array [55]. The LCP array of string T, denoted by $\texttt{LCP}_T$, contains the length of the longest common prefixes between any two suffixes of T that are consecutive in the suffix array of T. Formally, given the n-long string T and its suffix array $\texttt{SA}_T$, $\texttt{LCP}_T$ is an array such that $\texttt{LCP}_T[0] = \texttt{LCP}_T[n] = -1$ and $\texttt{LCP}_T[i] = |lcp(T[\texttt{SA}_T[i-1], n], T[\texttt{SA}_T[i], n])$ for $0 < i < n$ where $lcp(\cdot, \cdot)$ denotes the longest common prefix between two strings. See Figure 2.3 for an example of LCP array.

In [1], Abouelhoda et al. show that any problem that can be solved with a suffix tree can also be solved with a suffix array. To this aim, they introduced the notion of *lcp-interval*, that is an interval in the LCP array that allows to simulate

i	$\texttt{SA}_T$	$\texttt{LCP}_T$	$\texttt{BWM}_T$	$\texttt{BWT}_T$
0	11	-1	$MISSISSIPPI	I
1	10	0	I$MISSISSIPP	P
2	7	1	IPPI$MISSISS	S
3	4	1	ISSIPPI$MISS	S
4	1	4	ISSISSIPPI$M	M
5	0	0	MISSISSIPPI$	$
6	9	0	PI$MISSISSIP	P
7	8	1	PPI$MISSISSI	I
8	6	0	SIPPI$MISSIS	S
9	3	2	SISSIPPI$MIS	S
10	5	1	SSIPPI$MISSI	I
11	2	3	SSISSIPPI$MI	I
12		-1		

Figure 2.3: Example of suffix array, LCP array, BWM, and BWT for string $T = \texttt{MISSISSIPPI\$}$.

any traversal of a suffix tree. Indeed, intuitively, any lcp-interval corresponds to an internal node of the suffix tree. Given the LCP array $\texttt{LCP}_T$ of an n-long string T and an integer l, the interval $[i, j]$ with $0 \leq i < j \leq n$ of $\texttt{LCP}_T$ is called an lcp-interval of lcp-value l, denoted by l-$[i, j]$, if and only if: (i) $\texttt{LCP}_T[i] < l$, (ii) $\texttt{LCP}_T[k] \geq l$ for $i < k \leq j$, (iii) $\texttt{LCP}_T[k] = l$ for at least one k with $i < k \leq j$, and (iv) $\texttt{LCP}_T[j + 1] < l$.

An lcp-interval l_2-$[i_2, i_2]$ is said to be embedded in an lcp-interval l_1-$[i_1, j_1]$ if $i_1 \leq i_2 \leq j_2 \leq j_1$ and $l_2 > l_1$. In other words, l_2-$[i_2, i_2]$ is embedded in l_1-$[i_1, j_1]$ if the substring of T identified by the interval $[i_1, j_1]$ is a prefix of the one identified by $[i_2, j_2]$. If l_1-$[i_1, i_1]$ embeds l_2-$[i_2, i_2]$ and there is no other lcp-interval embedded in l_1-$[i_1, i_1]$ that also embeds l_2-$[i_2, i_2]$, then l_1-$[i_1, i_1]$ is called the *parent interval* of l_2-$[i_2, i_2]$. We denote this relationship as the *parent* relationship. For example, in Figure 2.3, the lcp-interval 1-$[8, 11]$, related to string $\texttt{S}$, is the parent of the lcp-interval 3-$[10, 11]$ related to string $\texttt{SSI}$.

2.4.2 Burrows-Wheeler Transform and FM-Index

The Burrows-Wheeler Transform (BWT) of a string is a reversible permutation of its characters. The BWT was first introduced by Burrows and Wheeler in 1994 [17] as a lossless data compression algorithm: it reorders the characters of the input string and makes the new string "easier to compress". Indeed such reorganization of the string tends to gathers the occurrences of the same character together allowing compression algorithms, such as `bzip2` [138], to obtain better compression ratio [96].

Let T be a \$-terminated string of length n. The BWT of string T, denoted by $\texttt{BWT}_T$, is a string of length $|T|$ defined as follows: $\texttt{BWT}_T[i] = T[\texttt{SA}_T[i] - 1]$ for all i such that $\texttt{SA}_T[i] \neq 1$ and $\texttt{BWT}_T[i] = \$$ otherwise. Less formally, $\texttt{BWT}_T[i]$ is the character that precedes the i-th lexicographically smaller suffix of T. The BWT of string T can be obtained by building the so-called Burrows-Wheeler Matrix (BWM), that is the matrix built by lexicographically sorting all the rotations (cyclic shifts) of T. The last column of this matrix, read from top to bottom, corresponds to $\texttt{BWT}_T$. Figure 2.3 shows an example of BWT and BWM.

As stated previously, the BWT of a string is a reversible permutation. Indeed, given $\texttt{BWT}_T$, it is possible to reconstruct the original string T by means of the *LF-mapping* function: the i-th occurrence of a character c in the last column of the BWM (that is $\texttt{BWT}_T$) corresponds to the i-th occurrence of the same character in the first column. Moreover, since by construction the i-th character of the first column follows, in the original string T, the i-th character of the last column, it is possible to reconstruct the original string T starting from its last character, *i.e.* \$.

By exploiting the same technique, it is also possible to search for a pattern in T using its BWT, similarly to suffix arrays. Indeed, suffix arrays and BWM are strictly related: an interval on the suffix array corresponds to an interval on the rows of the BWM. However this time, instead of using a binary search over the suffix array , we can exploit the LF-mapping function. The algorithm that exploits the LF-mapping property to search for a pattern is known as **backward search** and it allows to find all the occurrences in time linear to the length of the pattern. The algorithm consists in scanning the pattern P from its last character

modifying, at each iteration, a suffix array interval: the $P[i-1,m]$-interval is obtained extending the $P[i,m]$-interval with the $P[i-1]$ character by means of the LF-mapping function.

The backward search can be efficiently performed by augmenting the BWT with two additional functions, $\texttt{C} : \Sigma \cup \{\$\} \rightarrow [1,n]$ and $\texttt{Occ} : \Sigma \cup \{\$\} \times [1,n] \rightarrow [1,n]$, obtaining the so-called FM-Index [43]. The first function (`C`), given a character $\sigma \in \Sigma$, returns the number of occurrences of characters lexicographically smaller than σ in T. This function is typically implemented as an array of integers of length $|\Sigma|$. The second function (`Occ`), given a character $\sigma \in \Sigma$ and a position i of $\texttt{BWT}_T$, counts the occurrences of σ in the first i elements of $\texttt{BWT}_T$. In other words, this function performs a `rank` operation over string $\texttt{BWT}_T$, that is it counts the occurrences of a certain character in a prefix of the string. It is typically represented by means of a Wavelet Tree [54]. A Wavelet Tree can be informally described as a tree of bit vectors that compactly stores a string and allows to efficiently perform `rank` queries over it. The interested reader can find more details about Wavelet Trees in [115].

Algorithm 1 shows how functions `C` and `Occ` can be used to perform a backward search step, *i.e.* how to compute the $P[i-1,m]$-interval starting from the $P[i,m]$-interval and $P[i-1]$. For example, the `SI`-interval $[8,9]$ in Figure 2.3 can be computed from the `I`-interval $[1,4]$ and character `S`.

Due to its efficiency and simplicity, the FM-index has been extensively used in the field of bioinformatics and it is the building block of many bioinformatics tools, most notably the read aligners `Bowtie2` [85] and `BWA` [88].

Algorithm 1 Computation of the cQ-interval given a character c and a Q-interval $[i,j]$

function *backwardStep*$(c,[i,j])$
  $i \leftarrow \texttt{C}[c] + \texttt{Occ}(c, i-1) + 1$
  $j \leftarrow \texttt{C}[c] + \texttt{Occ}(c, j)$
  if $i \leq j$ **then**
    return $[i,j]$
  else
    return $\perp$

2.4.3 Maximal Exact Matches

A Maximal Exact Match (`MEM`) is a common substring between two strings that cannot be extended in either directions without introducing a mismatch. Given two strings T and P of length n and m respectively, a Maximal Exact Match is a triple (t, p, ℓ) with $0 \leq t < n$ and $0 \leq p < m$ such that: (i) $T[t, t+\ell-1] = P[p, p+\ell-1]$, (ii) $p+\ell-1 = m$ or $t+\ell-1 = n$ or $T[t+\ell] \neq P[p+\ell]$, and (iii) $p = 0$ or $t = 0$ or $T[t-1] \neq P[p-1]$. If only the first (last) two conditions hold, the triple is called *right* (*left*) Maximal Exact Match. For example, given $S_1 =$ `MISSISSIPPI` and $S_2 =$ `MIPPISSI`, the set of Maximal Exact Matches of length at least 2 is $\{(0,0,2),(1,4,4),(4,4,4),(7,1,4)\}$, representing strings `MI`, `ISSI`, `ISSI`, and `IPPI`.

Computing `MEMs` between two strings is a widely studied problem in the literature [73, 117, 167, 42, 74, 91] and it finds many applications in bioinformatics. For example, `MEMs` are used to align reads to a genome [87, 166], to perform genome-to-genome alignment [97], and to correct sequencing errors in long reads [103].

In [117], Ohlebusch et al. presented a method for computing the Maximal Exact Matches between two strings by backward searching the second string against the FM-Index of the first one. Given a n-long string T and a m-long string P, the set of `MEMs` (t, p, ℓ) longer than a threshold L can be computed in $\mathcal{O}(m + z + occ \cdot t)$ where occ is the number of Maximal Exact Matches of length $\geq L$, z is the number of right Maximal Exact Matches of length $\geq L$, and t is the access time to the compressed index of T [117].

Since the approach we will describe in the next chapter, namely `ASGAL`, exploits the algorithm proposed by Ohlebusch et al., we will now briefly present it. However before describing it, we need to introduce the notion of *matching statistics*. Matching statistics were introduced by Chang and Lawler in [20] as a set of information to identify, for each position of a pattern, the longest substring starting at that position that matches a substring of a text. Formally, the matching statistics of P with respect to T is an array `ms` such that, for $0 \leq i < m$, `ms`$[i]$ is the pair $(l, [a, b])$ where: l is the length of the longest substring of P starting at position i, *i.e.* $P[i, i+l-1]$, that is a substring of T and $[a, b]$ is the

$P[i, i + l - 1]$-interval on the suffix array of T. The matching statistics between an n-long text T and an m-long pattern P can be computed in $\mathcal{O}(m)$ time by exploiting the FM-index of T [117].

We are now ready to describe the approach proposed by Ohlebusch et al. in [117] based on the backward search algorithm (see algorithm 2). The algorithm starts by scanning the pattern from its last character. For each position p_2 of P, the algorithm computes the substrings of P of length at least L ending at position p_2 and occurring in T. To do so, it computes the matching statistics $\texttt{ms}(p_2) = (q, [lb, rb])$ and stores the triple $(q, [lb, rb], p_2)$ in a list `path` if and only if q is greater or equal than L (see *lines 4-12*).

If the backward search returns an empty interval or the first character of P is reached, then the algorithm analyzes the list `path` (*lines 13-20*) and checks each substring for maximality. Each element of the list `path` is a triple $(q', [lb', rb'], p'_2)$ that represents a set of right Maximal Exact Matches of length q' starting at position p'_2 on P and at positions $[lb', rb']$ on T. Observe that right maximality is guaranteed by backward search: $[lb', rb']$ refers to $P[p'_2, p'_2 + k - 1]$ that, by definition of matching statistics, is the longest substring starting at position p'_2 that occurs in string T. Each right Maximal Exact Match is tested for left maximality (*line 17*) checking if the match can be extended on the left without introducing an error, *i.e.* testing if the character at position $t \in [lb', rb']$ on T is different from the character at position p'_2 on P. If the triple is left-maximal, then it is a Maximal Exact Match (*line 18*).

Finally (*lines 21-24*), when all the elements of list `path` have been processed, the algorithm reiterates the process. If the last analyzed interval $[i, j]$ corresponds to $[1, n]$, the algorithm reiterates considering the remaining prefix of P (*line 22*) since the interval cannot be extended by a backward search. Otherwise, the algorithm considers the embedding interval of $[i, j]$, *i.e.* the interval representing the longest common prefix of the substring identified by $[i, j]$. Observe that if a substring cannot be extended by backward search, one of its prefixes may be extended. We note that the algorithm ends its computation when it reaches the beginning of the pattern.

Algorithm 2 Computing `MEMs` of length greater or equal than L between two strings T and P of length n and m, respectively (Ohlebusch et al., 2010 [117])

```
1: p_2 ← m
2: (q, [i, j]) ← (0, [1, n])
3: while p_2 ≥ 1 do
4:     path ← []
5:     [lb, rb] ← backwardStep(P[p_2], [i, j])
6:     while [lb, rb] ≠ ⊥ and p_2 ≥ 1 do
7:         q ← q + 1
8:         if q ≥ L then
9:             add(path, (q, [lb, rb], p_2))
10:        [i, j] ← [lb, rb]
11:        p_2 ← p_2 − 1
12:        [lb, rb] ← backwardStep(P[p_2], [i, j])
13:    for each (q', [lb', rb'], p'_2) in path do
14:        [lb, rb] ← ⊥
15:        while q' ≥ L do
16:            for each k ∈ [lb', rb'] \ [lb, rb] do
17:                if p'_2 = 1 or BWT[k] ≠ P[p'_2 − 1] then
18:                    output (q', SA[k], p'_2)
19:            [lb, rb] ← [lb', rb']
20:            q'-[lb', rb'] ← parent([lb', rb'])
21:    if [i, j] = [1, n] then
22:        p_2 ← p_2 − 1
23:    else
24:        q-[i, j] ← parent([i, j])
```

2.5 Biological concepts

In this section we will give a high level description of the biological concepts that are needed to understand the main content of this thesis.

DNA and RNA

DNA (Deoxyribonucleic Acid) is the nucleic acid that carries the genetic information for the development, functioning, and reproduction of all living organisms. From a chemical point of view, a DNA molecule consists of a long double-chain of *nucleotides*. Each nucleotide is made up of a phosphate group, a sugar group (the deoxyribose), and a nitrogenous base, that is one of *Adenine* (`A`), *Cytosine* (`C`), *Guanine* (`G`), and *Thymine* (`T`). By convention, each chain, also called *strand*, starts with a phosphate group and ends with a sugar group and this orientation is known as *5'-to-3' direction*. The two strands of a DNA molecule, known as *forward strand* and *reverse strand*, run in opposite direction and they are held together by bonds between the bases. Following the *Watson-Crick base pairing* model [172], adenine pairs up with thymine through two hydrogen bonds and cytosine pairs up with guanine through three hydrogen bonds.

From a computational point of view, the primary nucleic acid sequence of a strand of a DNA molecule is a string built over the alphabet $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. When a strand is known, the other one can be inferred through the so-called *reverse-and-complement* operation: the sequence is reversed and each single base is complemented (`A` is converted into `T`, `T` into `A`, `C` into `G`, and `G` into `C`).

RNA (Ribonucleic Acid) is a molecule similar to DNA that is involved in many different tasks such as the regulation of gene expression. Differently from a DNA molecule, an RNA molecule is a single poly-nucleotide chain where each nucleotide is composed of a phosphate group, a sugar group (the ribose), and a nitrogenous base, that is one of *Adenine*, *Cytosine*, *Guanine*, and *Uracil* (`U`). Similarly to DNA, from a computational point of view, RNA can be represented as a string built over the alphabet $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$.

Genome, chromosomes, genes, and proteins

The DNA found in the nucleus of each cell of an organism is known as *genome* and it is organized into *chromosomes*. A *gene* is a particular region (or locus) of a chromosome that contains the genetic information that is passed from parents to offspring. Generally, a gene contains the information that is needed for the encoding of a *protein*, *i.e.* a molecule with a specific function. For example, a human genome is longer than 3 billion pairs of nucleotides (or base pairs, bps), it is organized into 23 pairs of chromosomes, 23 inherited from the mother and 23 from the father, and it contains from 20,000 to 25,000 protein-coding genes. The exact number of human genes is still unknown [133].

The mechanism by which the information contained in a gene is processed to produce a protein is divided in three steps: *transcription*, *splicing*, and *translation*.

In the *transcription* step, the primary sequence of the gene is copied into a pre-messenger RNA molecule. More precisely, one of the two strands of the gene locus, the one containing a particular subsequence of nucleotides, called *promoter*, is selected as the *coding strand* and an enzyme, the *RNA polymerase*, starts to transcribe the primary sequence of the locus on such strand into a pre-mRNA molecule by replacing each thymine nucleotide with an uracil nucleotide.

During the *splicing* step, the non-coding portions of the pre-mRNA, called *introns*, are spliced out from the pre-mRNA molecule. The result of this step is a messenger RNA (mRNA) molecule, also called *transcript* or *isoform*, that contains only the coding portions of a gene, called *exons*. The boundaries between exons and introns are known as *splice sites*. The first two bases of an intron are known as $5'$ (or donor) splice site whereas the last two bases are known as $3'$ (or acceptor) splice site. The most common introns in humans, for example, start with `GT` and end with `AG` [23].

Finally, in the *translation* step, a substring of the mRNA molecule, called *coding sequence*, is translated into a protein, *i.e.* a chain of amino acids. More precisely, each triplet of the coding sequence, called *codon*, is translated into an amino acid by means of the genetic code, see [137, Table 1.2]. The first codon of the coding sequence, called *start codon*, is always `AUG` and it indicates where the

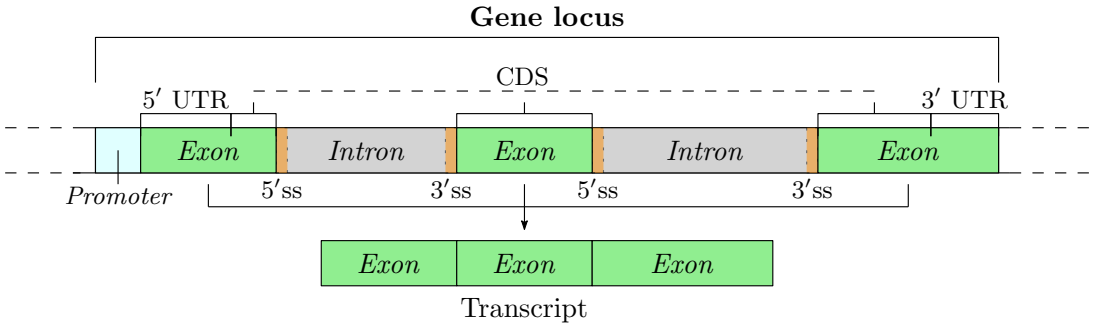


Figure 2.4: Structure of a simple gene with three exons and two introns. For ease of presentation, splice sites are indicated as *ss*.

translation starts. Translation ends when the last codon of the coding sequence, called *stop codon*, is reached. The stop codon is one among `UAA`, `UAG`, and `UGA` and it is not translated into an amino acid. The prefix and the suffix of the mRNA molecule that are not translated into a protein are known as 5′ UTR and 3′ UTR, respectively.

The structure of a gene is summarized in Figure 2.4.

Alternative Splicing

Alternative Splicing is a regulatory process that increases the complexity of gene expression by allowing a single gene to synthesize more than one protein. Due to this process, the 20,000/25,000 human protein-coding genes can synthesize more than 90,000 proteins [170]. It is estimated that more than 90% of human genes undergo alternative splicing [169, 119].

Alternative splicing occurs during the synthesis of a protein and alters how the introns are spliced out during the splicing step allowing different exons to be joined together. In such a way, the same gene can produce multiple transcripts that are translated into different proteins. The most common alternative splicing patterns, also called *alternative splicing events*, are: exon skipping, alternative acceptor site, alternative donor site, intron retention, and mutually exclusive exons. Each event allows the same gene to produce two different transcripts.

An *exon skipping* event occurs when one or more consecutive exons are included or skipped in the alternative isoform. This is the most common event in human [30]. An *alternative acceptor (donor) site* occurs when a different ac-

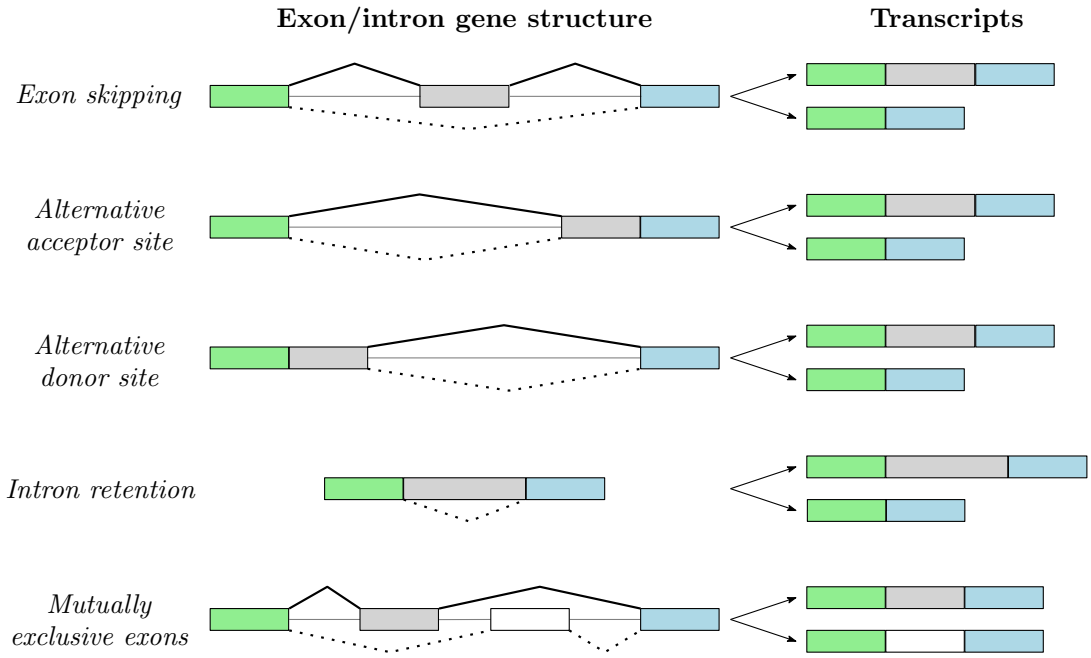


Figure 2.5: Alternative splicing events. Exons are shown as colored boxes whereas introns are represented as straight lines. Bold lines (solid and dotted) represent the splice sites that are supported by the two transcripts involved in the event.

ceptor (donor) splice site between an intron and an exon is used changing the starting (ending) position of an exon. Finally, an *intron retention* event occurs when an intron is retained or spliced out producing a different transcript whereas a *mutually exclusive exons* event takes place when two different exons are never included together in the same transcript. A graphical representation of these alternative splicing events is shown in Figure 2.5.

However alternative splicing is a completely unpredictable phenomenon and this classification is not adequate to fully capture its real complexity. Occasionally these "simple" alternative splicing events can occur simultaneously inside the same gene originating more complex splicing events. In this thesis we will focus our attention on the classic and more common alternative splicing events. The reader can find more details about complex events in [134, 44, 162].

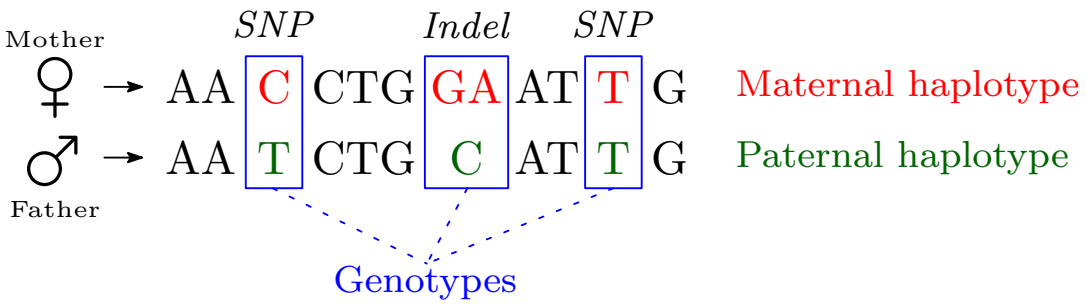


Figure 2.6: Example of Single Nucleotide Polymorphism (SNP), indel, haplotype, and genotype. The first two genotypes are heterozygous whereas the third one is homozygous.

Genomic variants

The differences occurring among the genomes of different individuals of a given specie are known as *genomic variants*. Genomic variants can involve either large or small regions of the genome. Genomic variants involving large portions of a genome are known as *structural variations*. Examples of structural variations are genomic duplications and genomic inversions.

In this thesis we will focus our attention on genomic variants that involve small regions of a genome, that are *Single Nucleotide Polymorphisms (SNPs)* and *indels*. SNPs are substitutions of a single base whereas indels are insertions or deletions of one or more consecutive bases.

An *allele* is one of the possible versions of a variant. For example, each allele of a SNP is a single nucleotide: at a specific position, the genomes of some individuals may have the allele `T` whereas the genomes of other individuals may have a different nucleotide. The set of alleles that are inherited from a parent is known as *haplotype*. Since humans are diploid organisms, they inherit two haplotypes, one from the mother and one from the father. The pair of alleles of a variant that occur in the two haplotypes at a specific position is known as *genotype* of the variant. The genotype is said *homozygous* when the two alleles are identical, *heterozygous* if the two alleles differ. We note that, although in the literature the term allele is used to describe the different forms of a gene inherited from the parents, in this thesis we will use the term allele to describe the different forms of a variant.

See Figure 2.6 for an example of genomic variants, haplotype, and genotype.

2.6 Bioinformatics

In this section we will describe the technologies used to sequence DNA and RNA and then we will give a high level description of the three bioinformatics problems tackled in this thesis that stem from the analysis of sequencing data. Finally we will describe the standard file formats used in bioinformatics presenting how the biological concepts introduced in this chapter can be stored as text files.

2.6.1 Sequencing technologies

DNA sequencing is the *"laboratory technique used to determine the exact sequence of bases (A, C, G, and T) in a DNA molecule"* [116]. Due to technical limitations, it is not possible to obtain the primary sequence of a whole DNA molecule but it is necessary to fragment the genome in small pieces and then sequence each piece independently. For this reason, the output of a sequencing experiment is not a single string representing the primary sequence of a DNA molecule, but it is a set of small strings, known as *reads*, that represent substrings of the primary sequence.

Similarly, RNA sequencing, also called Whole Transcriptome Shotgun Sequencing, is the technique used to determine the primary sequence of the transcripts expressed in a biological sample, *i.e.* a set of cells. The output of an RNA sequencing experiment is a set of *RNA-Seq reads*, that are portions of transcripts.

We will now give an overview of three different sequencing technologies proposed in the last 50 years: Sanger Sequencing, Next-Generation Sequencing, and Third-Generation Sequencing. For a more thorough and complete discussion of sequencing technologies, we refer the reader to the following reviews [84, 161, 59, 50, 145].

Sanger Sequencing

Sanger sequencing is the first sequencing technology proposed. It was developed by Frederick Sanger in 1977 [135] and it has been the most widely used approach for approximately 30 years. Sanger Sequencing was the technology used in the Human Genome Project [164], an international project with the goal of determining, for the first time, the sequence of a human genome.

Over the years, Sanger Sequencing has been improved and it can now produce reads of length up to 1000bps with a very low per-base error rate, ~ 0.001 [141]. However, the process is slow and expensive: it produces a maximum of approximately 6 Megabases per day and it costs on the order of $500 per Megabase [78].

Due to its inefficiency and expensiveness, in the last decades, Sanger sequencing has been partly supplanted by Next-Generation Sequencing (NGS) technologies.

Next-Generation Sequencing

Next-Generation Sequencing (NGS) [10] is a term used to describe a family of DNA and RNA sequencing technologies developed to lower the time and the cost required to perform a sequencing experiment and to increase the amount of data produced by a single run. Thanks to their efficiency and the high amount of data produced, NGS technologies are also known as *High-Throughput Sequencing* (HTS) technologies.

The first NGS technology was developed by 454 in 2005 [98] and since then many other NGS technologies have been proposed. The most widely-adopted NGS technology is the Illumina Sequencing Technology [59]. Illumina supports a variety of sequencing protocols, such as whole-genome sequencing and RNA sequencing, and provides many different sequencing machines, each one with its own capability and level of throughput. For example, the NextSeq machine can produce 400 million 2x150bp-long reads in less than 30 hours. For a more detailed description of the different available machines, we refer the reader to the Illumina website [1].

The biggest disadvantages of Illumina Sequencing technology are the read

[1] `https://www.illumina.com/systems/sequencing-platforms.html`

length and the error rate. Indeed, Illumina Sequencing can produce reads of length up to 300bps (against the 1000bps of Sanger Sequencing) with an error rate of $\sim 0.1\%$, that is approximately 100 times the error rate of Sanger Sequencing. Due to the short length, reads produced by NGS technologies are known also as *short reads*.

However, the advantages of Illumina technologies overcome their disadvantages. Indeed, thanks to their efficiency and cost-effectiveness and to the high quality of the data they produce, Illumina technologies (and in more general terms NGS technologies) are nowadays routinely being applied in clinical settings [147, 100] and they are supporting the spread of precision medicine [107, 106, 168].

Third-Generation Sequencing

Although their widespread availability and application, the main weakness of NGS technologies, *i.e.* the read length, stands out when it is necessary to analyze complex genomes, such as the human one. Indeed, many eukaryotic genomes contain very long repetitive regions and most of these repetitions exceed the length of the reads produced by NGS technologies. Therefore, data produced by NGS technologies do not allow to disambiguate such long repetitions.

To overcome this limitation, in the last years, Third-Generation Sequencing technologies have been introduced. These technologies produce very long reads, with length of several kilobases, and allows to resolve the long repetitions that generally occurs in eukaryotic genomes. The applications that may benefit of very long reads are not restricted to the resolution of long repetitions. Indeed, long reads may also facilitate the tasks of gene isoform identification and variant phasing [95]. Currently, the two dominant Third-Generation Sequencing technologies are Pacific Biosciences and Oxford Nanopore Technology.

As the previous sequencing technologies, also Third-Generation Sequencing technologies have both advantages and disadvantages. As previously stated, Third-Generation Sequencing technologies are able to produce very long reads. Read length mainly depends on the chosen technology and the library preparation and varies from 1kbps to several thousand kilobases [67]. In December 2017, the first read longer than 1 Megabase pairs was produced using Oxford

Nanopore sequencing technology [113].

