UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
DIPARTIMENTO DI INFORMATICA, SISTEMISTICA E COMUNICAZIONE
DOTTORATO DI RICERCA IN INFORMATICA - CICLO XXV


PH.D. THESIS

# Algorithms for Next Generation Sequencing Data Analysis

*Author:* Stefano BERETTA

*Advisors:* Prof.ssa Paola BONIZZONI
Prof. Gianluca DELLA VEDOVA

February 2013

*To my parents*
*Thank you*

# *Acknowledgements*

First and foremost, I want to thank my parents for their love and support throughout my life. Thank you both for giving me everything. This thesis would certainly not have existed without them.

I would like to thank my two advisors, Prof. Paola Bonizzoni and Prof. Gianluca Della Vedova for their guidance during my Ph.D.

I am extremely grateful to Yuri Pirola, Raffaella Rizzi and Riccardo Dondi for their continuous encouragement and for the useful discussions. Their contribution has been fundamental.

A special thank goes to Mauro, my colleague and my friend. I have spent wonderful moments with him and I could not have a better mate.

My thanks go to all my office mates, Luca, Enrico and Carlo, and to my other Ph.D. colleagues, Lorenza and Luca, with whom I shared good and bad moments during these three years.

I wish to acknowledge all the people I met during the Ph.D., and in particular to the patience and understanding shown by Laura, Ilaria and Gaia to me. It must not have been easy to bear with me. I would also like to offer my special thanks to Silvia and Elena, for their wonderful encouragement.

I am grateful to all my friends, especially Mauro and Stefano. I have known them for a long time and spending my spare time with them is always wonderful.

Me gustaría agradecer al Prof. Gabriel Valiente por dirigirme durante mi estancia en la *Universitat Politècnica de Catalunya* y a Daniel Alonso-Alemany por las discusiones fructíferas y por toda su ayuda. Trabajar con ellos ha sido un verdadero placer.
También quiero dar las gracias a Jorge, Alessandra, Ramon, Javier, Nikita, Eva, Albert y todos los otros compañeros del despacho.

# Contents

**Acknowledgements** iii

**List of Figures** vii

**List of Tables** x

**1 Introduction** 1

**2 Background** 6
- 2.1 DNA and RNA . . . 6
- 2.2 Gene and Alternative Splicing . . . 7
- 2.3 Sequencing . . . 10
  - 2.3.1 Sanger Technique . . . 10
  - 2.3.2 Next Generation Sequencing . . . 11
- 2.4 Data . . . 21
  - 2.4.1 RNA-Seq . . . 24
- 2.5 Definitions . . . 25
  - 2.5.1 Strings . . . 25
  - 2.5.2 Graphs and Trees . . . 26

**3 Reconstructing Isoform Graphs from RNA-Seq data** 31
- 3.1 Introduction . . . 32
  - 3.1.1 Transcriptome Reconstruction . . . 34
  - 3.1.2 Splicing Graph . . . 39
- 3.2 The Isoform Graph and the SGR Problem . . . 41
- 3.3 Unique solutions to SGR . . . 45
- 3.4 Methods . . . 46
  - 3.4.1 Isomorphism between predicted and true isoform graph . . . 50
  - 3.4.2 Low coverage, errors and SNP detection . . . 51
  - 3.4.3 Repeated sequences: stability of graph $G_R$ . . . 52
- 3.5 Experimental Results . . . 53
- 3.6 Conclusions and Future Work . . . 60

**4 Taxonomic Assignment to Multiple Trees in Metagenomics** 63
- 4.1 Introduction to Metagenomics . . . 65
  - 4.1.1 Taxonomic Alignment . . . 68

4.1.2 Taxonomic Assignment with TANGO . . . . . 69
4.2 Taxonomy Analysis . . . . . 72
4.2.1 16S Databases . . . . . 73
4.2.2 Tree Analysis . . . . . 74
4.2.3 Inferring Trees from Lineages . . . . . 76
4.3 Minimum Penalty Score Calculation . . . . . 81
4.3.1 An Asymptotically Optimal Algorithm . . . . . 82
4.4 TANGO Improvement . . . . . 86
4.4.1 New Data Structures . . . . . 86
4.4.2 Optimal Algorithm Implementation . . . . . 89
4.4.3 Input Mapping . . . . . 91
4.5 Conclusions and Future Work . . . . . 92

**A Additional Data** **96**

**Bibliography** **104**

# List of Figures


2.1 The central dogma of molecular biology. The DNA codes for the production of messenger RNA during transcription, which is then translated to proteins. . . . . 7

2.2 Basic types of alternative splicing events. . . . . 9

2.3 Sanger sequencing process. Figure 2.3(a) illustrates the products of the polymerase reaction, taking place in the ddCTP tube. The polymerase extends the labeled primer, randomly incorporating either a normal dCTP base or a modified ddCTP base. At every position where a ddCTP is inserted, the polymerization terminates; the final result is a population of fragments. The length of each fragment represents the relative distance from the modified base to the primer. Figure 2.3(b) shows the electrophoretic separation of the products of each of the four reaction tubes (ddG, ddA, ddT, and ddC), run in individual lines. The bands on the gel represent the respective fragments shown to the right. The complement of the original template (read from bottom to top) is given on the left margin of the sequencing gel. . . . . 12

2.4 Roche 454 sequencing. In the library construction, the DNA is fragmented and ligated to adapter oligonucleotides, and then the fragments are clonally amplified by emulsion PCR. The beads are then loaded into picotiter-plate wells, where iterative pyrosequencing process is performed. 14

2.5 Illumina sequencing 1/3. The DNA sample is fragmented and the adapters are bound to the ends (Figure 2.5(a)); the obtained single-strained fragments are attached to the flow cell surface (Figure 2.5(b)) and then they are "bridged" (Figure 2.5(c)). . . . . 16

2.6 Illumina sequencing 2/3. The fragments become double-strained (Figure 2.6(a)) and then they are denatured. This process leaves anchored single-strained fragments (Figure 2.6(b)). After a complete amplification, millions of clusters are formed on the flow cell surface (Figure 2.6(c)). . . 16

2.7 Illumina sequencing 3/3. The clusters are washed, leaving only the forward strands and, to initiate the sequencing process, primer, polymerase and a 4 colored terminators are added. The primers are bound to the primers of the strand and, after that, the first terminator is incorporated (Figure 2.7(a)). By iterating the incorporation process, all the sequence nucleotides are added and the laser activates the fluorescence (Figure 2.7(b)). During the incorporation a computer reads the image and determines the sequence of bases (Figure 2.7(c)). . . . . 17

2.8 Sequencing by ligation. In (1) the primer is annealed to the adaptor and the first two bases of the probe are bound to the first two bases of the sequence. The fluorescence signal is read (2) and the last bases of the probe are cleaved, so that the new phosphate group is created (3). This ligation process is repeated for seven cycles (4), in order to extend the primer. Finally, this latter primer is melted off and a new one is annealed (5). The previous steps are repeated for all the new primers, with different offsets (6). . . . . 19
2.9 Summary