The disadvantages of Third-Generation Sequencing technologies are their cost and the error rate. Indeed, Third-Generation Sequencing is more expensive and has a lower throughput than Next-Generation Sequencing and its average error rate varies from 11% to 14% [125]. Such a high error rate makes the analysis of this kind of data very challenging. To overcome this limitation, in the last years, many approaches have been proposed to correct the long reads using the highly-accurate reads produced by NGS technologies [80, 132, 103].

Although their disadvantages, Third-Generation Sequencing technologies are currently under active development and they have the potential to facilitate the next major advancements in medical genetics [95].

2.6.2 Processing of NGS data

We now give a high level description of the three bioinformatics problems tackled in the following chapters. These problems stem from the analysis of NGS data: they takes as input a read sample produced by an NGS technologies and they aim to extract some knowledge from it.

For an in-depth analysis of the approaches proposed in the literature to solve these problems, we refer the reader to the next chapters.

Read alignment

Read alignment is one of the fundamental and most studied problem in bioinformatics and it is the first step of many bioinformatics pipelines. Read alignment consists in identifying the most probable location along a reference genome from which each read of an input sample originated, *i.e.* was sequenced from.

Although this problem could look simple, there are many challenges that must be overcome. First of all, the reference genome is much longer than the reads. Secondly, reads are not perfect due to sequencing errors. Finally, reads may span genomic variations.

We can identify two types of read alignment: genomic read alignment and RNA-Seq read spliced alignment.

The problem of aligning genomic reads to a reference genome usually boils

down to mapping each read to a contiguous region of the genome. Examples of read aligners are `BWA` [88] and `Bowtie2` [85].

On the other hand, aligning RNA-Seq reads against a reference genome is a harder problem. RNA-Seq reads are sequenced from gene transcripts and, for this reason, they may span multiple exons of a gene. Therefore, spliced aligners must be able to align an RNA-Seq read to non contiguous regions of the reference genome, *i.e.* two or more coding regions separated by long non coding regions. Examples of spliced aligners are `STAR` [38], `TopHat2` [77], `BBmap` [19], and `Hisat` [75].

For a detailed description of spliced alignment and the `STAR` tool, we refer the reader to Chapter 3.

Prediction of alternative splicing events

The advent of Whole Transcriptome Shotgun Sequencing technologies aided biologists and geneticists in better understanding the relationship between alternative splicing and diseases [159]. One of the bioinformatics problems that can help to shed more light on this critical topic is the analysis of alternative splicing events from RNA-Seq samples.

In this thesis we will use the term *differential analysis of alternative splicing (events)* to indicate the task of analyzing multiple RNA-Seq samples describing different conditions to quantify alternative splicing events and their variability among the tested conditions. On the other hand, we will use the term *alternative splicing events detection* to indicate the task of identifying and quantifying the alternative splicing events that are supported by a single RNA-Seq sample. Both these tasks include a step of *quantification*, that consists in estimating how well each event is supported by the input RNA-Seq data.

For a more thorough description of the various tools proposed in the literature for analyzing alternative splicing events, we refer the reader to Chapter 3.

Variant Calling

Variant calling (or variant genotyping) is the task of computing the genotype of all the variants supported by an input NGS sample. Along with read alignment,

variant calling is the second main task performed in many genome-wide association studies, *i.e.* studies that analyze the genomes of multiple individuals of a population to identify genomic variants and their association with phenotypic traits, such as diseases.

When a dataset of variants produced by this studies is available, it is possible to limit the computation to genotype those variants only. We will refer to this task as *genotyping (or calling) of known variants*. On the other hand, when the set of variants is not known a priori, the variants must be first discovered and then genotyped. We will refer to this task as *variant discovery*.

For an in-depth analysis of the various approaches proposed for variant calling, we refer the reader to Chapter 4.

2.6.3 Standard file formats

All the biological notions introduced in this chapter have a computer science representation and they can be easily stored as text files. We will now briefly describe the standard file formats used in bioinformatics and mentioned in the next chapters.

FASTA/Q format As previously stated, from a computational point of view, a genome is a string built over the alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, where each character represents a nucleotide. By using the same formalism, it is also possible to represent the reads produced by any sequencing technology.

Typically, the reference genome of a given specie, *i.e.* the best assembled genome available for that specie, is stored as a FASTA file [174]. FASTA is a text-based format for representing nucleotides sequences. An entry in a FASTA file is divided in two parts: (i) a header, starting with the ">" character and containing a unique identifier and additional information and (ii) the string representing the sequence.

A read sample instead is typically stored as a FASTQ file [24], an extension of the FASTA format. In a FASTQ file, each read sequence is described by 4 lines: a header starting with the "@" character; the string representing the sequence; a separator line, usually a "+"; and a string representing the numeric quality score associated with each nucleotide of the sequence. The quality scores are Phred

quality scores and are encoded using ASCII characters. For more information about FASTQ file format, we refer the reader to [24].

GTF format From a computational point of view, the structure of a gene can be described as a set of features that occur at a specific location along the reference genome.

Typically, gene structures are stored using the GTF file format [47]. In a GTF file, each feature of a gene is represented as a tab-delimited list of 9 fields: the name of the chromosome where the feature is located; an identifier of the source of the feature; the feature type (*e.g.* gene, transcript, exon, intron, UTR...); the starting and the ending positions of the feature on the specified chromosome; a score representing the confidence of the source; the strand of the feature; the frame, indicating which base of the feature is the first base of a codon; and an optional set of additional information. The information stored in a GTF file is usually called *gene annotation*.

VCF format From a computational point of view, a genomic variant, such as a SNP or an indel, can be described as a set of alleles that may occur at a given position along a reference genome. Additionally, the description of a variant can be extended with information about its genotype with respect to a certain individual.

Typically, a set of variants identified in an individual or in a population of individuals is stored using the variant call format (VCF) [31]. A VCF file consists of a header section and a data section. The header section contains information about the set of variants stored in the file and describes semantically and syntactically the fields used in the data section. The data section describes each variant using one line composed of 8 mandatory fields and several optional fields. The mandatory fields are: the name of the chromosome on which the variant occurs; the position of the variant on the given chromosome; the variant identifier; the reference allele of the variant; the set of its alternate alleles observed in the considered population; a quality score; information about which filters the variant has passed; and a list of additional attributes. The optional fields are used to indicate the genotype of the variant (and additional information such as

the genotype quality) for each considered sample, *i.e.* individual of the analyzed population.

SAM format The Sequence Alignment/Map (SAM) format [89] is the de-facto standard format for storing the alignments of a set of reads against a reference genome. A SAM file is divided in two parts: a header section containing general information about the file, such as the format version used and the set of reference sequences, and an alignment section. Each alignment is described by a single tab-separated line composed of 11 mandatory fields and several optional fields containing additional information. The mandatory fields are: the identifier of the read; a binary flag describing the alignment; the reference sequence to which the read aligns; the position on the reference sequence at which the read has been aligned; the mapping quality; an extended CIGAR string describing how the read aligns; three information about the read mate; the read sequence; and the alignment quality.

Chapter 3

Alternative Splicing Events Detection

In this chapter we will discuss the first contribution of this thesis, that is an approach for detecting alternative splicing events by aligning RNA-Seq reads to a splicing graph (`ASGAL`, Alternative Splicing Graph ALigner). The problem tackled can be formulated as follows:

Input:	RNA-Seq sample and gene annotation
Output:	alternative splicing events supported by the sample that are novel with respect to the annotation

Highlights

- `ASGAL` is the first tool that aligns RNA-Seq reads against a splicing graph
- `ASGAL` is able to detect alternative splicing events even when only a single isoform involved in the event is supported by the input sample
- aligning reads to a splicing graph is a valid alternative to aligning reads to a reference genome

Outline This chapter is organized as follows. In Section 3.1 we introduce the context and the motivations behind our work. In Section 3.2 we present

some preliminary definitions and the first contribution of this chapter, that is the formal definition of *spliced graph-alignment*. In Section 3.3 we thoroughly detail the proposed approach, that is the second contribution of the chapter. In Section 3.4 we describe the experimental evaluation we performed to asses the performance of our tool, comparing it with state-of-the-art approaches. Finally, in Section 3.5 we draw conclusions and sketch possible future directions of this work.

3.1 Context and Motivations

Typical new sequencing technologies experiments produce millions, or even billions, of reads [61]. Although the amount of transcriptomic sequencing data is smaller compared to the genomic one, the problem of aligning RNA-Seq reads to a reference genome is much more complicated than that of mapping genomic reads to the same reference. Indeed, RNA-Seq reads may reflect the biological process of alternative splicing and each read may span two or more coding regions (exons) that are separated by very large non-coding regions (introns). The analysis of RNA-Seq data can help to shed light on the diversity of transcripts that results from *alternative splicing*.

Computational approaches for transcriptome analysis from RNA-Seq data can be classified according to two primary goals: detection of alternative splicing events and reconstruction of full-length transcripts [51, 143, 157, 56, 123, 156]. While reconstructing full-length isoforms from RNA-Seq data is a computationally intensive task, detecting alternative splicing events from RNA-Seq data is more straightforward and computationally feasible.

Various tools have been proposed in the literature for solving this latter task. Most of these approaches focus their attention on the differential analysis of alternative splicing from RNA-Seq data, *i.e.* they identify alternative splicing events that are differentially expressed among different RNA-Seq samples (replicates). Some approaches, such as `SplAdder` [69] and `rMATS` [140], base their prediction on the analysis of the spliced alignments of the input RNA-Seq reads whereas others, such as `SUPPA2` [158], use the transcript abundances estimated from RNA-Seq data. The quality of the transcript quantification or the read

spliced alignment is essential for obtaining good results and it may significantly affect the efficacy of such tools.

With the goal of detecting alternative splicing events supported by an RNA-Seq sample that may be used to enrich a gene annotation, we investigate an alternative approach that directly aligns the RNA-Seq reads against a splicing graph. Intuitively, a splicing graph [60] is a compact way to represent a gene annotation: it is a directed acyclic graph where each exon of a gene is represented as a vertex and each transcript, *i.e.* a sequence of exons, as a path.

The motivation of our proposal is that, by using the splicing graph, we are able to focus the alignment step on enriching the gene annotation with alternative splicing events that produce novel isoforms with respect to the already annotated ones. Indeed, using a non-flat representation of a gene structure can help in revealing details on how a read covers the known isoforms and the known splice sites of the gene.

For this purpose, we implemented `ASGAL` (Alternative Splicing Graph ALigner), a tool for detecting alternative splicing events through an accurate splice-aware alignment of RNA-Seq reads against the splicing graph of a gene of interest. Currently, there are several tools for the alignment of RNA-Seq reads against a reference genome [19, 38, 77] or a collection of transcripts [148] but, to the best of our knowledge, `ASGAL` is the first tool specifically designed for aligning RNA-Seq data directly to a splicing graph.

By mapping the input sample against the splicing graph of a gene of interest, `ASGAL` is able to detect the alternative splicing events that are novel with respect to the input gene annotation. More precisely, `ASGAL` extracts the introns supported by the read alignments computed against the splicing graph and compares them against the input annotation to detect whether novel events may be predicted from the input reads. From this perspective, differently from tools for event detection based on differential analysis, `ASGAL` is able to detect a novel event even when this event is supported by reads coming from a single unannotated isoform.

The most similar tool to `ASGAL` is `SpliceGrapher` [130]. Indeed, the goal of `SpliceGrapher` is to generate and visualize splicing graphs starting from a gene annotation, the spliced alignments of RNA-Seq data, and (optionally) the

alignments of EST data. Exploiting such information, `SpliceGrapher` is able to visualize a splicing graph enriched with alternative splicing events supported by the input samples. However `SpliceGrapher` computes only alternative splicing statistics and, for what concern the analysis of alternative splicing events, it is not easy to retrieve from its output the list of alternative splicing events supported by the input samples.

To model the problem of aligning an RNA-Seq reads against the splicing graph of a gene, we introduce the computational problem of matching a pattern against a hypertext in an approximate way and allowing gaps. This problem consists in matching an input pattern to a path (or subpath) of the hypertext allowing *errors* that model sequencing errors and *gaps* that model the presence of alternative splicing events.

The pattern matching to a hypertext problem was originally introduced by Manber and Wu [94] and then attacked by many researchers [2, 121, 6, 114]. In [155], the first algorithm that employs succinct data structures was proposed to solve the exact pattern matching in a hypertext problem. However no solution using succinct data structures was proposed to solve the approximated version of the problem.

Following this line of research, we introduce a novel problem formulation, *i.e.* the approximate pattern matching to a hypertext with gaps that we call *gap graph-alignment*, and we propose a novel approach that uses succinct data structure to solve it. We implemented our approach in the `ASGAL` tool and we performed an experimental evaluation on simulated and real data to assess its efficacy in aligning RNA-Seq reads and in detecting alternative splicing events.

In the first part of our experimental analysis, we compare the alignment step of `ASGAL` with `STAR` [38], one of the best-known and most used spliced aligner. The results show a good accuracy of `ASGAL` in producing correct alignments by directly aligning the RNA-Seq reads against the splicing graph of a gene.

Although `ASGAL` works under different assumptions than other existing tools for the differential quantification of alternative splicing events, we decided to compare `ASGAL` with `SplAdder`, `rMATS`, and `SUPPA2`. However we must note that, differently from the considered tools, the current implementation of `ASGAL` is not able to detect the insertion of novel exons inside an intron and intron

retention events caused by the union of two exons.

For this purpose we performed two distinct experimental analysis on simulated data to assess the efficacy of `ASGAL` in detecting alternative splicing events. In the first one, we evaluate the accuracy of the tools in predicting novel alternative splicing events, *i.e.* events that are not already described by some transcripts in the annotation. In the second analysis, we assess the accuracy of the considered tools in detecting alternative splicing events that are already present in the input annotation and are supported by the RNA-Seq sample.

We also ran an experimental analysis on real data with the goal of evaluating the ability of `ASGAL` in identifying RT-PCR validated alternative splicing events.

The results obtained in the simulated scenario show that `ASGAL` is the most accurate tool for predicting alternative splicing events that are supported by an RNA-Seq sample and that are novel with respect to the input annotation. The results on real data instead show the ability of `ASGAL` in detecting RT-PCR validated alternative splicing events even when they are simulated as novel events with respect to input gene annotation.

3.1.1 State of the Art

We will now review more in detail the tools we used in our experimental evaluation. We first review a spliced aligner and then we describe different approaches proposed in the literature for quantifying alternative splicing events.

As described in Chapter 2, RNA-Seq reads are sequenced from the transcripts of a gene and therefore, most of the times, they cannot be aligned on contiguous portion of a reference genome: there are long regions, *i.e.* the introns of the genes, that must be skipped. For this reason, to align RNA-Seq reads to a reference genome, it is necessary to perform a *spliced alignment*, that consists in aligning the read to non-contiguous portion of the reference, *i.e.* the exons of the genes. This problem is much more complicated than normal read alignment. Indeed, in addition to manage the presence of sequencing errors or genomic variations, a spliced aligner must be able to find splice sites and detect where a read must be split. To better perform this latter task, spliced alignment is usually guided by a gene annotation.

One of the most used and best known spliced aligner is `STAR`, Spliced Transcripts Alignment to a Reference [38]. `STAR` uses uncompressed suffix arrays and exploits the seed-and-extend paradigm to align RNA-Seq reads to a reference genome: it computes Maximal Mappable Prefixes between the reference and the read and then it stitches together near Maximal Mappable Prefixes to build the alignment. A Maximal Mappable Prefix is similar to a Maximal Exact Match and represents a substring of the read starting at a given position that matches exactly one or more substrings of the reference genome. `STAR` computes Maximal Mappable Prefixes via searches in the uncompressed suffix array of the reference genome. Maximal Mappable Prefixes represent exact matches between the reference genome and the read and they are successively clustered based on their proximity and stitched using a local alignment scoring scheme. Thanks to this second step, *i.e.* the stitching of Maximal Mappable Prefixes, `STAR` is able to align the read taking into account the possible presence of errors or splice sites.

Spliced alignment is typically the first step of procedures for analyzing gene expression and detecting alternative splicing events from RNA-Seq data.

For example, `SplAdder` [69] identifies and quantifies alternative splicing events starting from a given gene annotation and the spliced alignments of one or more RNA-Seq samples. `SplAdder` uses the gene annotation to builds a splicing graph and enriches it with novel vertices and novel edges by exploiting the splicing information contained in the input spliced alignments. Then, it analyzes this enriched graph and it uses the novel elements previously added to the graph to detect and quantify the alternative splicing events supported by the input samples. Optionally, it allows to perform differential analysis between different samples. Moreover, to simplify the analysis of its output, `SplAdder` also provides a set of functions to visualize splicing graphs and read coverages.

`rMATS` [140] is another popular tool for the analysis of alternative splicing from the spliced alignment of RNA-Seq reads. `rMATS` implements a statistical method that is specifically designed for the detection of differential alternative splicing events from replicate RNA-Seq data. It also allows to perform a statistical test to assess whether a difference in the isoform ratio of a gene between two conditions is significant with respect to a given threshold.

Differently from the previous approaches, `SUPPA2` [158] starts its analysis from

transcript quantification and not from spliced alignments. Similar to `rMATS`, it infers differential alternative splicing events across multiple conditions. `SUPPA2` generates from the input annotation the alternative splicing events and then it uses the pre-computed abundances of each transcript, expressed in transcript per million units, to quantify these events in terms of proportion spliced in (psi) for each sample. The differential splicing is given in terms of the differences of these relative abundances for each condition. However since `SUPPA2` quantifies only the alternative splicing events generated from the input annotation, it is not able to quantify novel events that may be supported by the samples.

Most recent tools, `MAJIQ` [162] and `LeafCutter` [90], are not limited to detecting "classic" alternative splicing events. They analyze RNA-Seq spliced alignments to quantify the more complex alternative splicing events that can be intuitively described as the combination of classic alternative splicing events.

`MAJIQ` analyzes RNA-Seq data and a set of (annotated) transcripts to quantify the relative abundances of a set of Local Splicing Variations, that are a generalization of classic alternative splicing events: for example, Local Splicing Variation allows to model the combination of an alternative acceptor site event and an alternative donor site event in different exons. More precisely, for each considered gene, `MAJIQ` builds its splicing graph from the input annotation and augments it with novel edges and splice sites supported by the input spliced reads. Then, from this *augmented* splicing graph, `MAJIQ` lists and quantifies the Local Splicing Variations that are well supported by the input alignments. Finally, an additional package (`Voila`) can be used to interactively visualize the output of `MAJIQ`, *i.e.* splicing graphs, Local Splicing Variations, and their quantification.

Similarly, `LeafCutter` analyzes RNA-Seq data and quantifies differential intron usage across samples, allowing the detection of novel introns which model complex alternative splicing events. Starting from the spliced alignments of RNA-Seq samples, `LeafCutter` retrieves all the reads spanning an intron and detects the set of introns supported by the samples. Then, it clusters introns that overlaps each other and builds a graph for each cluster where each vertex represents an intron and each edge links two introns that share a splice site. Finally, by analyzing the connected components of this graph that represent two or

more introns involved in the same alternative splicing events, `LeafCutter` is able to identify both classic alternative splicing events and Local Splicing Variations.

Differently from the previous approaches that base their analysis on RNA-Seq read alignment or transcript quantification, `KisSplice` [131] relies on local transcript assembly and it can be used even when a reference genome or a reference transcriptome is not available. Indeed `KisSplice` builds a *de Bruijn graph* [32], *i.e.* a graph where each vertex is a string of length k called k-mer, from the input reads and then analyze it to detect alternative splicing events. It relies on the idea that alternative splicing events are local variations between two transcripts that may be recognized as *bubbles* inside the de Bruijn graph, *i.e.* two distinct paths that share the starting and ending vertices. In such a way, without reconstructing the entire transcripts expressed by a sample, `KisSplice` is able to detect only the variable regions of the transcripts and it can use them to infer alternative splicing events.

However not all the tools for the analysis of alternative splicing events use RNA-Seq data. `AStalavista` [44] is a popular tool for the exhaustive extraction and visualization of alternative splicing events from gene annotations. This tool does not require RNA-Seq data as input but only a gene annotation and it lists all the alternative splicing events occurring between each pair of annotated transcripts. Similarly to `MAJIQ` and `LeafCutter`, `AStalavista` does not focus only on classic alternative splicing events but rather it uses a flexible coding of alternative splicing events [134] that can be used to model any kind of variation in the splicing structure of a gene, such as Local Splicing Variations.

3.2 Preliminaries

From a computational point of view, a genome is a string drawn from an alphabet of size 4 ($\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$). A gene is a *locus* of the genome, that is, a gene is a substring of the genome. Exons and introns of a gene are uniquely identified by their starting and ending positions on the genome. A transcript T of gene G is a sequence $\langle [a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n] \rangle$ of exons, where a_i and b_i are respectively the *start* and the *end* positions of the i-th exon of the transcript. Observe that a_1 and b_n are the *starting* and *ending positions* of transcript T, and each

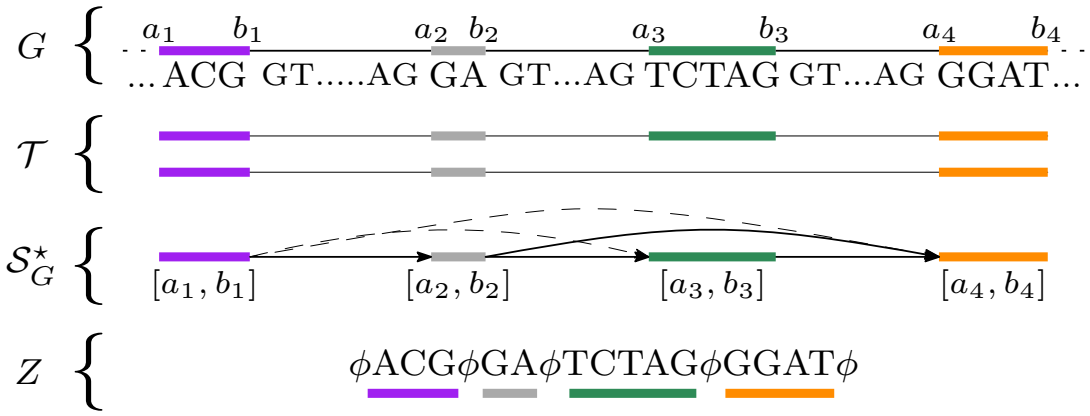


Figure 3.1: Example of splicing graph. A simple gene G with 4 exons is shown along with its annotation (transcripts) $\mathcal{T}_G$, the corresponding splicing graph $\mathcal{S}^\star_G$, and the linearization Z. In $\mathcal{S}^\star_G$, dashed arrows represent the novel edges whereas full arrows represent the edges contained in $\mathcal{S}_G$.

$[b_i + 1, a_{i+1} - 1]$ is an *intron* represented as a pair of positions on the genome. In the following, we denote by $\mathcal{E}_G$ the set of all the exons of the transcripts of gene G, that is $\mathcal{E}_G = \cup_{T \in \mathcal{T}_G} \mathcal{E}(T)$, where $\mathcal{E}(T)$ is the set of exons of transcript T and $\mathcal{T}_G$ is the set of transcripts of G, called the *annotation* of G. Given two exons $e_i = [a_i, b_i]$ and $e_j = [a_j, b_j]$ of $\mathcal{E}_G$, we say that e_i and e_j are *distinct* if $a_i \neq a_j \vee b_i \neq b_j$ and we say that e_i *precedes* e_j if $b_i < a_j$. We denote this by $e_i \prec e_j$. Moreover, we say that e_i and e_j are *consecutive* if there exists a transcript $T \in \mathcal{T}$ and an index k such that $e_k = e_i$ and $e_{k+1} = e_j$, and e_i, e_j are in $\mathcal{E}(T)$.

The *splicing graph* of a gene G is the directed acyclic graph $\mathcal{S}_G = (\mathcal{E}_G, E)$ where the vertex set is the set of exons of G and the edge set E is the set of pairs (v_i, v_j) such that v_i and v_j are consecutive in at least one transcript. We note that a splicing graph is a hypertext. Indeed, each vertex v is associated to a string, *i.e.* the genomic sequence of the exon associated to it. We denote this string by $\texttt{seq}(v)$. Finally, we say that $\mathcal{S}^\star_G$ is the graph obtained by adding to $\mathcal{S}_G$ all the edges $(v_i, v_j) \notin E$ such that $v_i \prec v_j$. We call these edges *novel* edges. Note that the novel edges represent putative novel junctions between two existing exons that are not consecutive in any transcript of G. Figure 3.1 shows an example of gene, exon, annotation, and splicing graph.

In the following, we will use the notion of *Maximal Exact Match* (MEM) to perform the spliced graph-alignment of an RNA-Seq read to $\mathcal{S}_G$. Given two strings R and Z, a MEM is a triple $m = (i_Z, i_R, \ell)$ representing the common substring of length ℓ between the two strings that starts at position i_Z on Z, at position i_R on R, and that cannot be extended in either direction without introducing a mismatch. For a more detailed definition of Maximal Exact Match, we refer the reader to Chapter 2. Computing the MEMs between a string R and a splicing graph $\mathcal{S}_G$ can be done by concatenating the labels of all the vertices and placing the special symbol ϕ before each label and after the last one, obtaining a string $Z = \phi\texttt{seq}(v_1)\phi\texttt{seq}(v_2)\phi \dots \phi\texttt{seq}(v_{|\mathcal{E}_G|})\phi$ that we call *linearization* of the splicing graph. See Figure 3.1 for an example.

It is immediate to see that, given a vertex v of $\mathcal{S}_G$, the label $\texttt{seq}(v)$ is a particular substring of the linearization Z. However, it is not so immediate to understand how to retrieve its sequence or how to extract its starting and ending position on Z. Indeed, to this aim, we pair the string Z with a bit vector of the same length, denoted as B_Z, such that $B_Z[i] = 1$ if and only if $Z[i] = \phi$, otherwise $B_Z[i] = 0$. Thanks to B_Z and the select operation, given an index i, we can compute the starting position s of the i-th label of the linearization, *i.e.* $s = \texttt{select}_{B_Z}(i) + 1$, its ending position $e = \texttt{select}_{B_Z}(i+1) - 1$, and its sequence, *i.e.* $Z[s, e]$.

By employing the algorithm by Ohlebusch et al. [117], all the MEMs longer than a constant L between R and Z, thus between R and $\mathcal{S}_G$, can be computed in linear time with respect to the length of the reads and the number of MEMs. See Chapter 2 for more details. Thanks to the special character ϕ which occurs in Z and not in R, each MEM occurs inside a single vertex label and cannot span two different labels. Indeed, we recall that a MEM is a common substring between the two input strings.

In the following, given a read R and the linearization Z of $\mathcal{S}_G$, we say that the MEM $m = (i_Z, i_R, \ell)$ *belongs to* the i-th vertex of the linearization with $i = \texttt{rank}_{B_Z}(i_Z)$, *i.e.* m represents a substring of the label of the i-th vertex. We say that a MEM $m = (i_Z, i_R, \ell)$ precedes another MEM $m' = (i'_Z, i'_R, \ell')$ in R if $i_R < i'_R$ and $i_R + \ell < i'_R + \ell'$, and we denote this by $m \prec_R m'$. Similarly, we can define an analogous property with respect to Z. If m and m' belong to the same

vertex, *i.e.* $\texttt{rank}_{B_Z}(i_Z) = \texttt{rank}_{B_Z}(i'_Z)$, then m precedes m' in Z if $i_Z < i'_Z$ and $i_Z + \ell < i'_Z + \ell'$. Otherwise, if the two MEMs belong to two different vertices v and v', we say that m precedes m' if and only if the exon associated to vertex v precedes the exon associated to vertex v'. We denote this relation of precedence on Z by $m \prec_Z m'$.

When m precedes m' in R, we say that $lgap_R = i'_R - (i_R + \ell)$ is the length of the gap between the two MEMs on R. Similarly, when $m \prec_Z m'$ and the two MEMs are on the same vertex, we say that $lgap_Z = i'_Z - (i_Z + \ell)$ is the length of the gap between the two MEMs on Z. If $lgap_R$ or $lgap_Z$ (or both) are positive, we refer to the gap strings as $sgap_R$ and $sgap_Z$, whereas when they are negative, we say that m and m' *overlap* either in R or Z (or both).

Finally, given MEM $m = (i_Z, i_R, \ell)$ belonging to the i-th vertex v of the linearization (we recall that $i = \texttt{rank}_{B_Z}(i_Z)$), we denote as $\texttt{PREF}_Z(m)$ and $\texttt{SUFF}_Z(m)$ the prefix and the suffix of $\texttt{seq}(v)$ upstream and downstream from the start and the end of m, respectively. More formally, $\texttt{PREF}_Z(m) = Z[\texttt{select}_{B_Z}(i)+1, i_Z-1]$ and $\texttt{SUFF}_Z(m) = Z[i_Z+\ell, \texttt{select}_{B_Z}(i+1)-1]$. Similarly, we denote as $\texttt{PREF}_R(m)$ and $\texttt{SUFF}_R(m)$ the prefix and the suffix of R upstream and downstream from the start and the end of m, *i.e.* $\texttt{PREF}_R(m) = R[0, i_R - 1]$ and $\texttt{PREF}_R(m) = R[i_R + \ell, |R| - 1]$.

Figure 3.2 summarizes the definitions of precedence between MEMs, gap, overlap, PREF, and SUFF.

Spliced graph-alignment

We are now able to define the central concepts that will be used in our method. In particular, we first define a general notion of gap graph-alignment and then we introduce specific constraints on the use of gaps to formalize a *spliced graph-alignment* that is fundamental for the detection of alternative splicing events. The key observation is that an RNA-Seq read may align to the sequence that derives from the concatenation of the exon sequences along a path of $\mathcal{S}^\star_G$ and this type of alignment is able to give an evidence of the (possibly unannotated) splice sites supported by the read with respect to the original splicing graph S_G.

A *gap graph-alignment* of R to graph $\mathcal{S}_G$ is a pair (A, π) where $\pi = \langle v_1, \ldots, v_k \rangle$

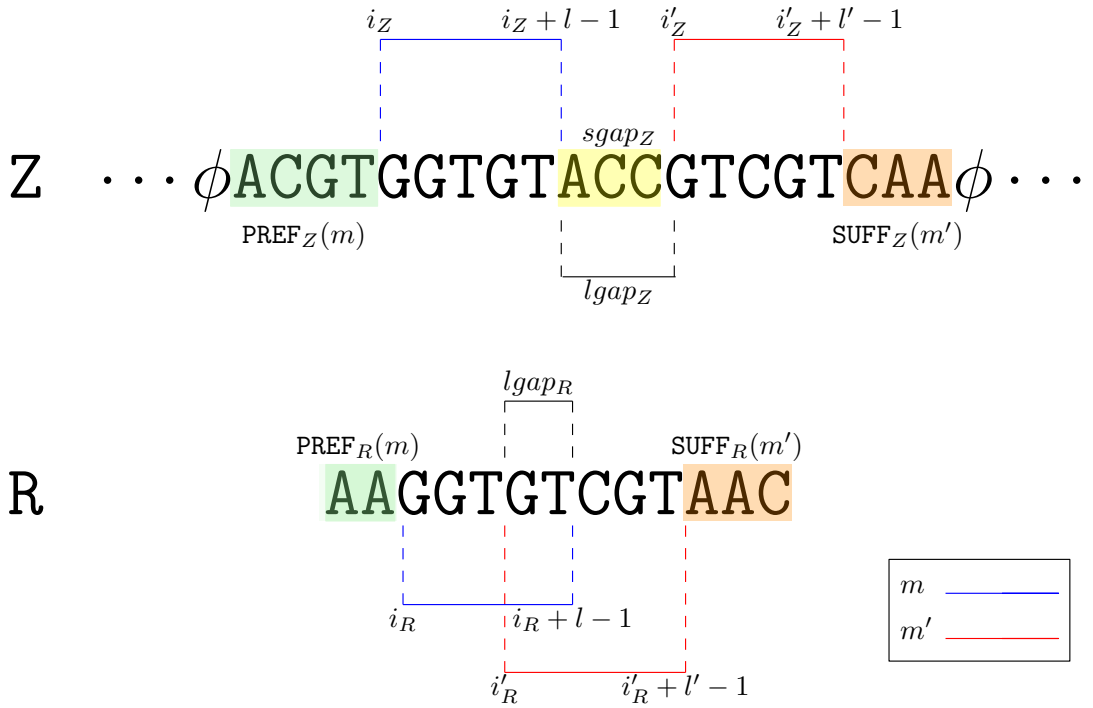


Figure 3.2: Precedence relation between MEMs. Two MEMs, $m = (i_Z, i_R, \ell)$ and $m' = (i'_Z, i'_R, \ell')$, are shown in the figure. For ease of presentation we represent in blue the former and in red the latter. Since $i_Z < i'_Z$ and $i_Z + \ell < i'_Z + \ell'$, m precedes m' on Z; analogously m precedes m' on R since $i_R < i'_R$ and $i_R + \ell < i'_R + \ell'$. The length $lgap_Z$ of the gap between the two MEMs on Z is positive and we refer to the string between $i_Z + \ell$ and $i'_Z - 1$ as $sgap_Z$ (highlighted in yellow on Z). Conversely, the length of the gap between the two MEMs on R is negative and we say that they overlap on R. Finally, we refer to the string between the start of the vertex label and $i_Z - 1$ as $\texttt{PREF}_Z(m)$ (highlighted in light green on Z), and to the string between $i'_Z + \ell$ and the end of the vertex label as $\texttt{SUFF}_\texttt{Z}(\texttt{m}')$ (highlighted in orange on Z). Moreover, we refer to the string between the start of the read and $i_R - 1$ as $\texttt{PREF}_R(m)$ (highlighted in light green on R), and to the string between $i'_Z + \ell$ and the end of the read as $\texttt{SUFF}_\texttt{R}(\texttt{m}')$ (highlighted in orange on R). For ease of presentation, we did not report $\texttt{SUFF}_Z(m)$, $\texttt{PREF}_Z(m')$, $\texttt{SUFF}_R(m)$, and $\texttt{PREF}_R(m')$.

is a path of the graph $\mathcal{S}^{\star}_{G}$ and

$$A = \langle (p_1, r_1), (p'_1, r'_1), \ldots, (p'_{n-1}, r'_{n-1}), (p_n, r_n) \rangle$$

is a sequence of pairs of strings, with $n \geq k$, such that $\texttt{seq}(v_1) = x \cdot p_1$ and $\texttt{seq}(v_k) = p_n \cdot y$, for x, y possibly empty strings and $P = p_1 \cdot p'_1 \cdot p_2 \cdot p'_2 \cdot p_3 \cdot \ldots \cdot p'_{n-1} \cdot p_n$ is the string labeling the path π and $R = r_1 \cdot r'_1 \cdot r_2 \cdot \ldots \cdot r'_{n-1} \cdot r_n$. We recall that $\cdot$ operator is the string concatenation operator.

The pair (p_i, r_i), called a *factor* of the alignment, consists of a non-empty substring r_i of R and a non-empty substring p_i of the label of a vertex in π. On the other hand, the pair (p'_i, r'_i) is called a *gap-factor* of the alignment if at least one of p'_i and r'_i is an empty substring ϵ. Moreover, either p'_i is empty or $|p'_i| > \alpha$, and either r'_i is empty or $|r'_i| > \alpha$, for a fixed value α which represents the maximum *alignment indel* size allowed. When an insertion (or a deletion) is smaller than α, we consider it an alignment indel and we incorporate it into a factor; otherwise, we consider it as a clue of the possible presence of an alternative splicing event and we represent it as a gap-factor. Intuitively, in a gap graph-alignment, factors correspond to portions of exons covered (possibly with errors) by portions of the read, whereas gap-factors correspond to introns, which can be already annotated or novel, and which can be used to infer the possible presence of alternative splicing events.

We associate to each factor (p_i, r_i) the cost $\delta(p_i, r_i)$, and to each gap-factor (p'_i, r'_i) the cost $\delta(p'_i, r'_i)$, by using a function $\delta(\cdot, \cdot)$ with positive values. The *cost* of the alignment (A, π) is computed as:

$$\texttt{cost}(A, \pi) = \sum_{i=1}^{n} \delta(p_i, r_i) + \sum_{i=1}^{n-1} \delta(p'_i, r'_i).$$

Moreover, we define the *error* of a gap graph-alignment as the sum of the edit distance of each factor but not of gap-factors. Indeed, factors are true portions of the alignment and must be evaluated when computing the overall error. Gap-factors instead represent gaps in the alignment and are used to infer alternative splicing events. Therefore, they should not be considered in the computation of the alignment error. Formally, the error of the alignment (A, π) is:

$$\texttt{Err}(A, \pi) = \sum_{i=1}^{n} d_E(p_i, r_i),$$

where $d_E(\cdot,\cdot)$ denotes the edit distance between two strings.

To define a splice-aware alignment, that we call *spliced graph-alignment*, we need to classify each gap-factor and to assign it a cost. Our primary goal is to compute a gap graph-alignment of the read to the splicing graph that possibly reconciles to the gene annotation; if this is not possible, then we want to minimize the number of novel alternative splicing events, *i.e.* the number of gap-factors. For this reason we distinguish three types of gap-factors: *annotated*, *novel*, and *uninformative*. Intuitively, an annotated gap-factor models an annotated intron, a novel gap-factor represents a novel intron, whereas an uninformative gap-factor does not represent any intron.

Formally, we classify a gap-factor (p'_i, r'_i) as annotated if and only if $p'_i = r'_i = \epsilon$ and the two strings p_i, p_{i+1} are on two different vertices that are linked by an edge in $\mathcal{S}_G$. We classify a gap-factor (p'_i, r'_i) as novel in the following cases:

1. $r'_i = \epsilon$ and $p'_i = \epsilon$ occurs between the strings p_i and p_{i+1} which belong to two distinct vertices linked by an edge in $\mathcal{S}^\star_G$ and not in $\mathcal{S}_G$. This gap-factor represents an exon skipping event (see Figure 3.3, case *a*).

2. $r'_i = \epsilon$ and $p'_i \neq \epsilon$ occurs between the strings p_i and p_{i+1} which belong to the same vertex of $\mathcal{S}^\star_G$. This gap-factor represents an intron retention event (see Figure 3.3, case *b*).

3. $r'_i = \epsilon$ and $p'_i \neq \epsilon$ occurs between the strings p_i and p_{i+1} which belong to two distinct vertices linked by an edge in $\mathcal{S}^\star_G$. This gap-factor represents an alternative splice site event shortening an exon (see Figure 3.3, case *c*).

4. $r'_i \neq \epsilon$ and $p'_i = \epsilon$ occurs between the strings p_i and p_{i+1} which belong to two distinct vertices linked by an edge in $\mathcal{S}^\star_G$. This gap-factor represents an alternative splice site extending an exon or a new exon event (see Figure 3.3, case *d*-*e*).

Note that case 1 allows to detect a novel intron whose splice sites are both annotated (see Figure 3.3, case *a*). Case 2 supports the presence of an intron retention (see Figure 3.3, case *b*) and our approach is able to locate the two novel splice sites inside the annotated exon. Case 3 gives an evidence of a novel

alternative splice event shortening an annotated exon (see Figure 3.3, case c) and our approach finds the novel splice site supported by this case. Finally, in case 4, we are able to detect a novel alternative splice site (extending an annotated exon) or a novel exon (see Figure 3.3, case d), but only in the first case (alternative splice site) our approach is able to find the novel splice site induced by the gap-factor.

For ease of presentation, Figure 3.3 shows only "classic" alternative splicing event and not their combination as those modeled with the notion of Local Splicing Variations [162]. We note here that our formalization takes into account the combination of alternative splicing events as those given by an exon skipping combined with an alternative splice site (see definition of gap-factor in cases 3 and 4). However, the actual version of the tool is designed only to detect the alternative splicing events shown in Figure 3.3.

Finally, we classify a gap-factor (p'_i, r'_i) as uninformative in the two remaining cases, which are (i) $r'_i = \epsilon$ and $p'_i = \epsilon$ occurs between strings p_i and p_{i+1} which belong to the same vertex, and (ii) $r'_i \neq \epsilon$ and $p'_i = \epsilon$ occurs between strings p_i and p_{i+1} which belong to the same vertex. We note that both these cases can never occur. Indeed, in the former case, factors (p_i, r_i) and (p_{i+1}, r_{i+1}) are joined into a unique factor. In the latter, the gap-factor represents an unrealistic situation that does not represent any alternative splicing event: there is a gap in the read and not in the exon.

Let $\mathcal{G}_{\mathcal{F}}$ be the set of novel gap-factors of a gap graph-alignment A. Then a *spliced graph-alignment* (A, π) of R to $\mathcal{S}_G$ is a gap graph-alignment in which uninformative gap-factors are not allowed, whose cost is defined as the number of novel gap-factors (therefore δ function assigns a cost 1 to each novel gap-factor and a cost 0 to all other factors and annotated gap-factors), and whose error is at most β, for a given constant β. Thus, in a spliced graph-alignment, $\texttt{cost}(A, \pi) = |\mathcal{G}_{\mathcal{F}}|$ and $\texttt{Err}(A, \pi) \leq \beta$. We focus on a bi-criteria version of the computational problem of computing the optimal spliced graph-alignment (A, π) of R to a graph $\mathcal{S}_G$, where first we minimize the cost, then we minimize the error. The intuition is that we want a spliced graph-alignment of a read that introduces the fewest novel alternative splicing events. Moreover, among all such alignments we look for the alignment that introduces the lowest number

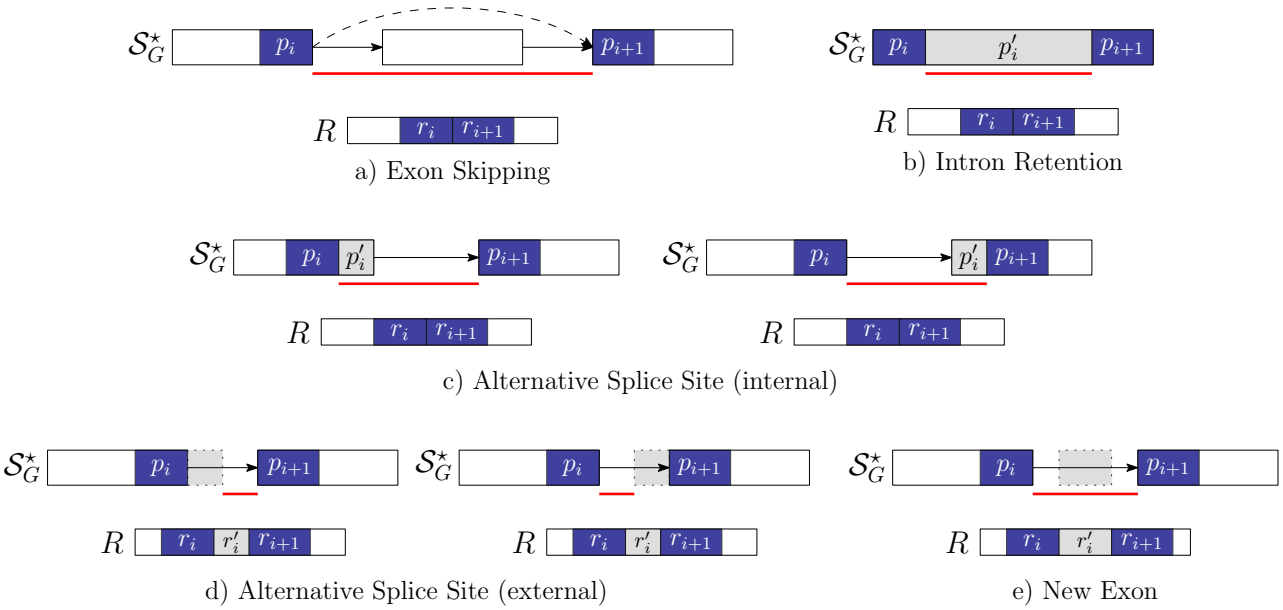


Figure 3.3: The relationship among novel gap-factors, introns, and alternative splicing events is shown. Each subfigure depicts an example of novel gap-factor (p'_i, r'_i) (gray boxes) in relation to a simple graph $\mathcal{S}^\star_G$, where dashed arrows represent novel edges (*i.e.* edges not present in the splicing graph $\mathcal{S}_G$) and a read R. The two consecutive factors (p_i, r_i) and (p_{i+1}, r_{i+1}) of a spliced graph-alignment are represented by blue boxes, and the red lines represent the novel introns supported by the gap-factors. In terms of novel alternative splicing events, gap-factor (ϵ, ϵ) supports an exon skipping event (case (a)), gap-factor (p'_i, ϵ) supports an intron retention event (case (b)) or an alternative splice site event shortening an exon (case (c)). Finally, gap-factor (ϵ, r'_i) supports an alternative splice site event extending an exon (case (d)) or a new exon (case (e)).

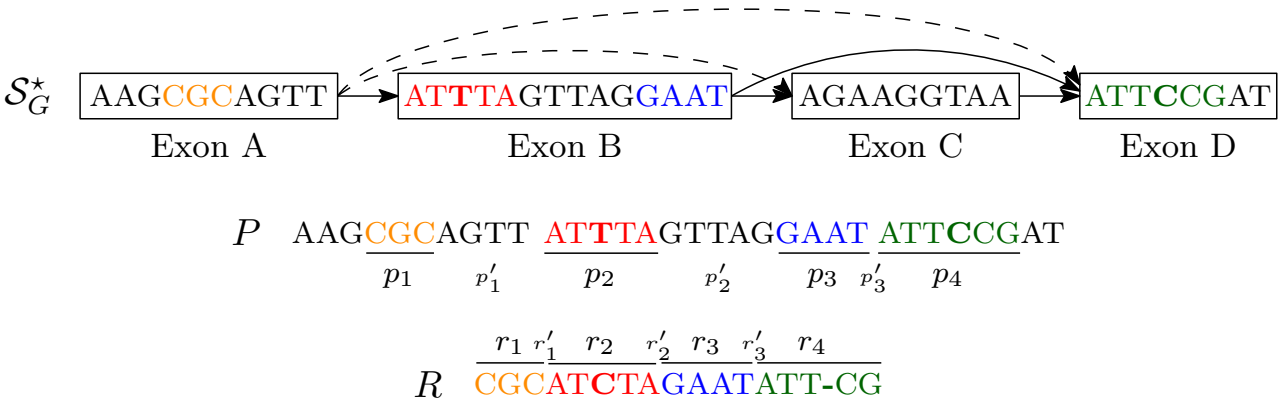


Figure 3.4: Example of a spliced graph-alignment of a read R to a splicing graph $\mathcal{S}^\star_G$ (the dashed arrows represent novel edges). The read R is factorized in four strings r_1, r_2, r_3, and r_4 matching to strings p_1, p_2, p_3, and p_4 of P, which is the concatenation of exon labels of path $\pi = \langle A, B, D\rangle$. This yields to the spliced graph-alignment $(\langle(p_1, r_1), (p'_1, r'_1), (p_2, r_2), (p'_2, r'_2), (p_3, r_3), (p'_3, r'_3), (p_4, r_4)\rangle, \pi)$. We observe that p'_3, r'_1, r'_2, and r'_3 are equal to ϵ. Moreover we note that (p'_1, r'_1), (p'_2, r'_2) are two novel gap-factors, r_2 matches p_2 with an error of substitution, and r_4 matches p_4 with an error of insertion: both the error and the cost of this spliced-graph alignment are equal to 2. The alignment of R to the splicing graph of G supports the evidence of two *novel* alternative splicing events: an alternative donor site of exon A and an intron retention on exon B.

of errors, *i.e.* the alignment with the smallest edit distance (which is likely due to sequencing errors and polymorphisms) in the non-empty regions that are aligned (*i.e.* the factors). Figure 3.4 shows an example of spliced graph-alignment of error value 2, and cost 2.

In this chapter we propose an algorithm that, given a read R, a splicing graph $\mathcal{S}_G$, and three constants, which are L (the minimum length of a `MEM`), α (the maximum alignment indel size), and β (the maximum number of allowed errors), computes an optimal spliced graph-alignment – that is, among all spliced graph-alignments with minimum cost, the alignment with minimum error. The next section details the approach we propose to detect alternative splicing events by computing the optimal spliced graph-alignments of a RNA-Seq sample.

3.3 Method

In this section we describe ASGAL (Alternative Splicing Graph ALigner), that is the approach we propose to identify the alternative splicing events supported by an RNA-Seq sample by computing the optimal spliced graph-alignments of the reads.

More precisely, ASGAL takes as input the annotation of a gene together with the related reference sequence, and an RNA-Seq sample. Then, by exploiting the notion of spliced graph-alignments, it computes and outputs the spliced alignments of each input read and the alternative splicing events supported by the sample which are novel with respect to the input annotation. We point out that ASGAL uses the input reference sequence only for building the splicing graph as well as for refining the alignments computed against it, with the specific goal of improving the accuracy of the alternative splicing event detection. Each identified event is described by its type (*i.e.* exon skipping, intron retention, alternative acceptor splice site, and alternative donor splice site), its genomic location, and a measure of its quantification, *i.e.* the number of alignments that support the identified event.

For the sake of clarity, we will describe our method considering as input the splicing graph of a single gene: it can be easily generalized to manage more than a gene at a time.

ASGAL consists of the following steps: (1) construction of the splicing graph of the gene, (2) computation of the spliced graph-alignments of the RNA-Seq reads, (3) remapping of the alignments from the splicing graph to the genome, and (4) detection of the novel alternative splicing events. Figure 3.5 depicts the ASGAL pipeline.

Splicing graph construction In the first step, ASGAL builds the splicing graph $\mathcal{S}_G$ of the input gene using the reference genome and the gene annotation, and adds the novel edges to obtain the graph $\mathcal{S}_G^\star$. To do so, we iterate over the gene annotation and we create a vertex for each distinct exon, we associate to each vertex its genomic sequence, extracted from the input reference genome, and we link with an edge each pair of exons that are consecutive in at least one

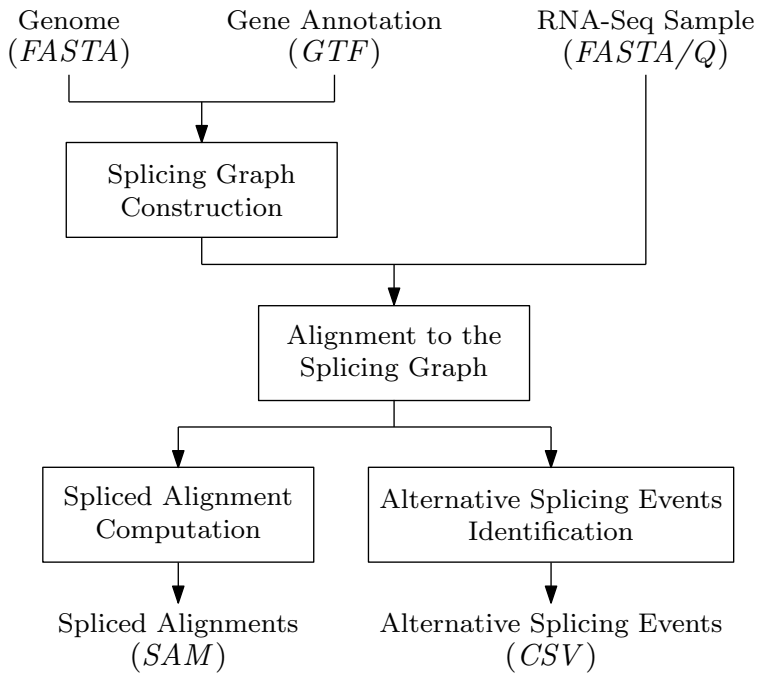


Figure 3.5: `ASGAL` pipeline. The steps of the pipeline implemented by `ASGAL` are shown together with their input and output: the splicing graph is built from the reference genome (FASTA file) and the gene annotation (GTF file), the RNA-Seq sample (FASTA or FASTQ file) is aligned to the splicing graph, and finally the alignments to the splicing graph are used to compute the spliced alignments to the reference genome (SAM file) and to detect the alternative splicing events supported by the sample (CSV file).

transcript. Then, we add an edge between each pair of exons (vertices) (e_i, e_j) such that e_i precedes e_j.

Alignment to the splicing graph The second step of `ASGAL` computes the spliced graph-alignments of each read R in the input RNA-Seq sample by combining `MEMs` into factors and gap-factors. We build the FM-Index of the string Z, the linearization of the splicing graph $\mathcal{S}_G$, and we use the approach proposed by Ohlebusch et al. [117] to compute, for each input read R, the set of `MEMs` between Z and R with minimum length L, a user-defined parameter. We recall that the string Z is obtained by concatenating the labels of the vertices of the splicing graph and interposing between them the ϕ character. This separator prevents `MEMs` from covering multiple exons by spanning the junction between two vertex labels. We point out that the splicing graph linearization is performed only once before aligning the input reads to the splicing graph.

Once the set M of `MEMs` between R and Z is computed, we build a graph $G_M = (M, E_M)$ with weighted edges by employing the two precedence relations between `MEMs`, $\prec_R$ and $\prec_Z$, respectively, defined in Section 3.2. Then we use such graph to extract the spliced graph-alignment.

Intuitively, each vertex of this graph represents a perfect match between a portion of the input read and a portion of an annotated exon whereas each edge models an alignment error, a gap-factor of the spliced graph-alignment, or both. More precisely, there exists an edge from m to m', with $m, m' \in M$, if and only if $m \prec_R m'$ (m precedes m' on the read) and one of the following six conditions, also depicted in Figure 3.6), holds:

1. m and m' belong to the same vertex, $m \prec_Z m'$, and either (i) $lgap_R > 0$ and $lgap_Z > 0$, or (ii) $lgap_R = 0$ and $0 < lgap_Z \leq \alpha$. The weight of the edge (m, m') is set to the edit distance between $sgap_R$ and $sgap_Z$.

2. m and m' belong to the same vertex, $m \prec_Z m'$, $lgap_R \leq 0$, and $lgap_Z \leq 0$. The weight of the edge (m, m') is set to $|lgap_R - lgap_Z|$.

3. m and m' belong to the same vertex, $m \prec_Z m'$, $lgap_R \leq 0$ and $lgap_Z > \alpha$. The weight of the edge (m, m') is set to 0.

4. m and m' belong to two different vertices v_1 and v_2 with $v_1 \prec v_2$, and $lgap_R \leq 0$. The weight of the edge (m, m') is set to 0.

5. m and m' belong to two different vertices v_1 and v_2 with $v_1 \prec v_2$, $lgap_R > 0$, and $\texttt{SUFF}_Z(m) = \texttt{PREF}_Z(m') = \epsilon$. The weight of the edge (m, m') is set to 0 if $lgap_R > \alpha$, and to $lgap_R$ otherwise.

6. m and m' belong to two different vertices v_1 and v_2 with $v_1 \prec v_2$, $lgap_R > 0$, and at least one between $\texttt{SUFF}_Z(m)$ and $\texttt{PREF}_Z(m')$ is not ϵ. The weight of the edge (m, m') is set to the edit distance between $sgap_R$ and the concatenation of $\texttt{SUFF}_Z(m)$ and $\texttt{PREF}_Z(m')$.

Note that the aforementioned conditions do not cover all of the possible situations that can occur between two `MEMs`, but they represent those that are relevant for computing the spliced graph-alignments of the considered read.

Intuitively, m and m' contribute to the same factor (p_i, r_i) in cases 1 and 2 and the non-zero weight of the edge (m, m') concurs to the spliced graph-alignment error. In cases 3-6, the edge (m, m') models the presence of a novel gap-factor. More precisely, m contributes to the end of a factor (p_i, r_i) and m' contributes to the start of the consecutive factor (p_{i+1}, r_{i+1}). The novel gap-factor in between these two factors models an intron retention on an annotated exon (case 3), an alternative splice site shortening an annotated exon or, if the edge linking the two vertices is a novel edge, an exon skipping event (case 4), and an alternative splice site extending an annotated exon or a novel exon (case 5). Finally in case 6, the gap-factor can identify either a novel exon skipping event or an already annotated intron, depending on the type of the edge linking the two vertices. In both these cases, the non-zero weight of the edge contributes to the spliced graph-alignment error.

Once we have processed all the `MEMs` and we have built the graph G_M, we add to the graph a global source vertex v_b and a global sink vertex v_e and we link these vertices to the rest of the graph as follows. For each source `MEM` $m = (i_Z, i_R, \ell)$ of the graph, we add an edge (v_b, m) and we weight it as $d_E(\texttt{PREF}_R(m), \texttt{PREF}_Z[i_Z - 1 - |\texttt{PREF}_R(m)|, i_Z - 1]))$, *i.e.* the edit distance between the read prefix and the exon prefix upstream `MEM` m. If the prefix of the read is longer than the prefix of the vertex label to which m belongs, we just iterate over all the parents of

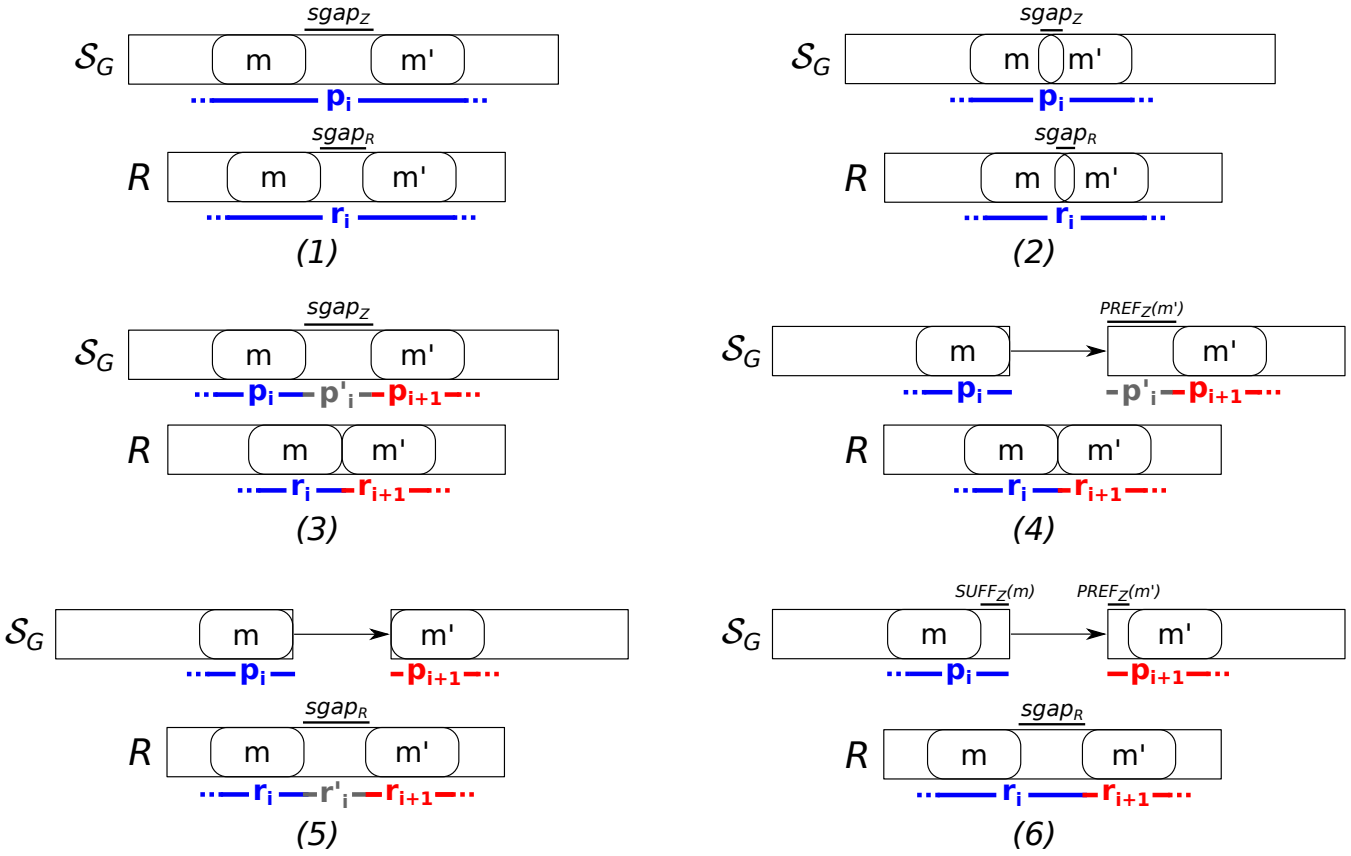


Figure 3.6: The six conditions used to connect two different `MEMs` and to build the factors and gap-factors of a spliced graph-alignment are shown. In all the conditions, the first `MEM` must precede the second one on the read. In condition (1) and (2), the two `MEMs` belong to the same vertex label and leave a gap (condition (1)) or overlap (condition (2)) on the read or on the vertex label. In these conditions, the two `MEMs` are joined in the same factor of the alignment. In condition (3), the two `MEMs` belong to the same vertex label but they leave a long gap only on the vertex label and not on the read. In this case the two `MEMs` are part of two different factors linked by a gap-factor. In the other conditions, the two `MEMs` belong to two different vertices of the splicing graph, linked by a (possible novel) edge. For this reason, in any of these cases, the two `MEMs` are part of two different factors of the alignment. In condition (4), the two `MEMs` leave a gap only on the path, in condition (5) they leave a gap only on the read, and in condition (6), they leave a gap on both the path and the read.

this vertex and we select the parent whose suffix better align to the prefix of the read, *i.e.* we compute the edit distance between the suffix of the parent and the prefix of the read and we increment the weight of the edge. Similarly, we link each sink `MEM` of the graph to the global sink vertex, possibly considering all the sons of the vertex the `MEM` belongs to. In this way, by using a local alignment step, we are able to align also the prefix or the suffix of a read that may have been left unmapped due to sequencing errors near its ends.

Finally the spliced graph-alignment of read R is computed by a visit of the graph G_M. Indeed, each path π_M of this graph starting from the global source vertex and ending at the global sink vertex represents a spliced graph-alignment and the weight of the path is the number of differences between the read R and a path of the splicing graph covered by π_M. For this reason, we use Dijkstra's algorithm to select the lightest path in G_M, with weight less than β (the given error threshold) which also contains the minimum number of novel gap-factors, *i.e.* we select an optimal spliced graph-alignment.

Spliced alignment computation The third step of `ASGAL` computes the spliced alignments of each input read with respect to the reference genome starting from the spliced graph-alignments computed in the previous step. Exploiting the gene annotation, we convert the coordinates of factors and gap-factors in the spliced graph-alignment to positions on the reference genome. Indeed, observe that factors map to coding regions of the genome whereas gap-factors identify the skipped regions of the reference, *i.e.* the introns induced by the alignment, modeling the possible presence of alternative splicing events (see Figure 3.3 for details).

As showed previously, a factor can be informally described as a set of `MEMs` that are consecutive both on the read and on one vertex of the splicing graph Most commonly, a factor is simply a `MEM`. To compute the starting and ending positions of a factor on the reference genome, we have to map its first and its last `MEM` from the linearization Z to the reference genome.

Let $m = (i_Z, i_R, \ell)$ be a `MEM`. The starting position of m on the linearization is i_Z. To map this position to the reference genome, we identify the vertex label to which m belongs, we retrieve from the input annotation the starting position on

the reference of the exon associated to such vertex, and we add to this position the length of the prefix of the vertex label upstream m, *i.e.* $\texttt{PREF}_Z(m)$. Similarly, we can map the ending position of m. In this way we are able to map a factor from the linearization to the reference genome.

However this holds in any possible situation except when factors p_i and p_{i+1} are on two different vertices and only p'_i is ϵ (case *d-e* of Figure 3.3). In this case the portion r'_i must be aligned to the intron between the two exons whose labels contains p_i and p_{i+1} as a suffix and prefix, respectively. If r'_i aligns to a prefix or a suffix of this intron, then the left or right coordinate of the examined intron is modified according to the length of r'_i (Figure 3.3(d)). In the other case (Figure 3.3(e)), the portion r'_i is not aligned to the intron and it is represented as an insertion in the alignment. Such an insertion is a hint of the possible presence of a novel exon.

However, the third step of our approach is not limited to a simply conversion of coordinates from the linearization to the reference genome but it also performs a further refinement of the computed splice sites. Indeed, whenever it identifies a novel splice site, it searches for the splice sites (in a maximum range of 3 bases with respect to the detected ones) determining the best intron pattern (firstly *GT-AG*, secondly *GC-AG* if *GT-AG* has not been found).

Alternative splicing events identification In the fourth step, `ASGAL` uses the set $\mathcal{I}$ of introns supported by the spliced alignments computed in the previous step, *i.e.* the set of introns associated to each gap-factor, to detect the novel alternative splicing events supported by the input RNA-Seq sample with respect to the input annotation.

Let $\mathcal{I}_n \subseteq \mathcal{I}$ be the set of introns which are not present in the annotation, that is the set of *novel* introns. For each novel intron $[p_s, p_e] \in \mathcal{I}_n$ which is supported by at least ω alignments with ω a user-defined parameter representing a confidence threshold, `ASGAL` identifies one of the following events:

- *Exon skipping*, if there exists an annotated transcript containing two non-consecutive exons $[a_i, b_i]$ and $[a_j, b_j]$, such that $b_i = p_s - 1$ and $a_j = p_e + 1$.
- *Intron retention*, if there exists an annotated transcript containing an exon

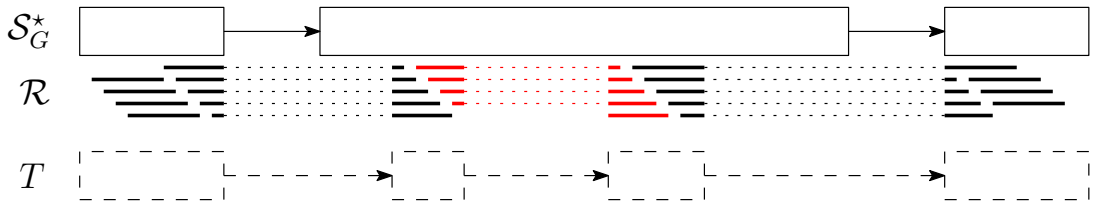


Figure 3.7: Example of false intron retention. The figure depicts a splicing graph $\mathcal{S}^{\star}_G$, a transcript T, and the alignments of a sample of reads. In this case the transcript T supported by the alignments shows a complex alternative splicing event consisting of two new exons. `ASGAL` detects the new intron supported by the red alignments, but the analysis of the neighboring introns shows that no simple alternative splicing event can explain the alignments: this situation is recognized by `ASGAL` that refuses to make any prediction of a novel (surely incorrect) intron retention event.

$[a_i, b_i]$ such that (i) $a_i < p_s < p_e < b_i$, (ii) there exists an intron in $\mathcal{I}$ ending at $a_i - 1$ or a_i is the start of the transcript and (iii) there exists another intron in $\mathcal{I}$ starting at $b_i + 1$ or b_i is the end of the transcript.

– *Alternative acceptor site*, if there exists an annotated transcript containing two consecutive exons $[a_i, p_s - 1]$ and $[a_j, b_j]$ such that $p_e < b_j$, and there exists an intron in $\mathcal{I}$ starting at $b_j + 1$ or b_j is the end of the transcript.

– *Alternative donor site*, if there exists an annotated transcript containing two consecutive exons $[a_i, b_i]$ and $[p_e + 1, b_j]$ such that $p_s > a_i$, and there exists an intron in $\mathcal{I}$ ending at $a_i - 1$ or a_i is the start of the transcript.

These definitions are accurately designed to minimize the chances of mistaking a complex alternative splicing event as those modeled with the notion of Local Splicing Variations for an alternative splicing event. For example, if we remove conditions (ii) and (iii) from the definition of intron retention, *i.e.* we analyze each intron locally without considering the neighboring introns, we could confuse the situation shown in Figure 3.7 with an intron retention event.

Genome-wide analysis

ASGAL is specifically designed to perform the prediction of alternative splicing events based on a splice-aware alignment of a RNA-Seq sample against the splicing graph of a specific gene. The current version of tool is time efficient when a limited set of genes are analyzed. For genome-wide analysis, *i.e.* when all the known genes of an organism are of interest, it is unrealistic to align the input sample against the splicing graph of each gene. For this reason, we have implemented a pre-processing step that aims to speed up the process of filtering reads that map to the genes under investigation. Given a set of genes and an RNA-Seq sample, this filtering procedure consists of three steps: (i) the quasi-mapping algorithm of Salmon [122] is first used to quantify the transcripts of the genes and to quickly assign each read to the transcripts, (ii) a set of smaller RNA-Seq samples, one for each gene, is then produced by analyzing the output of Salmon, and finally (iii) if the input sample is a paired-end sample, then the mate of the mapped reads that were not mapped by Salmon are added to these smaller samples. Once we split the input RNA-Seq experiment in smaller samples, it is possible to use ASGAL on the different genes without having to align the entire sample of reads against each of them. We note that we decided to use Salmon as pre-processing step since it is very fast and it allows to split the input sample faster than any other spliced aligner. Anyway, in aligning reads to a reference transcriptome, some reads which cover unannotated exons are not aligned, excluding them from any downstream analysis: for this reason, future works will focus on improving this step.

3.4 Results

We implemented ASGAL in C++ and Python and we performed an experimental analysis on simulated and real data to assess ASGAL's ability to align reads to a splicing graph and to detect alternative splicing events. ASGAL is freely available at https://github.com/AlgoLab/galig. Information and instruction on how to replicate the performed experiments are available at https://asgal.algolab.eu.

We ran ASGAL using its default parameters in all the experiments. The default values, chosen after a preliminary analysis performed on an RNA-Seq sample simulated from three human chromosomes, allows to achieve a good trade-off between accuracy and efficiency. More precisely, the minimum length of the MEMs (L) is 15, α and β are 3% of the length of the input reads, and the minimum support for reporting an alternative splicing events (ω) is 3 (alignments). The analyses were performed on a 64 bit Linux (Kernel 4.4.0) system equipped with Four 8-core Intel ® Xeon 2.30GHz processors and 256GB of RAM.

3.4.1 Implementation details

The FM-Index, the bit vector and the rank/select data structures were implemented using the sdsl-lite library [48]. To compute MEMs, we reimplemented the approach proposed by Ohlebusch et al. [117] available at https://www.uni-ulm.de/in/theo/research/seqana/ porting it to sdsl-lite (v2.0). The MEMs graph built and visited to compute the spliced graph-alignments was implemented using the lemon library [37].

3.4.2 Experimental analysis on simulated data

In the first phase of our experimental analysis, we evaluated our approach using simulated data. The goal of this analysis was twofold: (i) to assess the accuracy and the efficiency of our method in aligning an RNA-Seq sample against a splicing graph, and (ii) to assess how well the method detects the alternative splicing events supported by a sample with respect to a given annotation.

To avoid any bias in the experiments, we decided to reuse the same reference genome, gene annotations, and RNA-Seq samples simulated with Flux Simulator [53] used in [69][1]. We considered two different RNA-Seq datasets of this corpus. More precisely, the sample composed of 5 million reads and the one composed of 10 million reads. From now on, we will refer to these datasets as 5M and 10M, respectively. Each dataset covers 1000 randomly selected genes of the human GENCODE annotation (v19) [57]. We used AStalavista (version 4.0) to extract the alternative splicing events included in the annotation of each

[1]Data available at http://public.bmi.inf.ethz.ch/projects/2015/spladder/

gene, then we selected the genes whose annotation includes at least one alternative splicing event. After these filtering steps, the set of genes under analysis included 656 elements. Finally, we divided each read sample into 24 samples (by using the information included in the header of each entry of the simulated sample), one for each chromosome, and we used `cutadapt` [99] (version 1.14) to remove poly-A tails.

Validation of the alignment step

In the first part of our experimental analysis, we compared the alignment step of `ASGAL` with `STAR` (version 2.5.4b), one of the best-known spliced aligner. Let us recall that `ASGAL` performs a splice-aware alignment and its current implementation is specifically designed to confirm or detect novel splice sites by aligning reads to a splicing graph. Since our tool works at the gene level, that is, it considers the splicing graph of each gene independently, we ran `ASGAL` on each gene independently whereas we ran `STAR` in two-pass mode on each chromosome, providing the annotation of the considered genes. We then selected all primary alignments reported by the two tools and we compared them using different metrics, as in [41].

Table 3.1 reports the total number of mapped reads (91% for `ASGAL`, and 97% for `STAR`), as well as the number of alignments per read reported by the two tools. As expected, since we considered only primary alignments and since `STAR` aligns the input sample to the input reference, `STAR` yields a single alignment per read. Conversely, `ASGAL` considers each gene independently, thus it might align the same read to different genes and report multiple primary alignments for it (at most one for each gene). However, this behaviour is extremely rare. Indeed, less than 0.2% of the considered reads are aligned to multiple genes.

We also assessed the basewise accuracy of `ASGAL` and `STAR`. As shown in Table 3.2, $\sim$ 98% of the primary alignments produced by both the tools place the read to the correct location, *i.e.* all the bases of the read are placed in the position from where they were extracted. `ASGAL` produces fewer "Partially Mapped" alignments, *i.e.* the alignments which place some but not all the read bases in the correct positions, but more "Differently Mapped" alignments, *i.e.* the alignments which place all the read bases in positions different from those

Sample	Tool	Total reads	Unmapped	Alignments per read					Mapped reads
				1	2	3	4	5	
5M	ASGAL	3,226,895	281,767	2,938,383	6,734	2	9	0	2,945,128
	STAR		75,947	3,150,948	0	0	0	0	3,150,948
10M	ASGAL	6,522,455	571,202	5,942,220	9,009	13	10	1	5,951,253
	STAR		166,593	6,355,862	0	0	0	0	6,355,862

Table 3.1: Number of alignments on the simulated datasets. For each considered sample (5M and 10M) and for each considered tool (ASGAL and STAR), the number of reads simulated from the 656 considered genes is shown along with the number of Unmapped reads, the number of reads mapped only once, the number of reads mapped multiple times, and the total number of mapped reads.

from which the read was simulated.

Observe that the fewer "Partially Mapped" alignments is a consequence of the advantage of aligning directly to the splicing graph. Indeed, by investigating the "Partially Mapped" alignments of STAR, we found that the vast majority of these alignments (more than the 75%) place some read bases on an intron: this situation mainly occurs when the first (last) bases of an intron (exon) are equal to the first (last) bases of an exon (intron). By using the splicing graph, it is possible to avoid these situations since it forces the alignments to be placed (when possible) on the known exons of a gene.

On the other hand, the higher number of alignments to positions different from the one of extraction is a consequence of the fact that ASGAL works at the gene level and can produce multiple primary alignments, of which only one can be the correct one.

To perform a more thorough comparison, we also analyzed the number of incomplete alignments due to read truncation reported and each tool's tolerance for mismatches. Figure 3.8 and Figure 3.9 show the results of this analysis.

As previously stated, ASGAL aligned 91% of the input reads whereas STAR aligned 97% of the reads. Such difference in alignment performance is due to two reasons. First of all, with default parameters, the number of allowed errors in an

Sample	Tool	Alignments	Perfectly mapped	Partially mapped	Differently mapped
5M	ASGAL	2,951,893	2,907,881 (98.51%)	34,047 (1.15%)	9,965 (0.34%)
	STAR	3,150,948	3,072,183 (97.50%)	76,599 (2.43%)	2,166 (0.07%)
10M	ASGAL	5,960,322	5,879,604 (98.65%)	66,463 (1.11%)	14,255 (0.24%)
	STAR	6,355,862	6,201,270 (97.56%)	150,373 (2.37%)	4,219 (0.07%)

Table 3.2: Read placement accuracy on the simulated datasets. For each considered sample (5M and 10M) and for each considered tool (ASGAL and STAR), the number of alignments produced by each tool is shown along with the number of perfectly mapped alignments, *i.e.* the alignments which place all the read bases in the correct position, the number of partially mapped alignments, *i.e.* the alignments which place some (but not all) read bases in the correct position, and the number of differently mapped alignments, *i.e.* the alignments which place all the read bases in a position different from the one from which the read has been simulated.

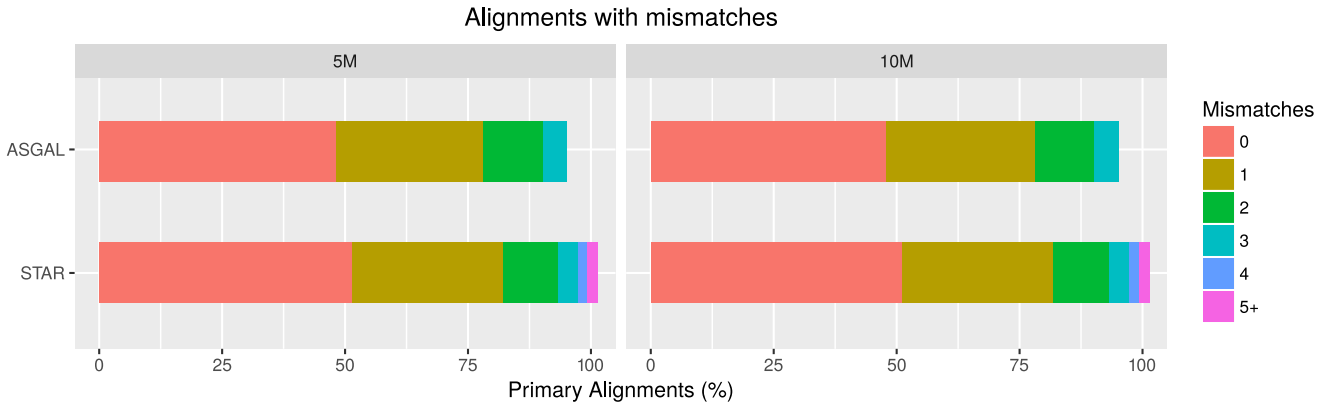


Figure 3.8: Mismatch frequencies. The figure shows for each considered sample (5M and 10M) and for each considered tool (ASGAL and STAR), the percentage of reads aligned divided by number of mismatches. The different colors indicates the number of mismatches.

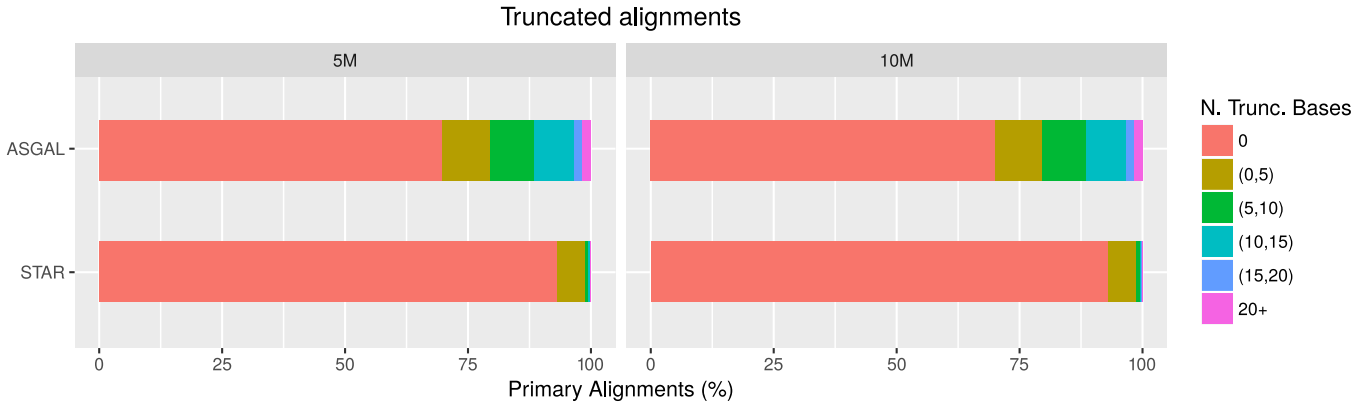


Figure 3.9: Reads truncation frequencies. The figure shows for each considered sample (`5M` and `10M`) and for each considered tool (`ASGAL` and `STAR`), the percentage of incomplete alignments due to reads truncation. The different colors indicates the number of truncated bases.

alignment is smaller for `ASGAL` than for `STAR`. Indeed, as shown in Figure 3.8), `STAR` reported alignments with more than 3 errors whereas, by setting the β parameter of `ASGAL` to 3% of the read length (that is 100bp), we forced our tool to output only alignments with less than 3 errors. Secondly, differently from `STAR` that is able to align reads to intronic regions, `ASGAL` only aligns reads to exonic regions and it cannot correctly align reads covering intronic regions or long novel exons.

Figure 3.9 reports the number of truncated alignments. We can observe that `ASGAL` outputs more incomplete alignments than `STAR` and this is mainly due to the choice of parameter L. Indeed, `ASGAL` builds each alignment starting from anchors of at least L bases. If a prefix or a suffix of some read is not covered by any anchors, for example due to a sequencing error or a junction near one of the read end, `ASGAL` may output a truncated alignment since it cannot correctly align that portion of the read. Although such behavior might remove parts of the alignments which are useful for the detection of alternative splicing events, we will show that these truncations do not affect significantly the step of alternative splicing event identification. However, at the cost of increasing the running time, it is possible to decrease the number of truncated alignments by setting a smaller value of L.

Finally, we analyzed the computational resources required by the two tools.

Sample	**Tool**	**Time** *(min)*	**Memory Peak** *(MB)*
5M	ASGAL	475	249
	STAR	179	7,921
10M	ASGAL	945	247
	STAR	197	7,921

Table 3.3: Computational resources required by ASGAL and STAR to align the simulated datasets (5M and 10M). These results are shown in terms of time (minutes) and memory peak (MegaBytes). These results take into account both the index and the alignment steps performed by the tools.

We ran both the tools using a single thread and we reported in Table 3.3 their total running time and their memory usage, computed using the /usr/bin/time system tool. As expected, since ASGAL works at the gene level, it required more time and less memory than STAR. Indeed, we ran STAR on each chromosome whereas we ran ASGAL on each gene and thus we needed to repeat its execution for the total number of processed genes. However we note that the slowest run of ASGAL on the 5M dataset took only 99 seconds whereas the slowest run on the 10M dataset took only 184 seconds. Since each run of ASGAL is independent from the others, by spreading a many-gene computation over multiple cores, we can reduce its total running time, proportionally to the number of cores. For instance, assuming to use 16 cores, ASGAL would require less than 67 minutes to align the 5M datasets to the 656 considered genes and less than 125 minutes to align the 10M datasets. We finally note that also STAR can be run using multiple threads. Nevertheless, we did not compare the time performance of ASGAL and STAR when ran in parallel. Indeed, the focus of this analysis was evaluating the considered tools in terms of accuracy and quality of the alignments and not in terms of efficiency.

Validation of the event identification step

In the second part of our experimental analysis on simulated data, we compared the alternative splicing event identification step of ASGAL with three other well-

known tools for the detection of alternative splicing events from RNA-Seq data: `SplAdder` (version 1.0.0), `rMATS` (version 4.0.2 turbo), and `SUPPA2` (version 2.3). Although these tools work under different assumptions than `ASGAL`, we decided to include them in our comparison anyway.

Indeed, `SplAdder` can identify an alternative splicing event only if all the isoforms involved in the event are supported by input samples. According to `SplAdder`'s supplementary material, the default behaviour of `SplAdder` can be modified by adapting different parameters that guide the confirmation process of each alternative splicing event detected. However it is not an easy task to modify these parameters since they are hard-coded and it is not even clear how to choose the best values without the risk of introducing undesired behaviors.

On the other hand, `rMATS` is able to detect only alternative splicing events that use already annotated splice sites whereas `SUPPA2` can quantify only alternative splicing events extracted from the input annotation, *i.e.* non novel events.

Differently from such tools, `ASGAL` is specifically designed to detect alternative splicing events that are novel with respect to an input annotation and it can also identify alternative splicing events even if only a single isoform inducing the event is expressed in the input sample. Indeed, `ASGAL` uses the input annotation as a reference for the identification of the novel alternative splicing events. This case is especially important, since usually there is a single transcript expressed per gene, when considering only a sample [129].

We did not include `MAJIQ` in the experimental comparison since the tool focuses on Local Splicing Variations (LSVs). Although LSVs capture previously defined types of alternative splicing as well as more complex transcript variations, `MAJIQ` does not provide a direct way to map one kind of event into the other one. Moreover, we did not include `LeafCutter` in our experiments since the tool focuses on introns which model complex alternative splicing events and there is no easy way to extract from them the simpler (and classic) alternative splicing events.

The goal of our experimental analysis on simulated data was to evaluate the ability of our tool in detecting alternative splicing events that are novel with respect to the input annotation, *i.e.* alternative splicing events that are

not already contained in the input annotation. In other words, we focused our analysis on events not already described by some transcripts of the input annotation but that are supported by the input RNA-Seq sample.

For this reason, we created a set of reduced annotations by removing some transcripts from the annotations of the considered genes. In such a way, by providing the tools with these reduced annotations and an RNA-Seq sample containing reads simulated from the original annotation, we assessed their capability to detect novel alternative splicing events. Observe that `ASGAL` is specifically designed to enrich a gene annotation with novel alternative splicing events supported by a RNA-Seq sample and thus this analysis better reflects its real effectiveness.

We obtained the reduced annotations in the following way. First of all, we used `AStalavista` to extract all the alternative splicing events contained in the annotations of the 656 considered genes. This resulted in a total of 2568 alternative splicing events: 1574 exon skippings, 416 alternative acceptor sites, 290 alternative donor sites, and 288 intron retentions. Then, for each gene and for each event identified by `AStalavista`, we created a new reduced annotation containing all the transcripts except those responsible for such event. We focused our attention on exon skippings, alternative splice sites (both acceptor and donor), and intron retentions caused by the insertion of a new intron inside an annotated exon. Since the alternative splice site can consist of both shortening or extending an annotated exon, we added both these cases to the events considered in the experimental evaluation. On the other hand, we did not consider the possible insertion of a new exon inside an intron and the intron retention caused by the union of two annotated exons, since detecting such events is impossible using a splicing graph as unique source of information. Moreover, when different events on the same gene produced the same reduced annotation, we considered the annotation only once. We obtained a total of 3274 alternative splicing events and 2792 reduced annotations.

Differently from [69], where the reduced annotation provided as input to the tools contained only the first transcript of each gene, we generated the gene annotations by keeping all the transcripts except the ones including the intron supporting the considered alternative splicing event. In these terms, `ASGAL` is

less general than the other tools in detecting novel exons that appear in intronic regions and that can only be detected by aligning reads to large intronic region. Indeed, the exons involved in the considered novel events must be present in the reduced annotation to allow `ASGAL` to detect them. However, observe that `ASGAL` uses the genomic regions close to exon splice sites to detect novel exons that are variants of existing ones.

For each gene and for each reduced annotation, we ran the tools on the two considered datasets of reads, namely `5M` and `10M`. We recall here that these RNA-Seq samples were simulated with `Flux Simulator`, cover 1000 randomly selected human genes, and were used in the experimental analysis performed in [69].

For each type of alternative splicing event we analyzed the predictions over the set of the 656 considered genes, computing the corresponding values of precision, recall, and F-measure. More precisely, given a gene and its reduced annotation, we consider as ground truth the set of events found by `AStalavista` in the original annotation of the considered genes.

To compute the values of precision, recall, and F-measure, we considered the number of events that are in the original annotation but not in the reduced one and are found by the tools as *true positives*, the number of events that are in the original annotation but not in the reduced one and are *not* found by the tool as *false negatives*, and the number of events found by the tools and not in the annotation as *false positives*.

We reported in Table 3.4 the quality results, for the different alternative splicing events, obtained by `ASGAL`, `SplAdder` and `rMATS`, for which we used `STAR` to compute the alignments, and `SUPPA2`, for which we used `Salmon` (version 0.9.1) to obtain the transcript quantification.

The results show that `ASGAL` achieved the best values of precision, recall, and F-measure in almost all the alternative splicing events with the only exception of the recall of alternative splice sites. We investigated those cases and we found that our method applies strict criteria in detecting the alternative splice site events that extend an annotated exon. As previously described, to detect this kind of event, `ASGAL` requires that the prefix of each read aligns near the end of an exon, the suffix near the start of another exon, and a central substring does not align to any exon, *i.e.* there is a gap on the read that represents the

extension of one of the two used exons. Therefore, ASGAL requires the presence of sufficiently long anchors on two different exons, typically 15bps, related to the length L. As a consequence, our method detects an alternative splice site event extending an exon when the length of the extension does not exceed the length of the read minus twice the length of L (the two anchors). By these requirements, our method may not be able to detect alternative splice site extending an exon of several bases, as observed in the cases analyzed in our experimental analysis: for this reason ASGAL shows a lower recall on alternative splice site events.

However, our method achieved the best values of F-Measure in all the alternative splicing event types hinting that our criteria are well balanced and highlighting the ability of ASGAL in detecting novel alternative splicing events.

To better analyze our results, we also decided to manually examine the exon skipping events found by ASGAL but not by the other tools. In this analysis, we considered only SplAdder and rMATS since SUPPA2 did not find any novel exon skipping event. We found that most of the events not detected by SplAdder are due to the fact that only one of the two isoforms involved in the event is supported by the input alignments. As said previously, to confirm an alternative splicing event, SplAdder requires that all the isoforms involved in the event are supported by the sample. Regarding rMATS, instead, we found that it does not output most of the events involving the skip of multiple exons. Moreover, by increasing the number of reads in the input set, all methods almost always achieve better recall with a slightly worse precision, since a higher coverage allows to detect a higher number of supported introns that are used to detect alternative splicing events.

As it is possible to notice from Table 3.4, results of both rMATS and SUPPA2 are incomplete: this was expected since these two methods are not designed to detect novel events. More precisely, rMATS is not able to detect alternative splicing events involving novel splice sites and, for this reason, it is able to detect only exon skipping events. SUPPA2 only detects and quantifies alternative splicing events that are already present in the input annotation.

To have a thorough comparison with these tools, we set up a second analysis in which we used each tool to detect already annotated alternative splicing events. For this purpose, we provided the tools with the original annotations of each considered gene and the same RNA-Seq samples used in the previous

Tool	Measure	5M				10M			
		ES	*A3*	*A5*	*IR*	*ES*	*A3*	*A5*	*IR*
ASGAL	Prec	0.997	0.955	0.905	0.862	0.995	0.938	0.895	0852
	Rec	0.917	0.741	0.737	0.674	0.963	0.789	0.781	0.681
	F-M	0.955	0.835	0.812	0.756	0.979	0.857	0.834	0.757
SplAdder	Prec	0.885	0.612	0.475	0.299	0.874	0.642	0.495	0.272
	Rec	0.802	0.884	0.821	0.531	0.848	0.925	0.891	0.521
	F-M	0.841	0.723	0.602	0.383	0.860	0.758	0.637	0.357
rMATS	Prec	0.996	-	-	-	0.997	-	-	-
	Rec	0.860	-	-	-	0.863	-	-	-
	F-M	0.923	-	-	-	0.925	-	-	-
SUPPA2	Prec	-	-	-	-	-	-	-	-
	Rec	-	-	-	-	-	-	-	-
	FM	-	-	-	-	-	-	-	-

Table 3.4: Quality measures in detecting novel alternative splicing events on the simulated datasets with 5M and 10M reads. Results obtained by ASGAL, SplAdder and rMATS (for which we used STAR to compute the alignments), and SUPPA2 (for which we used Salmon to obtain the transcript quantification) are reported. Precision (Prec), Recall (Rec), and F-Measure (F-M) achieved on the simulated datasets in detecting novel alternative splicing events: exon skipping (ES), alternative acceptor site (A3), alternative donor site (A5), and intron retention (IR). A dash "-" means that the considered tool is not designed to detect that type of *novel* alternative splicing events.

analysis. Table 3.5 reports the results obtained in this setting. Results of the tools are similar, with ASGAL performing slightly better on exon skipping events, whereas SUPPA2 outputs better predictions of alternative donor and acceptor sites, and both SUPPA2 and rMATS have better results on intron retention events.

We then carefully inspected the results obtained by ASGAL and we observed that more than 85% of the false positives alternative splice site events in Table 3.5 and more than 98% of false positives intron retention events are due to the following reason. As said above, ASGAL is designed to detect events that are novel with respect to the input gene annotation. For this reason, even in the annotated case, ASGAL looks for potential novel alternative splicing events by extracting from the alignments to the splicing graph those introns that may support the presence in the experiment of an isoform related to an event that is alternative with respect to an already annotated isoform. Moreover, by using only the computed set of introns to detect alternative splicing events, ASGAL's events prediction show a higher number of false positives, as illustrated in Figure 3.10: even though the computed alignments are correct, some introns can be misclassified as alternative splice site or intron retention. In more detail, this misclassification occurs when the considered intron is the first (or the last) intron of a transcript: in these cases, by using only the information provided by the introns, it is not possible to fully understand if the considered intron supports a real alternative splicing event. Consequently, a lower precision cannot be imputed to the quality of the alignment performed by ASGAL. Indeed, the false positive cases described above could be eliminated by using more conservative rules. In any case, we believed that the actual rules used by ASGAL are well-balanced: they produce higher precision in detecting novel event as shown in Table 3.4 as well as good results in detecting RT-PCR validated alternative splicing events, as discussed in the next section.

Finally, we discuss the efficiency of the tested methods. To this purpose, we retrieved the running time and the maximum memory used by each tool in the detection of annotated events on the 10M dataset using the /usr/bin/time system tool We decided to analyze the performance during the detection of annotated events to have a fair comparison, since rMATS and SUPPA2 are meant to detect only annotated events and do not provide any useful information when

Tool	Measure	5M				10M			
		ES	*A3*	*A5*	*IR*	*ES*	*A3*	*A5*	*IR*
ASGAL	Prec	0.999	0.850	0.702	0.657	0.999	0.846	0.703	0.642
	Rec	0.924	0.788	0.774	0.719	0.966	0.814	0.799	0.722
	F-M	0.960	0.818	0.736	0.687	0.982	0.830	0.748	0.680
SplAdder	Prec	0.963	0.844	0.734	0.513	0.957	0.857	0.733	0.450
	Rec	0.822	0.927	0.899	0.552	0.855	0.947	0.936	0.531
	F-M	0.887	0.884	0.808	0.532	0.903	0.900	0.822	0.487
rMATS	Prec	0.995	1	1	0.976	0.996	1	1	0.976
	Rec	0.905	0.685	0.755	0.830	0.905	0.685	0.755	0.830
	F-M	0.948	0.813	0.860	0.897	0.949	0.813	0.860	0.897
SUPPA2	Prec	1	0.880	0.754	0.976	1	0.880	0.754	0.976
	Rec	0.894	1	1	0.830	0.894	1	1	0.830
	F-M	0.944	0.936	0.860	0.897	0.944	0.936	0.860	0.897

Table 3.5: Quality measures in detecting annotated alternative splicing events on the simulated datasets with 5M and 10M reads. Results obtained by ASGAL, SplAdder and rMATS (for which we used STAR to compute the alignments), and SUPPA2 (for which we used Salmon to obtain the transcript quantification) are reported. Precision (Prec), Recall (Rec), and F-Measure (F-M) achieved on the simulated datasets in detecting annotated alternative splicing events: exon skipping (ES), alternative acceptor site (A3), alternative donor site (A5), and intron retention (IR).

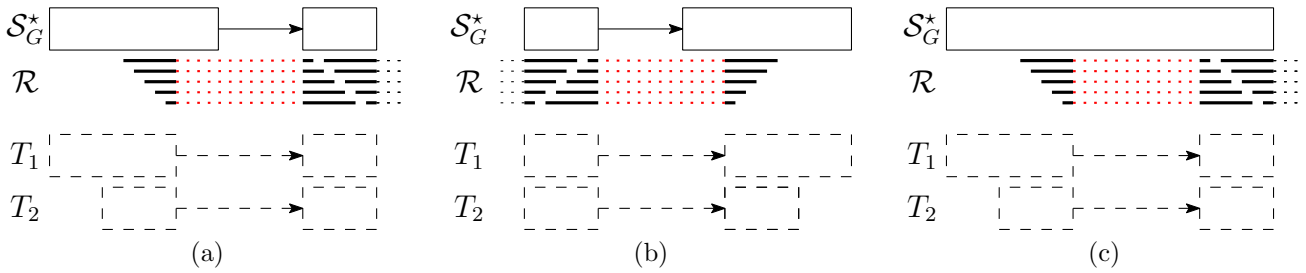


Figure 3.10: Examples of splicing event misclassification. The figure depicts three situations in which `ASGAL` may detect a false positive event: an alternative donor site in (a), an alternative acceptor site in (b), and an intron retention in (c). The black arrows represent an annotated intron (in the first two cases), whereas the red dotted lines represent the novel intron supported by the input sample with respect to the annotation, represented by $\mathcal{S}^\star_G$. In cases (a) and (b), the novel event induces an alternative acceptor and an alternative donor splice site (respectively) with respect to the intron in the annotation, and in case (c) the novel intron is inside an already annotated exon. On the assumption that the reads come from a hypothetical novel transcript T_1, `ASGAL` outputs a true positive event. Indeed, all the events refer to an annotated exon, thus one (in cases (a) and (b)) or two splice sites (in case (c)) involved in the predicted event are already annotated in $\mathcal{S}^\star_G$. On the other hand, if the aligned reads come from a hypothetical novel transcript T_2, `ASGAL` produces a false positive event: it outputs the events with respect to the annotated exon but the true events refer to a novel exon having both splice sites not annotated in $\mathcal{S}^\star_G$.

detecting novel events.

As described before, ASGAL is composed of two steps: the alignment of the reads and the detection of the events. These two steps required 945 and 60 minutes, respectively, and the memory usage was 760MB.

For SplAdder we considered the time required by STAR to align the reads to the chromosomes and by the tool itself to detect the events. The first step required 45 minutes whereas the second step required 36 minutes. The memory usage was 5.8GB and was due to the alignment step.

Similarly to SplAdder, rMATS aligns the reads using STAR. Therefore the first step of rMATS requires the same time as SplAdder (45 minutes). Detecting the events using rMATS, on the other hand, requires 268 minutes. The main memory usage of this tool is, again, due to the alignment step of STAR and is equal to 5.8GB.

Finally, SUPPA2 is composed of three steps: quantifying the transcripts using Salmon, generating events from the annotation using SUPPA2 generate, and computing the psi-value, *i.e.* the relative abundance value per sample, of the events, using SUPPA2 psi. These steps require 130 minutes, 19 minutes, and 19 minutes respectively. The main memory usage is due to the quantification step of Salmon and is equal to 192MB.

Note that in the previous analysis we do not consider the time required by STAR and Salmon to index the genome and the transcriptome, respectively. In fact, such indexes could already be available from previous runs of the tools. Nevertheless, we report the times for these steps for sake of completeness. STAR required 152 minutes and 7.9GB to index the chromosomes of the human genome whereas Salmon required 2 minutes and 14MB to index its transcriptome.

3.4.3 Experimental analysis on real data

To better assess the efficacy of our approach, we also performed an experimental analysis on a real dataset of RNA-Seq reads. Inspired by the experimental analysis performed in the SUPPA2 paper [158], we considered a set of 83 RT-PCR validated alternative splicing events [12]. More precisely, the experiment per-

formed in [12] consists of 3 samples[2] in which there is a double knockdown of the TRA2A and TRA2B splicing regulatory proteins and 3 control datasets[3]. The goal of the experimental analysis of SUPPA2 in [158] was to identify the 83 RT-PCR validated alternative splicing events in these knockdown versus control datasets. Since in 2 of these 83 events the positions of the intron(s) involved in the event were missing, we could not use such events to compare the predictions of the tools. For this reason, we decided to remove such events from the set, resulting in a set of 81 events on which we tested ASGAL, rMATS, SUPPA2, and SplAdder, with the specific goal of identifying the RT-PCR validated alternative splicing events. More precisely, we ran the tools on the 3 replicate datasets with the knockdown of the two splicing regulatory proteins (SRR1513332, SRR1513333, and SRR1513334).

We ran ASGAL in genome-wide mode on each dataset. Moreover, we provided the alignments obtained with STAR to SplAdder and rMATS and the quantifications obtained with Salmon to SUPPA2. We compared the results obtained on each dataset with the tested methods and, in particular, we considered all the events output by such tools that were in the list of events under analysis.

In Figure 3.11 we show a comparison of the results obtained by the tools on the 3 knockdown datasets. As it is possible to observe, rMATS was the tool that was able to detect more events (78 on SRR1513332, 78 on SRR1513333, and 77 on SRR1513334). Similarly, SUPPA2 identified 65 events in each of the 3 datasets, whereas ASGAL predicted 63, 59, and 61 events on the SRR1513332, SRR1513333, and SRR1513334 dataset, respectively. Finally, SplAdder was able to identify only 13, 13, and 12 RT-PCR validated alternative splicing events on the SRR1513332, SRR1513333, and SRR1513334 dataset, respectively. We note that the events not identified by ASGAL show an extremely low support.

To better validate the results obtained by ASGAL, we extracted from the alignments computed with STAR the spliced alignments supporting each considered event. More in detail, since each RT-PCR validated event is an exon skipping event, we counted the number of spliced alignments supporting the intron that confirms the skipping of one or more exons. We report the result of this analysis

[2] SRA accession numbers SRR1513332, SRR1513333, and SRR1513334.

[3] SRA accession numbers SRR1513329, SRR1513330, and SRR1513331.

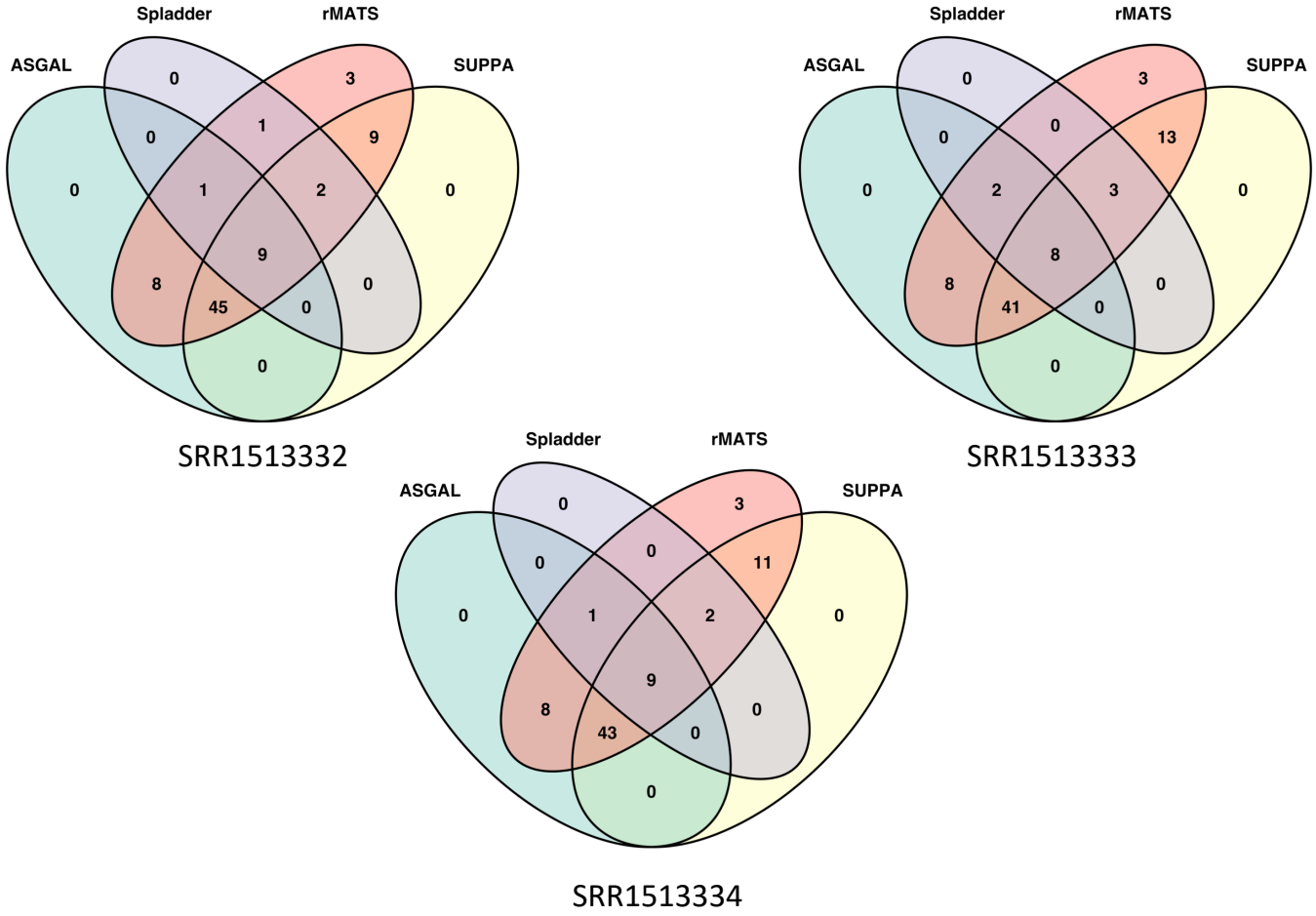


Figure 3.11: Results on RT-PCR validated events. Venn diagram showing the overlaps in results obtained by `ASGAL`, `SUPPA2`, `rMATS`, and `SplAdder`, on the 3 knockdown dataset. The results are expressed as the number of RT-PCR validated events detected by the various tools.

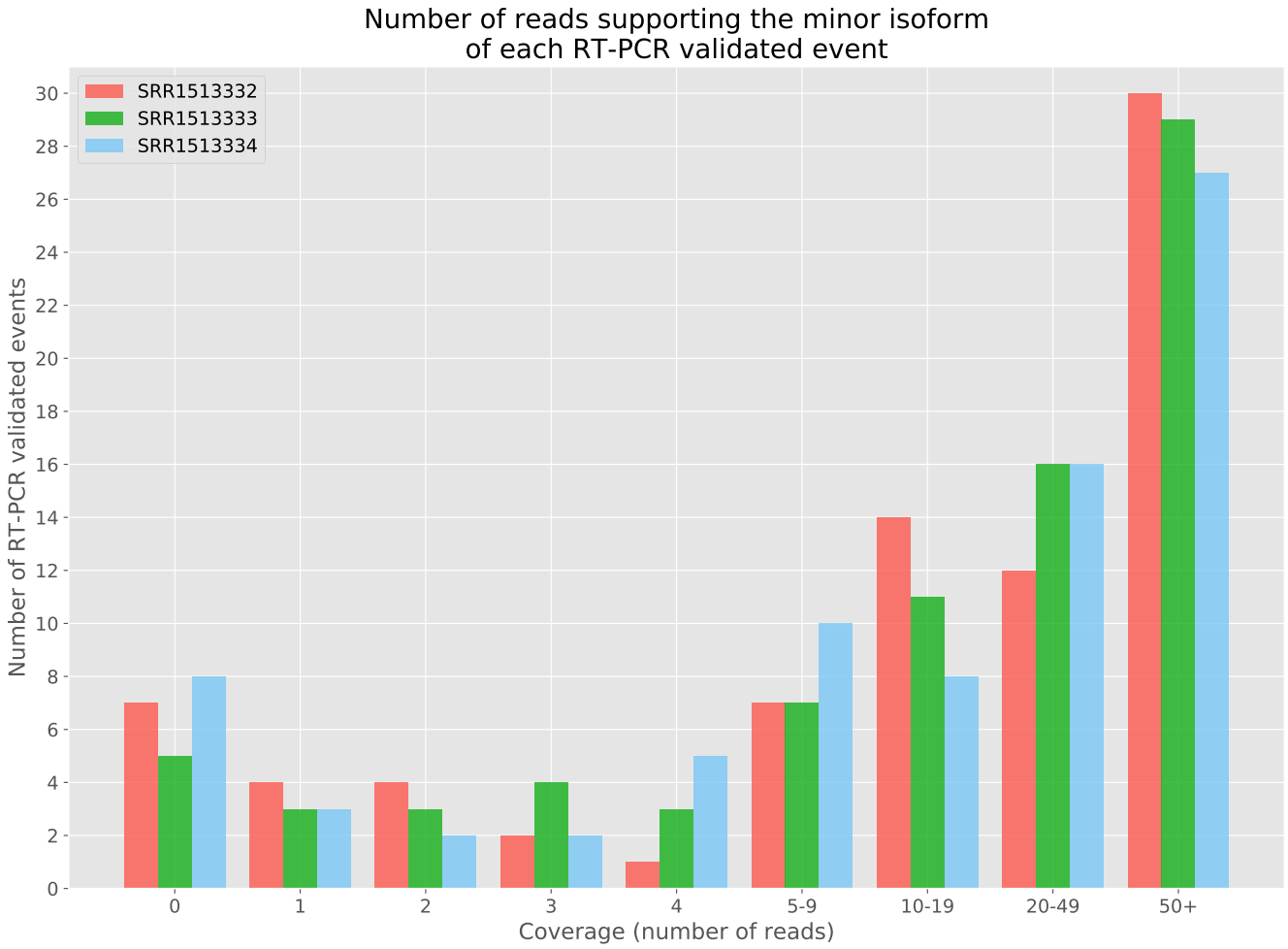


Figure 3.12: Coverage of RT-PCR validated events. Bar chart showing the coverage of the minor isoform of each RT-PCR validated events. The coverage is expressed as the number of spliced alignments supporting the intron that skips the exon(s).

in Figure 3.12, where the coverage of the 81 considered RT-PCR validated events is shown.

These results highlight the ability of ASGAL in predicting alternative splicing events from real datasets of RNA-Seq reads, especially if compared to SplAdder which is the most similar tool. We note one more time that, to detect an alternative splicing event, SplAdder needs that all the isoforms involved in the event are supported by the reads in the sample. Thus, we analyzed how many of the RT-PCR validated events are supported by a single isoform. The obtained results show that, in the three considered datasets, 59, 54, and 65 events, respectively, are not supported by both the isoforms involved in the event.

Moreover, the performances achieved by ASGAL on the tested datasets are similar, in terms of number of detected alternative splicing events, to those of SUPPA2, which still lacks in detecting novel events. The only tool that slightly outperformed ASGAL in this experimental analysis is rMATS. Anyway, we note one more time that this latter method is not able to detect alternative splicing events involving novel splice sites that are not already in the considered annotation.

Moreover, ASGAL and all the considered tools identified more alternative splicing events than the RT-PCR validated ones. To better analyze such behaviour, we considered the genes involved in the 81 RT-PCR validated alternative splicing events and we checked if the events additionally reported by ASGAL were also detected by the other considered tools. Table 3.6 summarizes the results of this analysis: except for intron retention events, the majority of the alternative splicing events identified by ASGAL were also reported by rMATS and SUPPA2.

Finally, we ran our tool providing a reduced annotation. As done in our experiments on simulated data, we removed all the transcripts which include the intron supporting the considered event and we used this annotation as input for our tool. In this way, by keeping the evidence of these events only in the RNA-Seq data, we could test the ability of our tool in detecting novel alternative splicing events from real data. In this setup ASGAL predicted the same events predicted using the full annotation, *i.e.* 63, 59, and 61 events on the SRR1513332, SRR1513333, and SRR1513334 dataset, respectively.

	SRR1513332				SRR1513333				SRR1513334			
	ES	*A3*	*A5*	*IR*	*ES*	*A3*	*A5*	*IR*	*ES*	*A3*	*A5*	*IR*
ASGAL	343	168	141	56	356	170	138	50	323	149	140	51
SplAdder	33	18	3	2	35	17	2	1	28	18	2	1
rMATS	314	65	48	18	328	65	54	16	302	63	46	18
SUPPA2	193	83	78	17	191	82	78	16	189	83	76	17

Table 3.6: For each considered knockdown datasets and for each considered event type, we report the number of alternative splicing events identified by ASGAL with respect to the considered genes, *i.e.* the genes involved in the 81 considered RT-PCR validated alternative splicing events, and how many of these events were also identified by the other tools considered in our analysis.

3.5 Final remarks and future directions

In this chapter we presented ASGAL, a tool for predicting alternative splicing events via an accurate splice-aware alignment of RNA-Seq data against a splicing graph. Differently from other approaches proposed in the literature, ASGAL detects the alternative splicing events supported by the input RNA-Seq sample that are novel with respect to the input gene annotation. Indeed, ASGAL detects alternative splicing events by analyzing differences at the intron level between the known annotation and the introns supported by the alignments computed against the splicing graph.

While the spliced alignment to a reference is a well understood notion, how to align a sample directly against a splicing graph is not a so obvious notion. For this reason, we investigated the problem of optimally aligning reads to a splicing graph by formalizing the notion of *spliced graph-alignment* and by proposing an algorithmic approach to compute optimal spliced graph-alignments.

Note that our notion of spliced graph-alignment is tailored to the detection of alternative splicing events that are either classic or a combination of two different simple events. However, such a notion deserves to be further investigated to detect more complex combinations of alternative splicing events, such those

modeled with the notion of Local Splicing Variations. This will be the goal of a future development of the tool.

Future works will also focus on allowing `ASGAL` to be used natively in a genome-wide analysis by improving its pre-processing step or by directly improving its code.

To the best of our knowledge, `ASGAL` is the first tool for computing splice-aware alignments of RNA-Seq data to a splicing graph. Compared with current tools for the spliced alignment to a reference genome, `ASGAL` produces high quality alignments that can be used to accurately detect alternative splicing events. However there is still room for further improvements in the direction of using `ASGAL` for confirming alternative splicing events already contained in the input annotation, for predicting novel alternative splicing events that involve long intronic regions of the reference genome, and for performing differential analysis between multiple samples.

The experimental analysis we performed shows the advantages of `ASGAL` in using a splice-aware aligner of RNA-Seq data to detect alternative splicing events that are novel with respect to a gene annotation, *i.e.* the events which involve novel splice sites.

In this sense `ASGAL` can be used to enrich a given annotation with novel alternative splicing events in order to allow a downstream tool for differential alternative splicing analysis such as `SUPPA2` to also quantify these new events. Indeed, compared to other approaches, `ASGAL` can work in presence of a poor gene annotation with the main goal of enriching its structure with novel events.

A natural extension of the `ASGAL` approach (not yet investigated) is the detection of alternative splicing events in a de-novo framework, where only the reference transcriptome is known. In this context a draft splicing graph may be built from the mRNA sequences related to the reference transcriptome and then it can be used to infer alternative splicing events by using the `ASGAL` procedure.

Chapter 4

Known Variants Genotyping

In this chapter we will discuss the second contribution of this thesis, that is an alignment-free approach for genotyping a set of known variants (`MALVA`, genotyping by Mapping-free ALternate-allele detection of known VAriants). The problem tackled can be formulated as follows:

Input:	NGS sample and dataset of known variants (SNPs and indels)
Output:	genotype of each known variant

Highlights

- `MALVA` is the first alignment-free approach for genotyping indels and multi-allelic variants
- `MALVA` calls correctly more indels than the most widely adopted genotyping pipelines
- `MALVA` proved to be a valid and faster alternative to alignment-based approaches

Outline This chapter is organized as follows. In Section 4.1 we introduce the context and the motivations behind our work. In Section 4.2 we present some definitions that are needed in Section 4.3, where we thoroughly detail the proposed approach. In Section 4.4 we describe our implementation and the

experimental analysis we performed to asses its performance, comparing it with state-of-the-art tools. Finally, in Section 4.5 we draw conclusions and sketch possible future directions of this work.

4.1 Context and Motivations

The discovery and characterization of Single Nucleotide Polymorphisms (SNPs) and indels in human populations is crucial in genetic studies [153]. A main challenge consists in efficiently characterizing the variations of a freshly-sequenced individual with respect to a reference genome and the available genetic variations data. Typically, variant calling (or genotyping) requires the detection of the variants (SNPs or indels) and the identification of its genotype *i.e.* the alleles expressed on each haplotype of the organism under investigation. For example, in diploid organisms, a genotype is a pair of alleles.

Standard pipelines for variant calling include aligning the sequenced reads with softwares like `BWA` [88] and `Bowtie2` [85] and then calling the genotypes with tools like `GATK` [101] and `BCFtools` [86]. Due to the high computational complexity of read alignment, such approaches can be highly time consuming, thus impractical for clinical applications, where time is often an issue.

Assembly-based methods such as `Cortex` [63] and `discoSnp++` [124] form another line of research. These approaches avoid the read alignment step by directly assembling the input set of reads in a de Bruijn graph that is successively analyzed to detect and call the variants. Such approaches show lower accuracy than alignment-based approaches [128] and are still highly time consuming, due to the computationally expensive task of read assembly.

Recent tools for variant calling like `Graphtyper` [40] and `vg` [46] are based on a graph representation of a set of genomes, called *variation graphs*. Such approaches consider a set of genomes to avoid biases introduced by considering only one genome as a reference [27]. However, they are heavy in both computational space and time. Indeed, the size of variation graph indexes may be subjected to an exponential growth in the number of variants, and indexes are typically static, requiring a great deal of computational resources to be updated with newly discovered variants.

When the task is to call the genotype in positions where variants have been previously annotated, alignment-free methods come to the aid. This is a typical case in a medical setting, where the discovery of new variants is not desired, but, rather, what is important is to know the genotype at certain loci that are already established to be of medical relevance. Recent alignment-free tools, such as `FastGT` [118], `LAVA` [139], and `VarGeno` [152], are able to genotype a set of known SNPs up to an order of magnitude faster than the usual alignment-based methods. A major shortcoming of these tools is the large memory requirement, that can easily exceed hundreds of GB of RAM. Indeed, their strategy is to model each known SNP as a k-mer, *i.e.* a string of length k, and to store this mapping in a sort-of dictionary. A set of additional information, such as the k-mers frequency in the input sample, is associated to each k-mer and it is used to genotype the variants. Thanks to this strategy, these approaches are able to call a set of known bi-allelic SNPs in a very efficient way but, on the other hand, cannot genotype multi-allelic SNPs and indels, *i.e.* short insertions and deletions of nucleotides.

However, genotyping indels is of utmost importance in genetic studies [9]. Indeed, indels are believed to represent around 16% to 25% of human genetic polymorphism [104]. Moreover, the presence of indels is associated with a number of human diseases [110, 58, 83]. For instance, cystic fibrosis [110], lung cancer [136], Mendelian disorders [92], and Bloom syndrome [70] are all known to be closely correlated to indels.

Nevertheless, indels are particularly challenging to call: reads covering an indel are more difficult to align to the correct location and, most of the time, even though the read is mapped correctly, the indel is wrongly placed [105]. To improve their accuracy, alignment-based approaches apply different technique such as local realignment around indels [35], haplotype assembly [126, 128] or a combination of both [108]. But this step increases the time requirements of such approaches.

Due to the high complexity of genotyping indels, many specialized approaches have been proposed in the literature for genotyping specifically only this kind of variations [176, 3, 13].

In this chapter we describe `MALVA`, a rapid alignment-free method to genotype

a set of known (*i.e.* previously characterized) variants from a sample of reads. Similarly to previous alignment-free approaches, `MALVA` is a word-based method: each allele of each known variant is assigned a *signature* in the form of a set of k-mers, which allows to efficiently model indels and close variants. The genotypes of such variants are then called according to the frequency of such signatures in the input read sample.

Based on the well-known Bayes' formula, we also design a new rule to genotype multi-allelic variants (*i.e.* variants with more than one known alternate allele): even if such variants are trickier to genotype than bi-allelic ones, `MALVA` is still able to achieve high precision and recall, as revealed in the real-data experiments we conducted.

`MALVA` directly analyzes a sample leveraging on the information of the variants included in a VCF file, that is the standard format for representing and storing a collection of variants, also adopted by the 1000 Genomes Project [150]. To the best of our knowledge, `MALVA` is the first alignment-free tool able to call multi-allelic variants and indels. Moreover, it proved to be the only alignment-free tool capable of handling the huge number of variants included in the latest version of the VCF released by the 1000 Genomes Project.

4.1.1 State of the Art

We will now review some of the computational approaches proposed in the literature for solving the problem of variant calling. We will focus our attention on *alignment-based*, *assembly-based*, and *alignment-free* approaches.

All the approaches that require to align the input sample against the reference genome fall in the first category, *i.e.* alignment-based. Indeed, these approaches call variants by analyzing the alignments of the input reads. Notable examples of alignment-based approaches are `BCFtools` [86] and `GATK`'s HaplotypeCaller [126].

Approaches such as `BCFtools` are known as "pileup" callers. They analyze each position along the reference genome and, whenever they encounter a mismatch between the reference and the alignments, they evaluate the probability that that position presents a variant.

Differently, `GATK`'s HaplotypeCaller focuses its computation on the so-called

ActiveRegions, that are regions of the genome characterized by a great dissimilarity between reference sequence and alignments. HaplotypeCaller analyzes the read alignments to identify such regions and then it assemblies all the reads covering an ActiveRegion in a *de Bruijn graph* [32], *i.e.* a graph where each vertex is a string of length k, called k-mer, and each edge links two k-mers that are consecutive in the input. Finally, it analyzes the locally-assembled haplotypes to call the genotypes by means of a probability model. In contrast to pileup callers, callers based on local haplotype-assembly show greater accuracy when calling indels and variants, even in regions with a high rate of polymorphisms.

Similarly to `GATK`'s HaplotypeCaller, *assembly-based* approaches assemble the input sample in a de Bruijn graph and then analyze the *bubbles* of this graph, *i.e.* two distinct paths that share the starting and ending vertices, to identify and call variants. However, differently from the previous approaches, such approaches do not require the alignments of the input sample. Such approaches show lower accuracy than alignment-based approaches [128] and suffer from the presence of genomic repetitions. However, assembly-based approaches show their real efficacy and usefulness when non-model organisms have to be analyzed, *i.e.* organisms for which a (high-quality) reference sequence is not available. Examples of assembly-based approaches are `Cortex` [63] and `discoSnp++` [124].

Most notably, `Cortex` uses *colored de Bruijn graphs*, a de Bruijn graphs where each vertex and each edge is associate to a list of colors representing the samples of origin of that element. Colored de Bruijn graphs are a generalization of classical de Bruijn graphs that is currently drawing the attention of more and more researchers [109, 5, 112, 4, 175].

Finally, *alignment-free* approaches call variants directly from the input sample without aligning or assembling it. To do so, these approaches rely on datasets of known variants and, for this reason, differently from the previously described approaches, they cannot be used for *variant discovery*. Such approaches model each known variant as a k-mer and use the k-mers frequency in the input sample to genotype the variants. Examples of alignment-free approaches are `FastGT` [118], `LAVA` [139], and `VarGeno` [152].

`FastGT` strongly relies on a pre-compiled dataset of bi-allelic SNPs and corresponding k-mers, obtained by subjecting the k-mers overlapping known SNPs to

several filtering steps. Such filters consist in removing from the dataset the SNPs for which unique k-mers (*i.e.* not occurring elsewhere in the reference genome) are not observed, those that are closely located (*i.e.* that are less than k bases apart), and others: after the filtering steps, only $\sim$ 30,000,000 bi-allelic SNPs (less than the $\sim$ 40% of the number of bi-allelic SNPs contained in the Phase 3 release of the 1000 Genomes Project) survive and can therefore be genotyped.

Differently from `FastGT`, `LAVA` and `VarGeno` do not rely on any pre-compiled custom-format dataset but they can be used with any dataset of SNPs, such those provided by *dbsnp* [142] or the 1000 Genomes Project [25]. `LAVA` starts by storing the k-mers covering the alleles of the known variants in a set of hash tables alongside with additional information such as the positions at which the k-mer occurs. Then, by analyzing the read sample, it uses approximate k-mer matching to assign to each k-mer stored in the index its frequency in the input sample. Finally, a probabilistic framework is used to call the genotypes.

`VarGeno` builds directly upon `LAVA` to improve its efficiency and accuracy. Differently from `LAVA`, `VarGeno` stores the k-mers and their satellite information by means of a Bloom filter (instead of hash tables) speeding up its computation and exploits quality values associated to each base of the input reads to achieve higher accuracy.

4.2 Preliminaries

Let Σ be an ordered and finite alphabet of size σ and let $t = c_1c_2 \dots c_k$, where $c_j \in \Sigma$ for $j = 1, \dots, k$, be a string of k characters drawn from Σ, we say that t is a k-mer. When a k-mer originates from a double stranded DNA, it is common to consider it and its reverse-complemented sequence as the same k-mer, and to say that the one that is lexicographically smaller among the two is the *canonical* one. In the following, we will abide by this definition and whenever we refer to a k-mer we implicitly refer to its canonical form. Moreover, to avoid k-mers being equal to their reverse-complement, we will only consider odd values of k.

The difference between the genetic sequence of two unrelated individual of the same species is estimated to be smaller than 0.1% [164]; therefore, it is common to represent the DNA sequence of an individual as a set of differences

from a *reference* genome. Indeed, thorough studies [25, 26, 28] of the variations across different individuals encode such information as a VCF (Variant Calling Format) file [31].

We will call *variant* the information encoded by a data line of a VCF file. Besides the genotype data, we are interested in the information carried by the second, fourth, fifth, and eighth field of a VCF line, namely: (i) field `POS` that is the position of the variant on the reference, (ii) field `REF` that is the reference allele starting in position `POS`, (iii) field `ALT` that is a list of alternate alleles that in some sample replace the reference allele, and (iv) field `INFO` that is a list of additional information describing the variant. From the latter list we will get the frequencies of reference and alternate alleles, which are needed to call the genotype of a given individual. We denote with $\texttt{POS}(v)$, $\texttt{REF}(v)$, $\texttt{ALT}(v)$, $\texttt{FREQ}(v)$, and $\texttt{GTD}(v)$ the reference position, reference allele, list of alternate alleles, list of allele frequencies, and genotype data of a variant v, respectively.

The variants we take into account are SNPs, *i.e.* both `REF` and all the elements of `ALT` are single base nucleotides) and indels, *i.e.* `REF` and at least one element of `ALT` are not of the same length. Given an allele a (either reference or alternate) of some variant v, we refer to its sequence of nucleotides as $\texttt{SEQ}(a)$, *i.e.* $\texttt{SEQ}(a)$ is the string that represents a.

Let R be a reference genome and let `V` be a VCF file that describes all the known variants of R. Since the genotype data provides information on the alleles expressed in each genome, another way of thinking of a VCF file is as an encoding of a set of genomes $\mathcal{G}$. Each haplotype of the genomes in $\mathcal{G}$ can be reconstructed by modifying R according to the genotype information associated to each variant. For ease of presentation, in the following we use the term genome and haplotype interchangeably, although each genome of a polyploid organism is composed of multiple haplotypes.

Let $\mathcal{G}$ be the set of genomes encoded by a VCF file and let a be an allele of some variant v, we denote by $\mathcal{G}^a \subseteq \mathcal{G}$ the subset of genomes that include a. We say that a variant v is *k-isolated* if there is no other known variant within a radius of $\lfloor k/2 \rfloor$ from the center of any of its alleles, as formally stated in the following definition.

Definition 1 (k-isolated variant)**.** *A variant v is k-isolated if, for all $a \in$*

$\texttt{ALLELES}(v)$ *and* $g \in \mathcal{G}^a$*, there is no variant* $v' \neq v$ *with an allele* $a' \in \texttt{ALLELES}(v')$ *such that* $g \in \mathcal{G}^{a'}$ *and either* $|\texttt{BEGIN}_g(a') - \texttt{CENTER}_g(a)| \leq \lfloor k/2 \rfloor$ *or* $|\texttt{CENTER}_g(a) - \texttt{END}_g(a')| \leq \lfloor k/2 \rfloor$*, where* $\texttt{ALLELES}(v) = \texttt{REF}(v) \cup \texttt{ALT}(v)$*,* $\texttt{BEGIN}_g(a)$ *is the position of the first base of* a *in* g*,* $\texttt{END}_g(a)$ *the position of the last base, and* $\texttt{CENTER}_g(a)$ *the position of the* $\lceil \frac{|a|}{2} \rceil$*-th base of* a *in* g*.*

The procedure we will present in the next section is heavily based on the concept of *signature* of an allele. Intuitively, a signature of the allele a of a variant v is the k-mer centered in a in some genome g in $\mathcal{G}^a$. Note that, depending on the genomes encoded by the VCF file, specifically if a variant is non k-isolated, its allele might have multiple signatures. Moreover if $\texttt{SEQ}(a)$ is longer than k bases, the previous definition is not well formed, since there is no k-mer that can be centered in a. In this case we define the signature of a as the set of its substrings of length k. The following definition formalizes the notion of signature of an allele.

Definition 2 (Signature of an allele)**.** *Let* $\mathcal{G}$ *be the set of all the genomes encoded by a VCF file* $\texttt{V}$ *and let* k *be an odd positive value. Let* v *be a variant in* $\texttt{V}$*, let* a *be one of the alleles of* v*, and let* $\mathcal{G}^a \subseteq \mathcal{G}$ *be the set of the genomes that include* a*. If* $\texttt{SEQ}(a)$ *is longer than* k *bases, we say that the signature of* a *is the set of all the substrings of length* k *of* $\texttt{SEQ}(a)$*. If* $\texttt{SEQ}(a)$ *is shorter than* k *bases, we say that* $\{x\texttt{SEQ}(a)y\}$ *is the signature of* a *in a genome* g *in* $\mathcal{G}^a$ *if: (i)* $x\texttt{SEQ}(a)y$ *is a* k*-mer, (ii)* $|x| = \lfloor \frac{k-|\texttt{SEQ}(a)|}{2} \rfloor$*, (iii)* $|y| = \lceil \frac{k-|\texttt{SEQ}(a)|}{2} \rceil$*, (iv)* x *is a suffix of the sequence that precedes* a *in* g*, and (v)* y *is a prefix of the sequence that follows* a *in* g*.*

We will refer to the set of all the possible signatures of an allele a as $\texttt{SIGN}(a)$ and we say that k is the *length of the signature*. An example of signatures of an allele is shown in Figure 4.1. Notice that the same k-mer may appear in the signature of more than one allele.

In the following we will leverage on the definition of signature of an allele to detect its presence in an individual without mapping the reads to the reference genome. More precisely, we will analyze whether the k-mers of a given signature are present in the reads and use such information as a hint of the presence of the allele. Unlike other approaches [118], Definition 2 admits the presence of the

$\mathcal{R}$

1	2	3	4	5	6	7	8	9	10	11	12	13
A	G	A	T	C	C	T	G	C	G	A	A	G

	Pos	**Ref**	**Alts**	**Donors**		
v_1	5	C	AAA	0|1	0|0	1|1
v_2	7	T	G	0|1	0|1	0|1
v_3	10	G	A,C	0|0	1|2	2|0

Variant	Allele	Signatures
v_2	a_0 = T	{ {TCC**T**GCG}, {TCC**T**GCA}, {AAC**T**GCC} }
	a_1 = G	{ {AAC**G**GCG}, {TCC**G**GCC} }

Figure 4.1: Signatures of the alleles of a variant. $\mathcal{R}$ is the reference sequence and the table on the right is a VCF information associated to it, representing 3 variants: an indel (v_1), a bi-allelic SNP (v_2), and a multi-allelic SNP (v_3). The last columns of the VCF file carry the genotype information of 3 individuals. The table at the bottom reports the signatures of each allele of variant v_2. Note that there are only 5 signatures although 6 haplotypes are encoded by the VCF file since the second haplotype of the first and third individual are the same. We highlighted in red the genotype information associated to the second haplotype of the second genome and the corresponding signature.

alleles of multiple variants in a single signature, allowing `MALVA` to manage variants that are not k-isolated. Indeed, the set of signatures of an allele represents all the genomic regions where the allele appears in the genomes encoded by the VCF file.

4.3 Method

In this section we will describe `MALVA`, the method we designed to genotype a set of known variants directly from a read sample. The general idea of `MALVA` is to use the frequencies of the signatures of a variant in the sample to call its genotype. The method works under the assumption that given a sample of reads from a genome with standard coverage depth, if an allele is included in the genome then at least one of its signatures must exist as substrings in multiple reads (depending on the coverage depth and the length of the signature). We leverage on this concept to genotype known variants directly from the input reads.

MALVA takes as input a reference genome, a VCF file representing all its known variants, and a read sample; it outputs a VCF file containing the most probable genotype for each variant. Our method is composed of four steps.

In the first step, MALVA computes the set of signatures of length k_s of all the alternate alleles of all the variants in VCF and stores them in the set ALTSIG. In the same step, the signatures of the reference alleles are computed and stored in a second set named REFSIG. For each k_s-mer t of a signature s two weights, one representing the number of occurrences of t in an alternate allele signature and one representing the number of occurrences of t in a reference allele signature, are stored. We will refer to these two values as $w_t^{\mathtt{A}}$ and $w_t^{\mathtt{R}}$, respectively.

We note that for small values of k_s the probability that the k_s-mers that constitute a signature appear in other regions of the genome is high. Since in the following steps MALVA exploits the signatures' sets of the alleles of each variant to call the genotypes, the presence of conserved regions of the reference genome identical to some signature could lead the tool to erroneously genotype some variants.

To get rid of a large amount of wrong calls, in the second step MALVA makes use of the context around the allele to distinguish its signatures from such regions. More precisely, if a k_s-mer of a signature of an alternate allele appears somewhere in the reference genome, MALVA extracts the context of length k_c (with $k_c > k_s$) covering the reference genome region and collects such k_c-mers in a third set (REPCTX).

In the third step, MALVA extracts all the k_c-mers from the sample along with the number of its occurrences. For each k_c-mer t_c that occurs w times in the sample, the k_s-mer t_s that constitutes the center of t_c is extracted. If t_s is found in REFSIG, $w_{t_s}^R$ is increased by w. Moreover, if t_c is not found in REPCTX and if t_s is in ALTSIG, $w_{t_s}^A$ is increased by w. Otherwise, if t_c is in REPCTX, $w_{t_s}^A$ is not updated since, although its central k_s-mer is identical to some k_s-mer of a signature of an alternate allele of some variant, it is indistinguishable from another region of the genome not covering the variant.

We note that when $w_{t_s}^A$ is not updated, our method might miss a variant in the donor and report a false negative, although for large values of k_c this would rarely occur. The rationale behind this choice is to avoid biases due to k_c-mers in

conserved regions of the reference genome, preferring not to include an alternate allele in the output whenever ambiguities arise.

Finally, in the fourth step, MALVA uses the weights computed in the previous step to call the genotypes.

In the rest of this section we will detail each one of the four steps of MALVA.

Signature computation The first step of MALVA consists in building the signatures of the alleles of all the variants and adding them either to ALTSIG, if they are the signatures of an alternate allele, or to REFSIG, if they are the signature of the reference allele. If a variant v is k_s-isolated, we build $1 + |\texttt{ALT}(v)|$ signatures, one for each allele of v. Otherwise, there are some genomes in $\mathcal{G}$ in which there is at least another allele of a variant that lays within a radius of $\lfloor k_s/2 \rfloor$ nucleotides from the center of the allele of v. In practice, this means that we have to look at the genotype data of the variants within such radius: for each allele a of v we reconstruct the k_s bases long portions of the genomes in $\mathcal{G}^a$ that constitute the signatures of a.

As pointed out in Definition 2, if $|\texttt{SEQ}(a)| \geq k_s$, the signature of a is the set of k_s-mers that appear in $\texttt{SEQ}(a)$. In this case we extract all such k_s-mers and add them either to REFSIG or ALTSIG. Otherwise, if $|a| < k_s$, we build the k_s bases long substrings of each genome in $\mathcal{G}^a$ centered in a by scanning the VCF file and reconstructing the sequences according to the genotype information it includes.

More precisely, let a be an allele of a variant v and let $V = \{v_1, \ldots, v_n\}$ be the set of variants such that, for all $1 \leq i \leq n$: (i) $v_i \neq v$, (ii) there exists an allele a_j in $\texttt{ALLELES}(v_i)$ such that a and a_j are both included in some genome g, and (iii) either ($\texttt{END}(a_j) < \texttt{BEGIN}(a)$ and $\texttt{CENTER}(a) - \lfloor k_s/2 \rfloor \leq \texttt{END}(a_j)$) or ($\texttt{END}(a) < \texttt{BEGIN}(a_j)$ and $\texttt{CENTER}(a) + \lfloor k_s/2 \rfloor \geq \texttt{BEGIN}(a_j)$) in g.

Given allele a, we use the genotype information stored in the VCF file to retrieve the haplotypes in which it is included, *i.e.* a subset of the haplotypes in $\mathcal{G}^a$, and build the set V. Using V we gather all the alleles that precede and succeed a in the selected haplotypes and we use them, together with the reference sequence, to reconstruct *on the fly* the k_s-mer that covers a, by interposing reference substrings and allele sequences. Doing so, we don't need to reconstruct

the whole haplotypes but we only analyze and reconstruct the required k_s-mers when needed.

Once all the k_s-mers have been constructed, they are added to `REFSIG` if a is the reference allele, to `ALTSIG` if it is an alternate allele.

Detection of repeated signatures This step is aimed to detect and store in set `REPCTX` all the k_c-mers of the reference sequence whose central k_s-mer is included in some signature of some alternate allele, $k_c > k_s$. `REPCTX` will be used in a further step to discard alternate alleles that might be erroneously reported as expressed by `MALVA` only because they cannot be told apart from other identical regions of the reference sequence.

To compute `REPCTX`, we extract all the k_c-mers of the reference sequence and test whether their central k_s-mer is in `ALTSIG`. If so, we add the k_c-mer to `REPCTX` to report that the k_s-mer is indistinguishable from some k_s-mer that is included in the signature of an alternate allele.

We will now exemplify the first two steps of our approach. Figure 4.2 shows an example composed of three variants and two reads. In this example the values of k_s and k_c are set to 7 and 11, respectively. Subfigure (a) shows the 26-bases long reference sequence. Subfigure (b) reports on the left two bi-allelic variants (v_1 and v_2) and one multi-allelic variant (v_3), and on the right the signatures of each allele of v_2. Subfigure (c) shows the elements of `ALTSIG` and `REFSIG` related to v_2. We note that the second signature in `ALTSIG` is composed of a single k_s-mer (t_s, equal to `TCCGGCG`) that appears in the reference genome, starting from position 17. Thus, the k_c-mer starting in position 15 and ending in position 25 (t_c, equal to `GATCCGGCGAA`) is added to `REPCTX`. Subfigure (d) shows two 11-bases long reads including t_s, extracted from position 3 and 15 of the donor. Clearly, only r_1 should contribute to the detection of the alternate allele of v_2 in the donor, since r_2 was sequenced from another position of the genome (*i.e.* $w^A_{t_s}$ should be equal to 1 in this case). To this aim, `REPCTX` comes to an aid; indeed, when analyzing r_1 the k_c-mer covering t_s is extracted (*i.e.* the whole read) and its inclusion in `REPCTX` is tested. Since `TATCCGGCGTA` is not in `REPCTX` and t_s is in `ALTSIG`, $w^A_{t_s}$ is increased by one. On the other hand, since `GATCCGGCGAA` is in `REPCTX`, the occurrence of t_s in r_2 is not considered in $w^A_{t_s}$, thus avoiding to

(a)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
$\mathcal{R}$	A	T	A	T	C	C	T	G	C	G	T	A	G	A	G	A	**T**	**C**	**C**	**G**	**G**	**C**	**G**	A	A	G

(b)

	Pos	Ref	Alts	Donors		
v_1	5	C	AAA	0|1	0|0	1|1
v_2	7	T	G	0|1	0|1	0|1
v_3	10	G	A,C	0|0	1|0	2|0

Variant	Allele	Signatures
v_2	a_0 = T	{ {TCC**T**GCG}, {TCC**T**GCA} , {AAC**T**GCC} }
	a_1 = G	{ {AAC**G**GCG}, {TCC**G**GCG}}

(c)

ALTSIG
{AAC**G**GCG}
{TCC**G**GCG}

REFSIG
{TCC**T**GCG}
{TCC**T**GCA}
{AAC**T**GCC}

REPCTX
GATCCGGCGAA

(d)

	1	2	3	4	5	6	7	8	9	10	11
r_1	T	A	T	C	C	G	G	C	G	T	A
r_2	G	A	T	C	C	G	G	C	G	A	A

Figure 4.2: Example with 3 variants and two reads. Subfigure (a) shows a reference genome of 26 bases, Subfigure (b) reports 3 variants and the signatures of each allele of variant v_2, Subfigure (c) reports the subsets of `ALTSIG`, `REFSIG`, and `REPCTX` including the elements related to v_2, and Subfigure (d) presents two reads of length 11.

erroneously overestimate the frequency of allele a_1 of v_2.

We note that on one hand this approach allows us to avoid overestimating the frequencies of some alternate allele but, on the other hand, it produces two major side effects. The first one is that some allele might be underestimated by `MALVA`; indeed, if the k_c-mer covering an alternate allele in a donor is equal to a k_c-mer in the genome it will not be detected. The second side effect is that `MALVA` might overestimate the frequency of some allele due to identical signature. Indeed, suppose that the signature of some alternate allele a_i of another variant $v_j \neq v_2$ is equal to the signature of alternate allele a_1 of variant v_2. It is obvious that the weights of the k_s-mers of the two signatures will be identical and that the occurrences of both the alleles will concur towards their final value, overestimating it.

Although the two side effects pose some limit to our method, they arise

rarely and we think they are a fair price to pay to avoid biases introduced by the reference genome.

Alleles' signatures weights computation In the third step, `MALVA` computes how many times the k_s-mers of each signature appear in the dataset. First, `MALVA` extracts all the k_c-mers of the read sample and tests their existence in `REPCTX` to check whether their central k_s-mer cannot be told apart from some repetition in the reference genome. Then, given a k_c-mer t_c that occurs w times in the read sample, the k_s-mer t_s that constitutes its center is extracted. If t_s is found in `REFSIG`, *i.e.* t_s is the signature of the reference allele of some variant, the weight $w^R_{t_s}$ is increased by w. Moreover, if t_c is not found in `REPCTX` and t_s is in `ALTSIG`, *i.e.* k_s-mer t_s is uniquely associated to an alternate allele of some variant, the weight $w^A_{t_s}$ is increased by w. Conversely, if t_c is in `REPCTX`, $w^A_{t_s}$ is not updated. The last scenario happens when t_s is identical to the signature of an alternate allele of some variant (indeed, t_s is in `ALTSIG`), but even the enlarged context t_c (and consequently t_s) appears somewhere else in the reference genome.

Genotype calling In the last step, `MALVA` uses the allele frequencies stored in the `INFO` field of the VCF file and the weights of the signatures computed in the previous step to call the genotype of each variant. To this aim, we extend the approaches proposed in the literature for bi-allelic variants (specifically, the one introduced in `LAVA` [139]) to multi-allelic variants. While the approaches designed for genotyping bi-allelic variants only need to compute the likelihood of three genotypes, our technique must consider a larger number of possible genotypes.

Let v be a variant with $n-1$ alternate alleles. The number of possible distinct genotypes is $\binom{n}{2} + n = \frac{n(n+1)}{2}$, that is: one *homozygous reference* genotype, $\binom{n}{2}$ *heterozygous* genotypes, and $n-1$ *homozygous alternate* genotypes. We will refer to the homozygous reference genotype as $G_{0,0}$, to the heterozygous genotypes as $G_{i,j}$ with $0 \leq i < j \leq n-1$, and to the homozygous alternate genotypes as $G_{i,i}$ with $1 \leq i \leq n-1$. Following well-established techniques [139, 101, 86], we compute the likelihood of each genotype $G_{i,j}$ by means of the Bayes' theorem.

Given the observed coverage C, we compute the posterior probability of each genotype as:

$$P(G_{i,j}|C) = \frac{P(G_{i,j})P(C|G_{i,j})}{P(C)}$$

that, by the law of total probability, can be expressed as:

$$P(G_{i,j}|C) = \frac{P(G_{i,j})P(C|G_{i,j})}{\sum_{p=0}^{n-1}\sum_{q=p}^{n-1} P(G_{p,q})P(C|G_{p,q})}$$

To calculate this probability, we compute the *a priori* probabilities of each genotype $G_{i,j}$ ($P(G_{i,j})$) and the *conditional probability* of the observed coverage given the considered genotype ($P(C|G_{i,j})$).

The Hardy-Weinberg equilibrium equation ensures that for each variant v, $(\sum_{i=0}^{n-1} f_i)^2 = 1$, where $f_i = \texttt{FREQ}(v)[i]$, *i.e.* the frequency of the i-th allele of v. We recall that $\texttt{FREQ}(v)$ is stored in the `INFO` field of the VCF file. The *a priori* probability of each genotype $G_{i,j}$ is therefore computed as follows:

$$P(G_{i,j}) = \begin{cases} f_i^2 & \text{if } i = j \\ 2f_i f_j & \text{otherwise} \end{cases}$$

To compute the conditional probability $P(C|G_{i,j})$, it is first necessary to compute the *coverages* of the alleles of the variant. Without loss of generality, let a_0 be the first allele of the variant, *i.e.* a_0 is the reference allele with index 0. We recall that $\texttt{SIGN}(a_0)$ is the set of signatures of allele a_0 and that each signature is a set of one or more k-mers. We also recall that, in the previous step, for each k-mer t that belongs to some signature we computed two weights, namely w_t^R and w_t^A.

Given a signature $s \in \texttt{SIGN}(a_0)$, we define its weight as the mean of the weights associated to the k-mers it contains, *i.e.* $\frac{\sum_{t\in s} w_t^R}{|s|}$ where $|s|$ denotes the number of k-mers contained in signature s. Since the same allele may exhibit more signatures, we define the coverage c_0 of allele a_0 as the maximum value among the weights of its signatures, *i.e.* $\max\{\frac{\sum_{t\in s} w_t^R}{|s|} : s \in \texttt{SIGN}(a_0)\}$. This formula can be easily modified to compute the coverage of an alternate allele (c_i for $i \geq 1$) by switching w_t^R with w_t^A. The coverage c_i of an allele a_i of a variant

is thus computed as follows:

$$c_i = \begin{cases} \max\{\frac{\sum_{t \in s} w_t^R}{|s|} : s \in \texttt{SIGN}(a_0)\} & \text{if } i = 0 \\ \max\{\frac{\sum_{t \in s} w_t^A}{|s|} : s \in \texttt{SIGN}(a_i)\} & \text{otherwise} \end{cases}$$

By extending the approach adopted in [139], we consider each $P(C|G_{i,j})$ to be multinomially distributed. Given a homozygous genotype $G_{i,i}$, we assume to observe the i-th allele, which is the correct one, with probability $1 - \varepsilon$ (where ε is the expected error rate) whereas the other $n - 1$ alleles (the erroneous ones) with probability $\frac{\varepsilon}{n-1}$ each. Hence, we compute the conditional probability of a homozygous genotype as:

$$P(C|G_{i,i}) = \binom{c_i + C_E}{c_i} (1 - \varepsilon)^{c_i} \left(\frac{\varepsilon}{n-1}\right)^{C_E}$$

where C_E is the total sum of the coverages of the erroneous alleles, *i.e.* $C_E = \sum_{j \in \{0,\ldots,n-1\} \setminus \{i\}} c_j$.

For what concerns heterozygous genotypes, we assume to observe the correct alleles, *i.e.* the i-th and the j-th allele, with equal probability $\frac{1-\varepsilon}{2}$ whereas the other $n - 2$ erroneous alleles with probability $\frac{\varepsilon}{n-2}$ each. We compute the conditional probability of a heterozygous genotype as follows:

$$P(C|G_{i,j}) = \binom{c_i + c_j + C_E}{c_i + c_j} \binom{c_i + c_j}{c_i} \left(\frac{1-\varepsilon}{2}\right)^{c_i} \left(\frac{1-\varepsilon}{2}\right)^{c_j} \left(\frac{\varepsilon}{n-2}\right)^{C_E}$$

where, again, C_E is the sum of the coverages of the erroneous alleles, *i.e.* $C_E = \sum_{p \in \{0,\ldots,n-1\} \setminus \{i,j\}} c_p$.

Finally, after computing the posterior probability of each genotype, `MALVA` outputs the genotype with the highest likelihood.

4.4 Results

We implemented `MALVA` in `C++` and we performed an experimental analysis on real data to evaluate the real feasibility of our method, comparing it with other approaches available in the literature. `MALVA` is freely available at `https://github.com/AlgoLab/malva`. Information and instruction on how to replicate the performed experiments are available at `https://github.com/AlgoLab/malva_experiments`.

This section is organized as follows. We will first describe the implementation details of MALVA and then we will provide a thorough description of the performed experiments and the obtained results. All the analyses were performed on a 64 bit Linux (Kernel 4.4.0) system equipped with four 8-core Intel ® Xeon 2.30GHz processors and 256GB of RAM.

4.4.1 Implementation details

Bloom filters were implemented as the union of a bit vector and a single hash function H. Although it is not conventional, in most cases to use a single hash function has similar results as using multiple ones, as noticed by other authors [151, 152]. To check this claim, while developing the tool we tested whether using multiple hash functions would improve the results by extending the Bloom filters to count-min sketches [29]. As expected, the deterioration of the performance far outweighed the gain in precision and recall (that was less than 0.1%).

Moreover, to use a single hash function allows us to store w_t^R and w_t^A efficiently for each k-mer t. Indeed, note that once all the signatures of all the alternate alleles have been added to ALTSIG, the latter is only used to check whether some k_s-mer is part of a signature, *i.e.* it becomes static. By representing ALTSIG as a Bloom filter $B_{\texttt{ALTSIG}}$ we can create an integer array CNTS of size $\texttt{rank}_1(|B_{\texttt{ALTSIG}}|+1, B_{\texttt{ALTSIG}})$ to store the weights of each k-mer compactly and, if a k-mer t of a signature s is in ALTSIG (*i.e.* if $B_{\texttt{ALTSIG}}[\texttt{H}(s)] = 1$) we can access its weight by accessing $\texttt{CNTS}[\texttt{rank}_1(\texttt{H}(s), B_{\texttt{ALTSIG}})]$.

In a nutshell, after adding all the alternate alleles to $B_{\texttt{ALTSIG}}$, we *freeze* it, build a rank data structure over it, compute the number of ones, and create the CNTS array of the correct size. We pose an upper limit of 255 to the value of each cell of the CNTS array, so as to store each counter using only 8 bits. Similarly, we implemented REPCTX as a Bloom filter $B_{\texttt{REPCTX}}$ using a single hash function.

Conversely, REFSIG was implemented as a simple hash table, because the number of elements it stores is usually smaller than the number of elements stored in ALTSIG.

Finally, instead of scanning all the k_c-mers in the read sample, we used

KMC3 [39] to efficiently extract them and counting their occurrences. Therefore, in step 3 MALVA parses the output of KMC3 and updates the counts for each k_s-mer accordingly.

For completeness, we note that the bit vectors, the rank data structure, and the CNTS array were implemented using the sdsl-lite library [48].

4.4.2 Experimental analysis

We performed an experimental analysis on real data to evaluate the real feasibility of our method and we compared MALVA to one alignment-free method, to one assembly-based approach, and to two different alignment-based pipelines. Among the alignment-free methods proposed in literature we chose VarGeno, as it is an improved version of LAVA that provides better efficiency and accuracy [152]. For completeness, we included in our evaluation discoSnp++ (assembly-based) and the two most widely used alignment-based pipelines, denoted by BCFtools and GATK, respectively. The pipeline denoted by BCFtools consists of an alignment step performed with BWA-MEM followed by a variant discovering step performed using BCFtools. The latter consists of an alignment step performed with BWA-MEM and a variant discovering step performed with GATK, as recommended by the *GATK Best Practices* [35].

MALVA was run setting k_s equal to 47, k_c equal to 53, ε equal to 0.1%, Bloom filters size equal to $8GB$, and considering the genotype data and the a priori frequencies of the alleles of the EUR population, since the individual under analysis is part of it.

We tested the tools using the Illumina WGS dataset of the well-studied NA12878 individual provided by the Genome In A Bottle (GIAB) consortium [177]. We chose this individual because the variant calls provided are highly reliable and can be effectively used to assess the precision and the recall of the considered methods. We downloaded the alignments of its 30x downsampled version and used SAMtools [89] to extract the corresponding FASTQ file, obtaining 696,168,435 150bp-long reads. As reference genome and set of known variants, we used the GRCh37 primary assembly and the VCF files provided by Phase 3 of the 1000 Genomes Project [150]. These VCF files contain a total of 84,739,838

variants, the phased genotype information of 2504 individuals, and the a priori frequency of each allele of each variant of 5 populations. As stated above, from this VCF, in our evaluation MALVA extracted and considered only the individuals from the EUR population, for a total of 502 individuals. We note that we also removed the NA12878 individual from the input to better analyze MALVA and its capability in genotyping an unknown individual.

We note that VarGeno requires a different formatting of the fields describing the a priori frequencies of the alleles than the ones in the VCF file provided by the 1000 Genomes Project. Thus, we formatted the input files as required before running VarGeno. However, VarGeno could not complete the analysis of this dataset, from now on denoted as FullGenome, on our server. To test whether VarGeno crashed due to excessive memory usage, we tried to run it on the same instance on a cluster with 1TB of RAM, but nevertheless it could not complete the analysis, crashing after 20 minutes.

In order to include VarGeno in our evaluation, we chose 12 chromosomes to create a smaller dataset, denoted by HalfGenome, that thus contains some half of the variants and the reads of the FullGenome dataset.

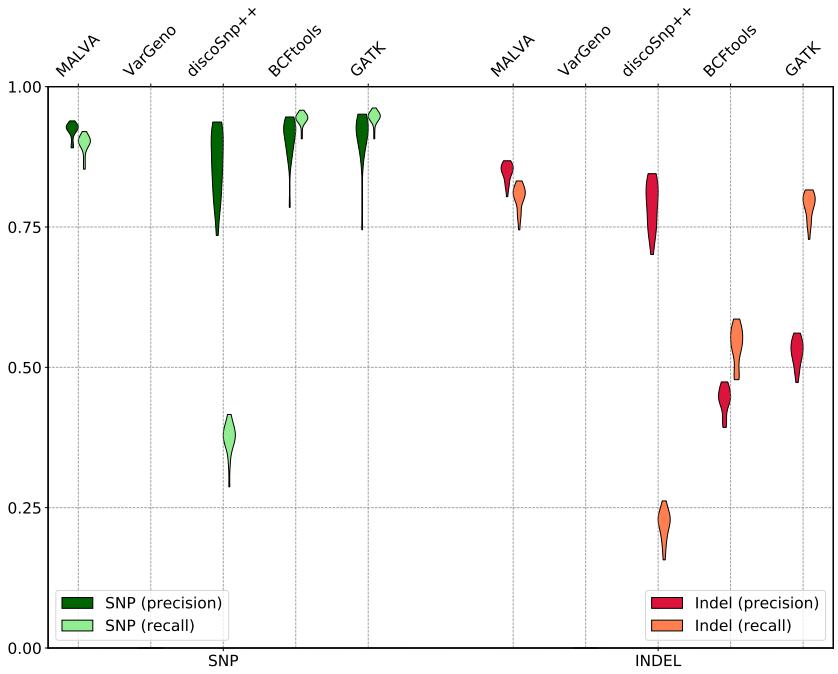
Each tool was evaluated in terms of variant calling accuracy and efficiency (wall time and memory usage). We note that some steps of the previously cited tools can use multiple threads to improve the time performance (namely, KMC3 for MALVA, discoSnp++, BWA-MEM for BCFtools and GATK, and the variant discovery steps of GATK). Whenever we had this choice, we provided 4 threads to each tool.

We used hap.py [82], the tool developed for the evaluation of variant callers in the recent *PrecisionFDA Truth Challenge*[1], and the /usr/bin/time system tool to gather the required data.

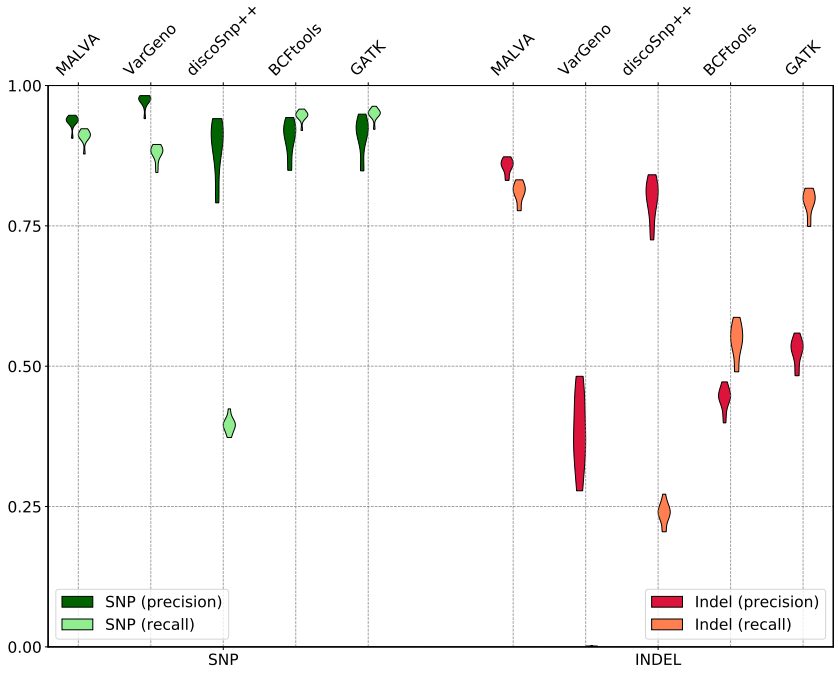
Table 4.1 shows the results obtained by the considered tools on both the FullGenome and the HalfGenome datasets. We point out that hap.py computes precision and recall considering only non-reference VCF records (*i.e.* non 0/0 calls). A qualitative representation of these results is available in Figure 4.3.

As expected, MALVA, VarGeno, and discoSnp++ are faster than the tested alignment-based approaches, *i.e.* BCFtools and GATK. Indeed, MALVA, VarGeno, and discoSnp++ required 4.5, 2.5, and 7.5 hours to analyze the HalfGenome

[1] https://precision.fda.gov/challenges/truth



(a) `FullGenome` dataset



(b) `HalfGenome` dataset

Figure 4.3: Qualitative representation of the accuracy results. Each violin plot represents the precision and the recall (computed with `hap.py`) achieved by the considered tools on both SNPs and indels. This is a qualitative representation of the information presented in Table 4.1.

Dataset	Tool	Prec *(SNPs)*	Rec *(SNPs)*	Prec *(Indels)*	Rec *(Indels)*	Time *(hh:mm)*	RAM *(GB)*
HalfGenome	MALVA	93.8%	91.1%	86.0%	81.4%	04:33	30
	VarGeno	97.5%	88.1%	39.5%	0.1%	02:31	52
	discoSnp++	89.5%	39.3%	80.8%	24.2%	07:45	7
	BCFtools	91.2%	94.8%	44.9%	55.4%	24:35	6
	GATK	91.7%	95.1%	53.2%	79.9%	34:43	32
FullGenome	MALVA	92.5%	90.0%	85.0%	80.6%	09:18	39
	VarGeno	-	-	-	-	-	-
	discoSnp++	86.9%	37.7%	80.0%	22.6%	14:19	9
	BCFtools	91.6%	94.4%	44.7%	54.6%	54:23	9
	GATK	92.1%	94.7%	53.2%	79.2%	73:36	33

Table 4.1: Accuracy and efficiency results on the HalfGenome and FullGenome datasets. For each dataset, we reported the values of Precision (Prec) and Recall (Rec) obtained by the considered tools on both SNPs and indels. The efficiency results are shown in terms of wall clock time and peak memory usage. VarGeno could not complete the analysis of the FullGenome dataset, thus we did not report its results on this dataset.

dataset, respectively, while BCFtools and GATK required 24.5 and 34.5 hours. We note that half of the time required by BCFtools and one third of the time required by GATK was spent running BWA-MEM, that completed its task in 12.5 hours (using 4 threads). The same trend applies to the analysis of the FullGenome dataset, on which each tool required roughly twice the time required on the HalfGenome dataset. A qualitative representation of the running time and the memory usage of each tool is shown in Figure 4.4.

For what concerns the memory usage, BCFtools proved to be the least memory intensive approach, requiring less than 10GB of RAM on both datasets to map the reads and less than 1GB of RAM to call the variants. MALVA and GATK showed similar memory requirements, with GATK showing almost no difference between the two analyses and MALVA increasing the memory consumption by

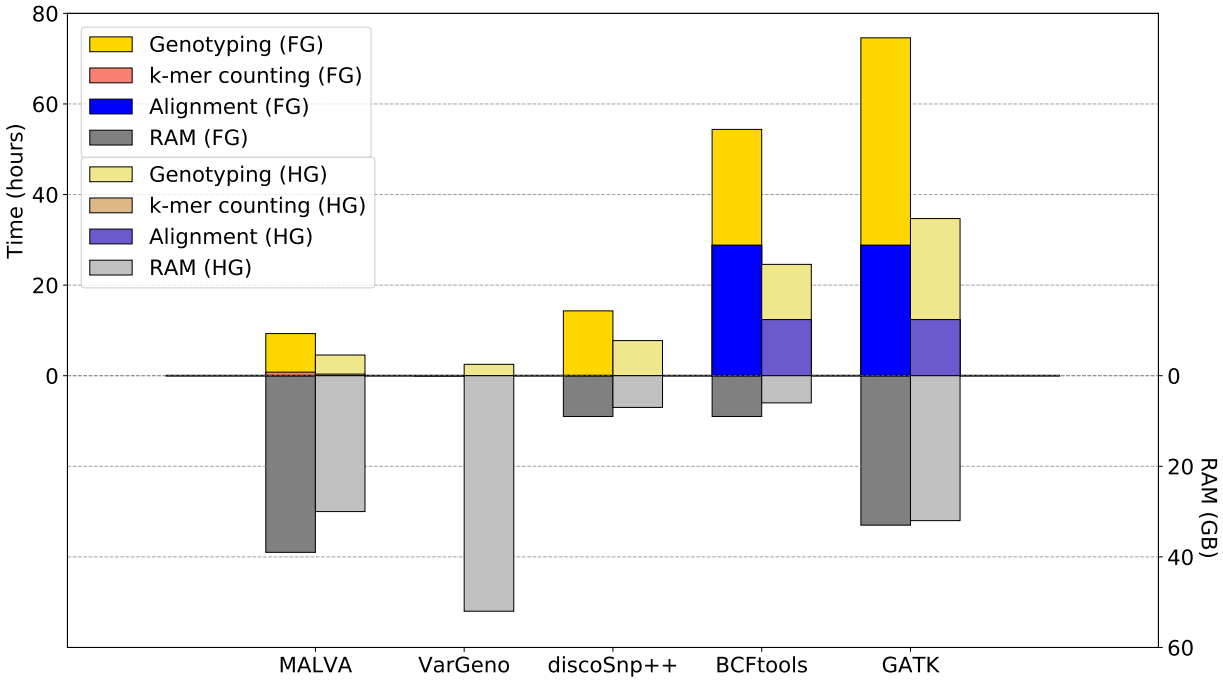


Figure 4.4: Time and RAM required by each tool to analyze both datasets. The running times are partitioned by steps performed whereas the RAM usage represents the peak memory of the entire process. For ease of presentation, we denoted the `FullGenome` dataset as `FG` whereas the `HalfGenome` dataset as `HG`. Note that we did not include `VarGeno` running time and RAM usage on the `FullGenome` dataset since it crashed after 20 minutes.

only 23% for the bigger dataset. VarGeno required slightly less than twice the amount of memory required by MALVA on the HalfGenome dataset.

Precision and recall of all the tools varied little over the two datasets, proving that the number of variants and reads only slightly affects their accuracy.

As expected, BCFtools and GATK achieved the best recall for non-homozygous reference SNPs due to the mapping step, that provides a more precise coverage of the alleles and allows to better discern repeated regions of the reference genome.

discoSnp++ achieved the lowest recall whereas VarGeno obtained 3% less recall than MALVA, that in turn called correctly 91.1% of the SNPs. On the other hand, MALVA, discoSnp++, BCFtools, and GATK achieved comparable precision on SNPs, whereas VarGeno obtained the highest one.

Overall, on non-homozygous reference SNPs, VarGeno seems to be the most conservative tool among those tested, as it prefers not to call SNPs when there is any uncertainty. On the contrary, MALVA deliberately prefers to detect any potential alternate allele in the donor, at the cost of a slight loss in precision.

Remarkably, on indels MALVA obtained significantly better recall than BCFtools and discoSnp++ and better precision than any other tool. As expected, since the method of VarGeno is not designed to manage indels, it was only able to genotype a negligible percentage of them. On the other hand, discoSnp++ achieved a high precision but it was only able to call less than a quarter of the total indels. Finally, BCFtools showed a very low precision and recall on indels, whereas GATK achieved a recall similar to MALVA but a low precision. The low precision achieved by the alignment-based tools is mainly due to the difficulties in aligning reads that overlap with indels.

We also performed a more detailed analysis on the influence of indel size on the recall obtained by the tools on the FullGenome dataset. As shown in Figure 4.5, MALVA proved to be the only tool able to call long indels (more than $\sim 40/50$ bases) whereas the other tools are limited to short indels (that are also the most common ones). In any case, MALVA outperformed the other tools even on these shorter indels. We did not include VarGeno in this analysis since its recall on indels was lower than 1% even on the HalfGenome dataset.

Finally, we also performed a more in-depth comparison of MALVA and VarGeno, the two alignment-free approaches considered in our evaluation, to assess whether

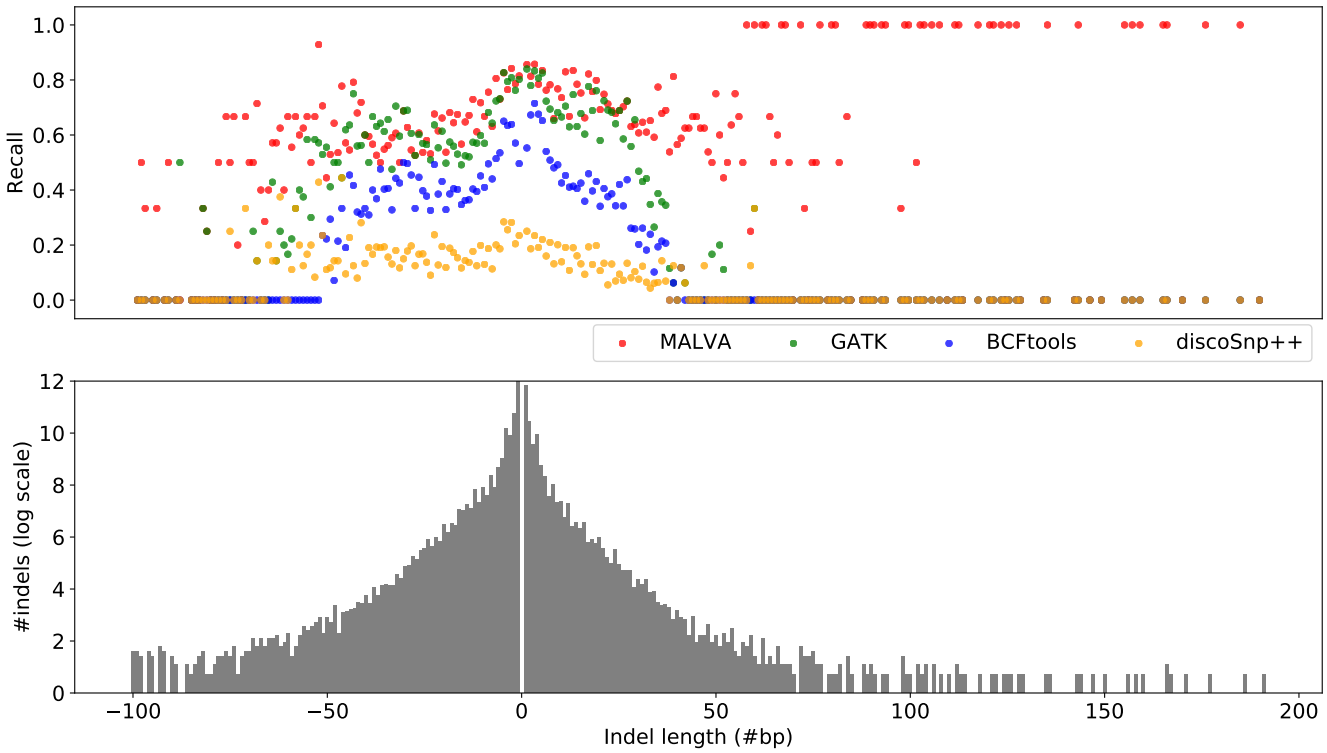


Figure 4.5: Influence of indel size on the recall achieved by the four considered tools on the `FullGenome` dataset. The histogram shows the frequency distribution (on logarithmic scale) of the indels with respect to their length. The scatter plot shows the recall of the tools with respect to the indel size.

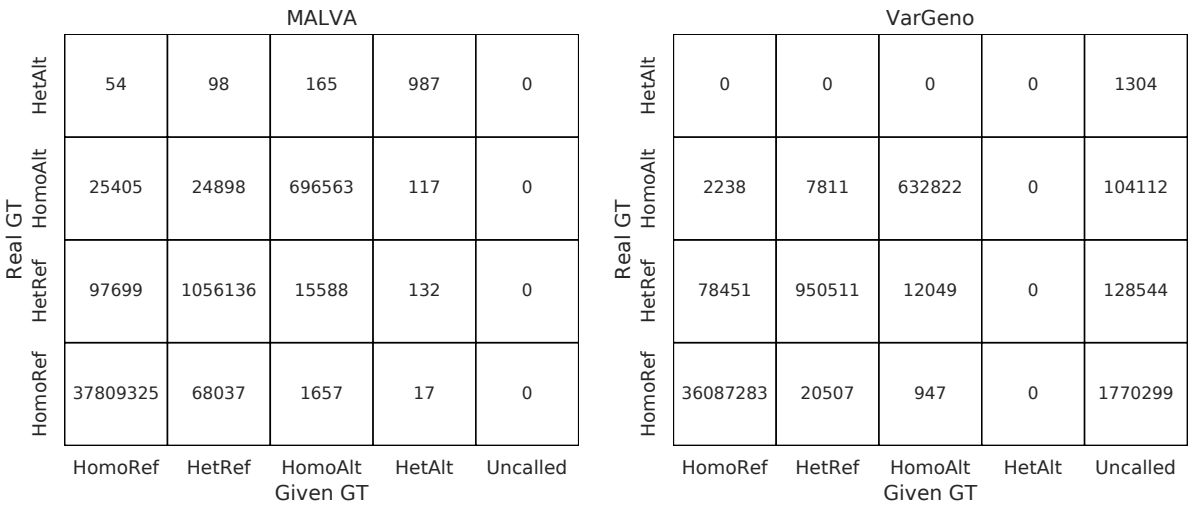


Figure 4.6: **Comparison between real genotype (provided by the 1000 Genomes Project) and genotype called by `MALVA` and `VarGeno`.** HomoRef stands for Homozygous Reference, HetRef stands for Heterozygous Reference, HomoAlt stands for Homozygous Alternate, HetAlt stands for Heterozygous Alternate, and Uncalled means that the given variant was not called by the tool.

they produce some systematic error. With this aim, we considered the `HalfGenome` dataset and we analyzed the SNPs genotyped by the two tools.

For each tool, Figure 4.6 reports the number of correct genotypes output, grouping them in *homozygous reference* (*i.e.* 0|0), *heterozygous reference* (*i.e.* 0|1, 0|2, and so on), *homozygous alternate* (*i.e.* 1|1, 2|2, and so on), and *heterozygous alternate* (*i.e.* 1|2, 1|3, 2|3, and so on). Figure 4.7 shows the same data row-normalized.

We recall that the precision and recall output by `hap.py` do not consider homozygous reference genotypes, thus this analysis allows us to better understand the behavior of the tools. Since `VarGeno` is not able to manage indels, we decided to not include them in this analysis.

Consistently with the precision and recall computed by `hap.py`, `MALVA` detects between 5% and 10% more correct variants than `VarGeno` in all classes, at the cost of producing more erroneous calls. We note that overall `VarGeno` filters out 2,004,259 of the 39,796,878 SNPs in the truth.

Both tool show similar pattern in erroneous calls. More precisely erroneously

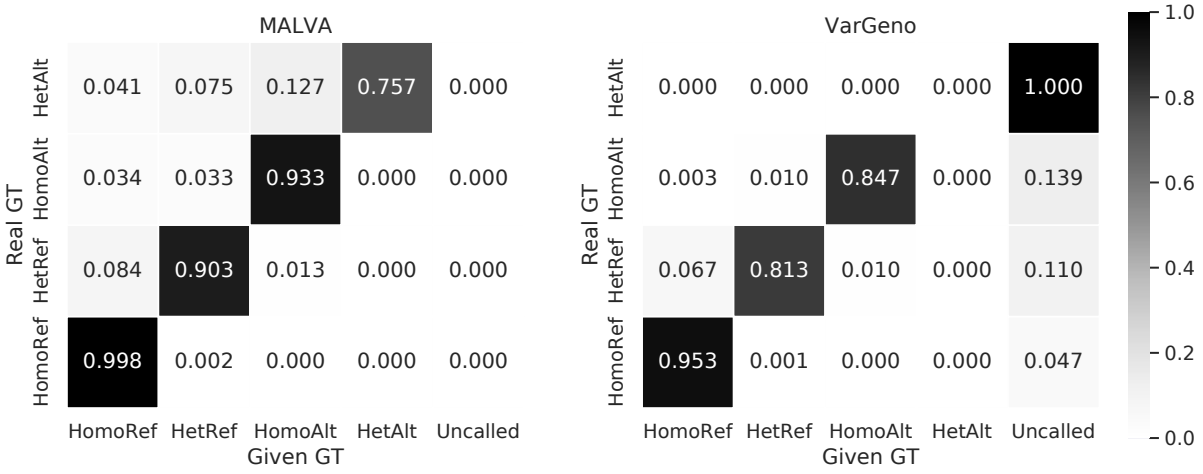


Figure 4.7: **Comparison between real genotype (provided by the 1000 Genomes Project) and genotype called by `MALVA` and `VarGeno`, normalized by rows.** HomoRef stands for Homozygous Reference, HetRef stands for Heterozygous Reference, HomoAlt stands for Homozygous Alternate, HetAlt stands for Heterozygous Alternate, and Uncalled means that the given variant was not called by the tool.

genotyped homozygous reference variants were mostly genotyped as heterozygous reference and, vice-versa, erroneously genotyped heterozygous reference variants were mostly genotyped as homozygous reference. On the other hand, erroneous homozygous alternate variants in the donor were mostly genotyped as heterozygous reference by `VarGeno` whereas `MALVA` evenly distributed the errors between homozygous reference and heterozygous reference calls. Finally, erroneous heterozygous alternate variants in the donor were mostly genotyped as homozygous alternate variants by `MALVA`, meaning that the method proposed in this paper was able to detect the fact that one of the alleles was not the reference allele.

Overall, the errors produced by both tools were "partial" errors in the sense that they rarely mis-call both alleles of the donor.

4.5 Final remarks and future directions

In this chapter we presented `MALVA`, the first efficient alignment-free genotyping tool that is able to handle multi-allelic variants and indels. We compared `MALVA` with `VarGeno`, the state-of-the-art alignment-free genotyping tool, showing that our method is less memory intensive, achieves better recall, handles dozens of millions of variants effectively, and is also able to genotype indels and multi-allelic variants. These fundamental features allow our method to exploit the whole information in input, without filtering out any data that might be crucial in downstream analyses. Most notably, `MALVA`'s ability to genotype indels allows to apply alignment-free techniques to many clinical contexts, including screens for genetic predispositions for disease linked to the presence of indels [128, 171].

We also compared our tool with two variant discovery pipelines, namely `GATK` and `BCFtools`, showing that `MALVA` is an order of magnitude faster while achieving better accuracy on indels and similar accuracy on SNPs.

Overall, `MALVA` proved to be an accurate and efficient alternative to alignment-based pipelines for variant calling and the experimental evaluation we performed showed the usefulness of the formalization of signature of an allele. However, although using the concept of signature proved to be effective in this context, in some edge cases two alleles might share the same signature and our method

will not be able to discern between the two. A simple solution to reduce the occurrences of different alleles sharing the same signature is to increase the value of k. Unfortunately, increasing k beyond 40–50 has two main drawbacks: (i) it is computationally expensive and (ii) due to errors, the probability that such k-mers appear in the input reads decreases. To face this limitation, future works should investigate the effect of using multiple k-mers spanning each allele and exploit the k-mers flanking the potential occurrence of an allele in the read.

Future steps will be also devoted to improving the efficiency of `MALVA` by exploiting the parallel architecture of modern machines, and to extending the method to discovery and genotype *de novo* variants, *i.e.* variants that exist in a child but do not exist in both its parents, as done by tools like `COBASI` [49] and `kevlar` [149].

Finally, another possible future direction consists in designing an alignment-free method for genotyping known variants using long reads such those produced by the latest PacBio and Oxford Nanopore technologies. Indeed, `MALVA` is specifically designed for dealing with Illumina short-read data and cannot be directly applied to long-read data due to their higher error rate.

Chapter 5

Conclusions

The analysis of NGS data allows to improve the quality of our lives by enhancing our understanding of genetic diversity and genetic diseases. In this thesis we presented two novel algorithmic approaches for the analysis of NGS data. Our goal was to propose good alternatives to current state-of-the-art approaches and, in the meantime, solve open computational problems.

The first approach we presented is `ASGAL`, a tool that identifies the alternative splicing events supported by an RNA-Seq sample by performing an accurate alignment to a splicing graph. Differently from current approaches that rely on transcript reconstruction or spliced alignment against a reference genome, we devised a novel algorithmic technique solving the open problem of approximately matching a string against a labeled graph via the use of succinct data structures. As proved by our experimental evaluation, aligning reads to a splicing graph is a valid alternative to spliced alignment to a reference genome and it allows to accurately detect alternative splicing events supported by an RNA-Seq sample. Indeed, by tailoring the alignment step to the identification of alternative splicing events, `ASGAL` is able to detect events that are novel with respect to a gene annotation even when a single transcript per gene is supported by the RNA-Seq sample.

The second approach we presented is `MALVA`, a tool for genotyping known Single Nucleotide Polymorphisms and indels from a NGS sample. The most used approaches for variant genotyping rely on read alignment. Conversely, we proposed an alignment-free approach that is able to genotype a set of known

variant directly from the raw reads contained in a NGS sample. The open problem we aimed to solve was the design of an alignment-free algorithmic technique able to genotype indels and multi-allelic variants. Indeed all the alignment-free approaches proposed in the literature can manage only single-allelic Single Nucleotide Polymorphisms. Thanks to its alignment-free approach, `MALVA` is really fast, requiring one order of magnitude less time than alignment-based pipelines, while achieving similar accuracy. Remarkably, `MALVA` achieves the best accuracy in genotyping indels.

An in-depth analysis of alternative splicing events and small genomic variations that may occur between different individuals is essential for a better understanding of genetic diversity and for discovering the causes behind diseases and tumors. For this reason, future developments will focus on the enhancement of the two tools presented in this thesis.

Moreover, since the two topics tackled in this thesis, *i.e.* alternative splicing and small genomic variants, are closely related, further steps will also focus on investigating if the proposed algorithmic approaches, or part of them, can be combined in a single tool. The main goal would be the development of a tool for the detection of alternative splicing events that is also able to impute them to one or more genomic variants in order to facilitate any downstream analysis.

Finally, future work will be also devoted to investigating the problem of detecting structural variations from NGS data in an alignment-free fashion. We are motivated by our interest in developing knowledge in this context and we believe that the alignment-free approach we presented may be a good starting point for tackling such a problem.

Bibliography

[1] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of discrete algorithms*, 2(1):53–86, 2004.

[2] Tatsuya Akutsu. A linear time pattern matching algorithm between a string and a tree. In *Combinatorial Pattern Matching*, pages 1–10. Springer, 1993.

[3] Cornelis A Albers, Gerton Lunter, Daniel G MacArthur, Gilean McVean, Willem H Ouwehand, and Richard Durbin. Dindel: accurate indel calls from short-read data. *Genome research*, 21(6):961–973, 2011.

[4] Fatemeh Almodaresi, Prashant Pandey, Michael Ferdman, Rob Johnson, and Rob Patro. An efficient, scalable and exact representation of high-dimensional color information enabled via de Bruijn graph search. In *International Conference on Research in Computational Molecular Biology*, pages 1–18. Springer, 2019.

[5] Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: a succinct colored de Bruijn graph representation. In *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[6] Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. *Journal of Algorithms*, 35(1):82–99, 2000.

[7] Debora Angeloni, Fuh-Mei Duh, Michele Moody, Michael Dean, Eugene R Zabarovsky, Vera Sentchenko, Eleonora Braga, and Michael I Lerman. C to a single nucleotide polymorphism in intron 18 of the human mst1r (ron) gene that maps at 3p21. 3. *Molecular and cellular probes*, 17(2-3):55–57, 2003.

[8] Ricardo A Baeza-Yates and Gaston H Gonnet. A new approach to text searching. In *ACM SIGIR Forum*, volume 23, pages 168–175. ACM, 1989.

[9] Edward V Ball, Peter D Stenson, Shaun S Abeysinghe, Michael Krawczak, David N Cooper, and Nadia A Chuzhanova. Microdeletions and microinsertions causing human genetic disease: common mechanisms of mutagenesis and the role of local dna sequence complexity. *Human mutation*, 26(3):205–213, 2005.

[10] Sam Behjati and Patrick S Tarpey. What is next generation sequencing? *Archives of Disease in Childhood-Education and Practice*, 98(6):236–238, 2013.

[11] Stefano Beretta, Paola Bonizzoni, Luca Denti, Marco Previtali, and Raffaella Rizzi. Mapping RNA-seq data to a transcript graph via approximate pattern matching to a hypertext. In *International Conference on Algorithms for Computational Biology*, pages 49–61. Springer, 2017.

[12] Andrew Best, Katherine James, Caroline Dalgliesh, Elaine Hong, Mahsa Kheirolahi-Kouhestani, Tomaz Curk, Yaobo Xu, Marina Danilenko, Rafiq Hussain, Bernard Keavney, et al. Human tra2 proteins jointly control a chek1 splicing switch among alternative and constitutive target exons. *Nature Communications*, 5:4760, 2014.

[13] Md Shariful Islam Bhuyan, Itsik Pe'er, and M Sohel Rahman. Sicario: Short indel call filtering with boosting. *bioRxiv*, page 601450, 2019.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[15] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of Bloom filters. *Information Processing Letters*, 108(4):210–213, 2008.

[16] Robert S Boyer and J Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[17] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *Research Report*, 124, 1994.

[18] William S Bush and Jason H Moore. Genome-wide association studies. *PLoS computational biology*, 8(12), 2012.

[19] Brian Bushnell. Bbmap: a fast, accurate, splice-aware aligner. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2014.

[20] William I. Chang and Eugene L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4-5):327–344, 1994.

[21] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8:22, 2013.

[22] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a bloom filter. *Information Processing Letters*, 110(21):944–949, 2010.

[23] Suzanne Clancy. Rna splicing: introns, exons and spliceosome. *Nature Education*, 1(31), 2008.

[24] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The sanger FASTQ file format for sequences with quality scores, and the solexa/illumina FASTQ variants. *Nucleic acids research*, 38(6):1767–1771, 2009.

[25] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[26] International HapMap Consortium et al. The international hapmap project. *Nature*, 426(6968):789, 2003.

[27] The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 10 2016.

[28] UK10K consortium et al. The uk10k project identifies rare variants in health and disease. *Nature*, 526(7571):82, 2015.

[29] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[30] Ying Cui, Meng Cai, and H Eugene Stanley. Comparative analysis and classification of cassette exons and constitutive exons. *BioMed research international*, 2017, 2017.

[31] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, 06 2011.

[32] Nicolaas G de Bruijn and Paul Erdös. On a combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, 51(10):1277–1279, 1948.

[33] Luca Denti, Marco Previtali, Giulia Bernardini, Alexander Schönhuth, and Paola Bonizzoni. MALVA: genotyping by Mapping-free ALlele detection of known VAriants. *iScience*, 18:20–27, 2019.

[34] Luca Denti, Raffaella Rizzi, Stefano Beretta, Gianluca Della Vedova, Marco Previtali, and Paola Bonizzoni. ASGAL: aligning RNA-Seq data to a splicing graph to detect novel alternative splicing events. *BMC Bioinformatics*, 19(1):444, 2018.

[35] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire, Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt Hanna, et al. A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nature Genetics*, 43(5):491–498, 2011.

[36] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 425–455. Springer, 2018.

[37] Balázs Dezső, Alpár Jüttner, and Péter Kovács. LEMON–an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[38] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[39] Maciej Długosz, Marek Kokot, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 05 2017.

[40] Hannes P Eggertsson, Hakon Jonsson, Snaedis Kristmundsdottir, Eirikur Hjartarson, Birte Kehr, Gisli Masson, Florian Zink, Kristjan E Hjorleifsson, Aslaug Jonasdottir, Adalbjorg Jonasdottir, et al. Graphtyper enables population-scale genotyping using pangenome graphs. *Nature Genetics*, 49(11):1654–1660, 2017.

[41] Pär G Engström, Tamara Steijger, Botond Sipos, Gregory R Grant, André Kahles, Tyler Alioto, Jonas Behr, Paul Bertone, Regina Bohnert, Davide Campagna, et al. Systematic evaluation of spliced alignment programs for RNA-seq data. *Nature Methods*, 10(12):1185, 2013.

[42] Francisco Fernandes and Ana T Freitas. slaMEM: efficient retrieval of maximal exact matches using a sampled LCP array. *Bioinformatics*, 30(4):464–471, 2013.

[43] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398. IEEE, 2000.

[44] Sylvain Foissac and Michael Sammeth. Astalavista: dynamic and flexible analysis of alternative splicing events in custom gene datasets. *Nucleic acids research*, 35(suppl_2):W297–W299, 2007.

[45] Laurent C Francioli, Androniki Menelaou, Sara L Pulit, Freerk Van Dijk, Pier Francesco Palamara, Clara C Elbers, Pieter BT Neerincx, Kai Ye, Victor Guryev, Wigard P Kloosterman, et al. Whole-genome sequence variation, population structure and demographic history of the dutch population. *Nature Genetics*, 46(8):818, 2014.

[46] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 2018.

[47] GFF/GTF File Format - Definition and supported options. `https://www.ensembl.org/info/website/upload/gff.html` [Online, accessed: 25 October, 2019].

[48] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[49] Laura Gómez-Romero, Kim Palacios-Flores, José Reyes, Delfino García, Margareta Boege, Guillermo Dávila, Margarita Flores, Michael C Schatz, and Rafael Palacios. Precise detection of de novo single nucleotide variants in human genomes. *Proceedings of the National Academy of Sciences*, 115(21):5516–5521, 2018.

[50] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333, 2016.

[51] Manfred Grabherr, Brian Haas, Moran Yassour, Joshua Levin, Dawn Thompson, Ido Amit, Xian Adiconis, Lin Fan, Raktima Raychowdhury, Qiandong Zeng, Zehua Chen, Evan Mauceli, Nir Hacohen, Andreas Gnirke, Nicholas Rhind, Federica di Palma, Bruce Birren, Chad Nusbaum, Kerstin Lindblad-Toh, Nir Friedman, and Aviv Regev. Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nature Biotechnology*, 29:644–652, 2011.

[52] Szymon Grabowski and Marcin Raniszewski. Rank and select: Another lesson learned. *Information Systems*, 73:25–34, 2018.

[53] Thasso Griebel, Benedikt Zacher, Paolo Ribeca, Emanuele Raineri, Vincent Lacroix, Roderic Guigó, and Michael Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic acids research*, 40(20):10073–10083, 2012.

[54] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850. Society for Industrial and Applied Mathematics, 2003.

[55] Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge university press, 1997.

[56] Mitchell Guttman, Manuel Garber, Joshua Levin, Julie Donaghey, James Robinson, Xian Adiconis, Lin Fan, Magdalena Koziol, Andreas Gnirke, Chad Nusbaum, John Rinn, Eric Lander, and Aviv Regev. Ab initio reconstruction of transcriptomes of pluripotent and lineage committed cells reveals gene structures of thousands of lincrnas. *Nature Biotechnology*, 28(5):503–510, 2010.

[57] Jennifer Harrow, Adam Frankish, Jose M Gonzalez, Electra Tapanari, Mark Diekhans, Felix Kokocinski, Bronwen L Aken, Daniel Barrell, Amonida Zadissa, Stephen Searle, et al. Gencode: the reference human genome annotation for the encode project. *Genome research*, 22(9):1760–1774, 2012.

[58] Mohammad Shabbir Hasan, Xiaowei Wu, and Liqing Zhang. Performance evaluation of indel calling tools using real short-read data. *Human Genomics*, 9(1):20, 2015.

[59] James M Heather and Benjamin Chain. The sequence of sequencers: The history of sequencing dna. *Genomics*, 107(1):1–8, 2016.

[60] Steffen Heber, Max Alekseyev, Sing-Hoi Sze, Haixu Tang, and Pavel A Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics*, 18(suppl_1):S181–S188, 2002.

[61] David Stephen Horner, Giulio Pavesi, Tiziana Castrignanò, Paolo D'Onorio De Meo, Sabino Liuni, Michael Sammeth, Ernesto Picardi, and Graziano Pesole. Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing. *Briefings in Bioinformatics*, 11(2):181–197, 2010.

[62] Jeremy Hull, Susana Campino, Kate Rowlands, Man-Suen Chan, Richard R Copley, Martin S Taylor, Kirk Rockett, Gareth Elvidge, Brendan Keating, Julian Knight, et al. Identification of common genetic variation that modulates alternative splicing. *PLoS genetics*, 3(6), 2007.

[63] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

[64] Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*, pages 549–554. IEEE, 1989.

[65] Chirag Jain, Sanchit Misra, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru. Accelerating sequence alignment to graphs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 451–461. IEEE, 2019.

[66] Chirag Jain, Haowen Zhang, Yu Gao, and Srinivas Aluru. On the complexity of sequence to graph alignment. In *International Conference on Research in Computational Molecular Biology*, pages 85–100. Springer, 2019.

[67] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36(4):338, 2018.

[68] André Kahles, Kjong-Van Lehmann, Nora C Toussaint, Matthias Hüser, Stefan G Stark, Timo Sachsenberg, Oliver Stegle, Oliver Kohlbacher, Chris Sander, Samantha J Caesar-Johnson, et al. Comprehensive analysis of alternative splicing across tumors from 8,705 patients. *Cancer cell*, 34(2):211–224, 2018.

[69] André Kahles, Cheng Soon Ong, Yi Zhong, and Gunnar Rätsch. *SplAdder*: identification, quantification and testing of alternative splicing events from RNA-Seq data. *Bioinformatics*, 32(12):1840–1847, 2016.

[70] Takao Kaneo, Shoichi Tahara, and Mitsuyoshi Matsuo. Non-linear accumulation of 8-hydroxy-2'-deoxyguanosine, a marker of oxidized dna damage, during aging. *Mutation Research/DNAging*, 316(5-6):277–285, 1996.

[71] Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM journal of research and development*, 31(2):249–260, 1987.

[72] Takakazu Kawase, Yoshiki Akatsuka, Hiroki Torikai, Satoko Morishima, Akira Oka, Akane Tsujimura, Mikinori Miyazaki, Kunio Tsujimura, Koichi Miyamura, Seishi Ogawa, et al. Alternative splicing due to an intronic snp in hmsd generates a novel minor histocompatibility antigen. *Blood*, 110(3):1055–1063, 2007.

[73] Zia Khan, Joshua S Bloom, Leonid Kruglyak, and Mona Singh. A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays. *Bioinformatics*, 25(13):1609–1616, 2009.

[74] Nilesh Khiste and Lucian Ilie. E-MEM: efficient computation of maximal exact matches for very large genomes. *Bioinformatics*, 31(4):509–514, 2014.

[75] Daehwan Kim, Ben Langmead, and Steven L Salzberg. Hisat: a fast spliced aligner with low memory requirements. *Nature Methods*, 12(4):357, 2015.

[76] Daehwan Kim, Joseph M Paggi, Chanhee Park, Christopher Bennett, and Steven L Salzberg. Graph-based genome alignment and genotyping with hisat2 and hisat-genotype. *Nature Biotechnology*, 37(8):907–915, 2019.

[77] Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L Salzberg. Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome biology*, 14(4):R36, 2013.

[78] Martin Kircher and Janet Kelso. High-throughput dna sequencing–concepts and limitations. *Bioessays*, 32(6):524–536, 2010.

[79] Karlo Knezevic, Stjepan Picek, Luca Mariot, Domagoj Jakobovic, and Alberto Leporati. The design of (almost) disjunct matrices by evolutionary algorithms. In *International Conference on Theory and Practice of Natural Computing*, pages 152–163. Springer, 2018.

[80] Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie, Erich D Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology*, 30(7):693, 2012.

[81] Jana Královičová, Sophie Houngninou-Molango, Angela Krämer, and Igor Vořechovskỳ. Branch site haplotypes that control alternative splicing. *Human molecular genetics*, 13(24):3189–3202, 2004.

[82] Peter Krusche, Len Trigg, Paul C Boutros, Christopher E Mason, M Francisco, Benjamin L Moore, Mar Gonzalez-Porta, Michael A Eberle, Zivana Tezak, Samir Lababidi, et al. Best practices for benchmarking germline small-variant calls in human genomes. *Nature Biotechnology*, 37:555–560, 2019.

[83] Chee Seng Ku, En Yun Loy, Agus Salim, Yudi Pawitan, and Kee Seng Chia. The discovery of human genetic variations and their use as disease markers: past, present and future. *Journal of Human Genetics*, 55(7):403–415, 2010.

[84] Hugo YK Lam, Michael J Clark, Rui Chen, Rong Chen, Georges Natsoulis, Maeve O'huallachain, Frederick E Dewey, Lukas Habegger, Euan A Ashley, Mark B Gerstein, et al. Performance comparison of whole-genome sequencing platforms. *Nature Biotechnology*, 30(1):78, 2012.

[85] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature Methods*, 9(4):357, 2012.

[86] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, 2011.

[87] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[88] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[89] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[90] Yang I Li, David A Knowles, Jack Humphrey, Alvaro N Barbeira, Scott P Dickinson, Hae Kyung Im, and Jonathan K Pritchard. Annotation-free quantification of rna splicing using leafcutter. *Nature Genetics*, 50(1):151, 2018.

[91] Yuansheng Liu, Leo Yu Zhang, and Jinyan Li. Fast detection of maximal exact matches via fixed sampling of query k-mers and Bloom filtering of index k-mers. *Bioinformatics*, 2019.

[92] Daniel G MacArthur and Chris Tyler-Smith. Loss-of-function variants in the genomes of healthy humans. *Human Molecular Genetics*, 19(R2):R125–R130, 2010.

[93] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[94] Udi Manber and Sun Wu. Approximate string matching with arbitrary costs for text and hypertext. In *Proc. of the IAPR International Workshop on Structural and Syntactic Pattern Recognition*, pages 22–33, 1993.

[95] Tuomo Mantere, Simone Kersten, and Alexander Hoischen. Long-read sequencing emerging in medical genetics. *Frontiers in genetics*, 10:426, 2019.

[96] Giovanni Manzini. An analysis of the burrows—wheeler transform. *Journal of the ACM (JACM)*, 48(3):407–430, 2001.

[97] Guillaume Marçais, Arthur L Delcher, Adam M Phillippy, Rachel Coston, Steven L Salzberg, and Aleksey Zimin. MUMmer4: a fast and versatile genome alignment system. *PLoS computational biology*, 14(1), 2018.

[98] Marcel Margulies, Michael Egholm, William E Altman, Said Attiya, Joel S Bader, Lisa A Bemben, Jan Berka, Michael S Braverman, Yi-Ju Chen, Zhoutao

Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376, 2005.

[99] Marcel Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet. journal*, 17(1):pp–10, 2011.

[100] W Richard McCombie, John D McPherson, and Elaine R Mardis. Next-generation sequencing technologies. *Cold Spring Harbor perspectives in medicine*, 2018.

[101] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome Research*, 20(9):1297–1303, 2010.

[102] Páll Melsted and Jonathan K. Pritchard. Efficient counting of k-mers in DNA sequences using a Bloom filter. *BMC Bioinformatics*, 12:333, 2011.

[103] Giles Miclotte, Mahdi Heydari, Piet Demeester, Stephane Rombauts, Yves Van de Peer, Pieter Audenaert, and Jan Fostier. Jabba: hybrid error correction for long sequencing reads. *Algorithms for Molecular Biology*, 11(1):10, 2016.

[104] Ryan E Mills, Christopher T Luttig, Christine E Larkins, Adam Beauchamp, Circe Tsui, W Stephen Pittard, and Scott E Devine. An initial map of insertion and deletion (indel) variation in the human genome. *Genome Research*, 16(9):1182–1190, 2006.

[105] Stephen B Montgomery, David L Goode, Erika Kvikstad, Cornelis A Albers, Zhengdong D Zhang, Xinmeng Jasmine Mu, Guruprasad Ananda, Bryan Howie, Konrad J Karczewski, Kevin S Smith, et al. The origin, evolution, and functional impact of short insertion–deletion variants identified in 179 human genomes. *Genome Research*, 23(5):749–761, 2013.

[106] Margaret Morash, Hannah Mitchell, Himisha Beltran, Olivier Elemento, and Jyotishman Pathak. The role of next-generation sequencing in precision medicine: a review of outcomes in oncology. *Journal of personalized medicine*, 8(3):30, 2018.

[107] Elena Morini, Federica Sangiuolo, Daniela Caporossi, Giuseppe Novelli, and Francesca Amati. Application of next generation sequencing for personalized medicine for sudden cardiac death. *Frontiers in genetics*, 6:55, 2015.

[108] Lisle E Mose, Matthew D Wilkerson, D Neil Hayes, Charles M Perou, and Joel S Parker. Abra: improved coding indel detection via assembly-based realignment. *Bioinformatics*, 30(19):2813–2815, 2014.

[109] Martin D Muggli, Alexander Bowe, Noelle R Noyes, Paul S Morley, Keith E Belk, Robert Raymond, Travis Gagie, Simon J Puglisi, and Christina Boucher. Succinct colored de Bruijn graphs. *Bioinformatics*, 33(20):3181–3187, 2017.

[110] Julienne M Mullaney, Ryan E Mills, W Stephen Pittard, and Scott E Devine. Small insertions and deletions (indels) in human genomes. *Human Molecular Genetics*, 19(R2):R131–R136, 2010.

[111] James K Mullin. A second look at Bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.

[112] Harun Mustafa, Ingo Schilken, Mikhail Karasikov, Carsten Eickhoff, Gunnar Rätsch, and André Kahles. Dynamic compression schemes for graph coloring. *Bioinformatics*, 35(3):407–414, 2018.

[113] Oxford Nanopore. World first: continuous dna sequence of more than a million bases achieved with nanopore sequencing. `https://nanoporetech.com/about-us/news/world-first-continuous-dna-sequence-more-million-bases-achieved-nanopore-sequencing` [Online, accessed: 25 October, 2019].

[114] Gonzalo Navarro. Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237(1):455–463, 2000.

[115] Gonzalo Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014.

[116] National Human Genome Research Institute (NHGRI). Talking glossary of genetic terms. `https://www.genome.gov/genetics-glossary` [Online, accessed: 25 October, 2019].

[117] Enno Ohlebusch, Simon Gog, and Adrian Kügel. Computing matching statistics and maximal exact matches on compressed full-text indexes. In *Interna-*

tional Symposium on String Processing and Information Retrieval, pages 347–358. Springer, 2010.

[118] Fanny-Dhelia Pajuste, Lauris Kaplinski, Märt Möls, Tarmo Puurand, Maarja Lepamets, and Maido Remm. FastGT: an alignment-free method for calling common SNVs directly from raw sequencing reads. *Scientific Reports*, 7(1):2537, 2017.

[119] Qun Pan, Ofer Shai, Leo J Lee, Brendan J Frey, and Benjamin J Blencowe. Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing. *Nature Genetics*, 40(12):1413, 2008.

[120] Louis Papageorgiou, Picasi Eleni, Sofia Raftopoulou, Meropi Mantaiou, Vasileios Megalooikonomou, and Dimitrios Vlachakis. Genomic big data hitting the storage bottleneck. *EMBnet. journal*, 24, 2018.

[121] Kunsoo Park and Dong Kyue Kim. String matching in hypertext. In *Annual Symposium on Combinatorial Pattern Matching*, pages 318–329. Springer, 1995.

[122] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417, 2017.

[123] Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature Biotechnology*, 33(3):290, 2015.

[124] Pierre Peterlongo, Chloe Riou, Erwan Drezen, and Claire Lemaitre. DiscoSnp++: de novo detection of small variants from raw unassembled read set(s). *BioRxiv*, 2017.

[125] Martin O Pollard, Deepti Gurdasani, Alexander J Mentzer, Tarryn Porter, and Manjinder S Sandhu. Long reads: their purpose and place. *Human molecular genetics*, 27(R2):R234–R241, 2018.

[126] Ryan Poplin, Valentin Ruano-Rubio, Mark A DePristo, Tim J Fennell, Mauricio O Carneiro, Geraldine A Van der Auwera, David E Kling, Laura D Gauthier, Ami Levy-Moonshine, David Roazen, et al. Scaling accurate genetic variant discovery to tens of thousands of samples. *BioRxiv*, page 201178, 2018.

[127] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, 35(19):3599–3607, 2019.

[128] Andy Rimmer, Hang Phan, Iain Mathieson, Zamin Iqbal, Stephen RF Twigg, Andrew OM Wilkie, Gil McVean, Gerton Lunter, WGS500 Consortium, et al. Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, 46(8):912, 2014.

[129] Adam Roberts, Harold Pimentel, Cole Trapnell, and Lior Pachter. Identification of novel transcripts in annotated genomes using RNA-Seq. *Bioinformatics*, 27(17):2325–2329, 2011.

[130] Mark Rogers, Julie Thomas, Anireddy Reddy, and Asa Ben-Hur. Splicegrapher: detecting patterns of alternative splicing from RNA-Seq data in the context of gene models and EST data. *Genome Biology*, 13(1):R4, 2012.

[131] Gustavo AT Sacomoto, Janice Kielbassa, Rayan Chikhi, Raluca Uricaru, Pavlos Antoniou, Marie-France Sagot, Pierre Peterlongo, and Vincent Lacroix. Kis Splice: de-novo calling alternative splicing events from RNA-seq data. In *BMC Bioinformatics*, volume 13, page S5. BioMed Central, 2012.

[132] Leena Salmela and Eric Rivals. Lordec: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514, 2014.

[133] Steven L Salzberg. Open questions: How many genes do we have? *BMC Biology*, 16(1):94, 2018.

[134] Michael Sammeth, Sylvain Foissac, and Roderic Guigó. A general definition and nomenclature for alternative splicing events. *PLoS computational biology*, 4(8), 2008.

[135] Frederick Sanger, Steven Nicklen, and Alan R Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467, 1977.

[136] Lecia V Sequist, Renato G Martins, David Spigel, Steven M Grunberg, Alexander Spira, Pasi A Janne, Victoria A Joshi, David McCollum, Tracey L Evans, Alona Muzikansky, et al. First-line gefitinib in patients with advanced non-small-cell lung cancer harboring somatic egfr mutations. *Journal of Clinical Oncology*, 26(15):2442–2449, 2008.

[137] Joao Carlos Setubal and Joao Meidanis. *Introduction to computational molecular biology*. PWS Pub. Boston, 1997.

[138] Julian Seward. bzip2 and libbzip2. `http://www.bzip.org` [Online, accessed: 25 October, 2019], 1996.

[139] Ariya Shajii, Deniz Yorukoglu, Yun William Yu, and Bonnie Berger. Fast genotyping of known SNPs through approximate k-mer matching. *Bioinformatics*, 32(17):i538–i544, 2016.

[140] Shihao Shen, Juw Won Park, Zhi xiang Lu, Lan Lin, Michael D. Henry, Ying Nian Wu, Qing Zhou, and Yi Xing. rMATS: robust and flexible detection of differential alternative splicing from replicate RNA-Seq data. *Proc Natl Acad Sci*, 111(51):E5593–E5601, 2014.

[141] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nature Biotechnology*, 26(10):1135, 2008.

[142] Stephen T Sherry, M-H Ward, M Kholodov, J Baker, Lon Phan, Elizabeth M Smigielski, and Karl Sirotkin. dbsnp: the ncbi database of genetic variation. *Nucleic acids research*, 29(1):308–311, 2001.

[143] Jared Simpson, Kim Wong, Shaun Jackman, Jacqueline Schein, Steven Jones, and İnanç Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[144] Nimisha Singla and Deepak Garg. String matching algorithms and their applicability in various applications. *International Journal of Soft Computing and Engineering (IJSCE)*, 1(6):2231–2307, 2012.

[145] Barton E Slatko, Andrew F Gardner, and Frederick M Ausubel. Overview of next-generation sequencing technologies. *Current protocols in molecular biology*, 122(1):e59, 2018.

[146] Christof C Smith, Sara R Selitsky, Shengjie Chai, Paul M Armistead, Benjamin G Vincent, and Jonathan S Serody. Alternative tumour-specific antigens. *Nature Reviews Cancer*, page 1, 2019.

[147] Wendy Weijia Soon, Manoj Hariharan, and Michael P Snyder. High-throughput sequencing for biology and medicine. *Molecular systems biology*, 9(1), 2013.

[148] Avi Srivastava, Hirak Sarkar, Nitish Gupta, and Rob Patro. RapMap: a rapid, sensitive and accurate tool for mapping RNA-seq reads to transcriptomes. *Bioinformatics*, 32(12):i192–i200, 2016.

[149] Daniel S. Standage, C. Titus Brown, and Fereydoun Hormozdiari. Kevlar: A mapping-free framework for accurate discovery of de novo variants. *iScience*, 18:28 – 36, 2019. Special Issue: RECOMB-Seq 2019.

[150] Peter H Sudmant, Tobias Rausch, Eugene J Gardner, Robert E Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, et al. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015.

[151] Chen Sun, Robert S. Harris, Rayan Chikhi, and Paul Medvedev. AllSome sequence Bloom trees. In *Research in Computational Molecular Biology - 21st Annual International Conference, RECOMB 2017*, pages 272–286, 2017.

[152] Chen Sun and Paul Medvedev. Toward fast and accurate SNP genotyping from whole genome sequencing data for bedside diagnostics. *Bioinformatics*, 35(3):415–420, 2018.

[153] Vivian Tam, Nikunj Patel, Michelle Turcotte, Yohan Bossé, Guillaume Paré, and David Meyre. Benefits and limitations of genome-wide association studies. *Nature Reviews Genetics*, page 1, 2019.

[154] Jamal Tazi, Nadia Bakkour, and Stefan Stamm. Alternative splicing and disease. *Biochimica et Biophysica Acta (BBA)-Molecular Basis of Disease*, 1792(1):14–26, 2009.

[155] Chris Thachuk. Indexing hypertext. *J. Discrete Algorithms*, 18:113–122, 2013.

[156] Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14(5):S15, 2013.

[157] Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, 2010.

[158] Juan L Trincado, Juan C Entizne, Gerald Hysenaj, Babita Singh, Miha Skalic, David J Elliott, and Eduardo Eyras. Suppa2: fast, accurate, and uncertainty-aware differential splicing analysis across multiple conditions. *Genome biology*, 19(1):40, 2018.

[159] Natalie A Twine, Karolina Janitz, Marc R Wilkins, and Michal Janitz. Whole transcriptome sequencing reveals gene expression and splicing differences in brain regions affected by alzheimer's disease. *PloS one*, 6(1), 2011.

[160] Eli Upfal and Michael Mitzenmacher. *Probability and computing: randomized algorithms and probabilistic analysis*, pages 109–111. Cambridge university press, 2005.

[161] Erwin L Van Dijk, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes. Ten years of next-generation sequencing technology. *Trends in genetics*, 30(9):418–426, 2014.

[162] Jorge Vaquero-Garcia, Alejandro Barrera, Matthew R Gazzara, Juan Gonzalez-Vallinas, Nicholas F Lahens, John B Hogenesch, Kristen W Lynch, and Yoseph Barash. A new view of transcriptome complexity and regulation through the lens of local splicing variations. *elife*, 5, 2016.

[163] Charline Vauchy, Clementine Gamonet, Christophe Ferrand, Etienne Daguindau, Jeanne Galaine, Laurent Beziaud, Adrien Chauchet, Carole J Henry Dunand, Marina Deschamps, Pierre Simon Rohrlich, et al. Cd20 alternative splicing isoform generates immunogenic cd 4 helper t epitopes. *International journal of cancer*, 137(1):116–126, 2015.

[164] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[165] Sebastiano Vigna. Broadword implementation of rank/select queries. In *International Workshop on Experimental and Efficient Algorithms*, pages 154–168. Springer, 2008.

[166] Michaël Vyverman, Bernard De Baets, Veerle Fack, and Peter Dawyndt. A long fragment aligner called ALFALFA. *BMC Bioinformatics*, 16(1):159, May 2015.

[167] Michaël Vyverman, Bernard De Baets, Veerle Fack, and Peter Dawyndt. essaMEM: finding maximal exact matches using enhanced sparse suffix arrays. *Bioinformatics*, 29(6):802–804, 2013.

[168] Toshifumi Wakai, Pankaj Prasoon, Yuki Hirose, Yoshifumi Shimada, Hiroshi Ichikawa, and Masayuki Nagahashi. Next-generation sequencing-based clinical sequencing: toward precision medicine in solid tumors. *International journal of clinical oncology*, 24(2):115–122, 2019.

[169] Eric T Wang, Rickard Sandberg, Shujun Luo, Irina Khrebtukova, Lu Zhang, Christine Mayr, Stephen F Kingsmore, Gary P Schroth, and Christopher B Burge. Alternative isoform regulation in human tissue transcriptomes. *Nature*, 456(7221):470, 2008.

[170] Yan Wang, Jing Liu, BO Huang, Yan-Mei Xu, Jing Li, Lin-Feng Huang, Jin Lin, Jing Zhang, Qing-Hua Min, Wei-Ming Yang, et al. Mechanism of alternative splicing and its regulation. *Biomedical reports*, 3(2):152–158, 2015.

[171] Stephen T Warren, Fuping Zhang, Greg R Licameli, and Jeanne F Peters. The fragile x site in somatic cell hybrids: an approach for molecular cloning of fragile sites. *Science*, 237(4813):420–423, 1987.

[172] James D Watson, Francis HC Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[173] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 1–11. IEEE, 1973.

[174] What is FASTA format? `https://zhanglab.ccmb.med.umich.edu/FASTA/` [Online, accessed: 25 October, 2019].

[175] Roland Wittler. Alignment-and reference-free phylogenomics with colored de-Bruijn graphs. *arXiv preprint arXiv:1905.04165*, 2019.

[176] Kai Ye, Marcel H Schulz, Quan Long, Rolf Apweiler, and Zemin Ning. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, 25(21):2865–2871, 2009.

[177] Justin M Zook, Brad Chapman, Jason Wang, David Mittelman, Oliver Hofmann, Winston Hide, and Marc Salit. Integrating human sequence data sets provides

a resource of benchmark SNP and indel genotype calls. *Nature Biotechnology*, 32(3):246–251, 2014.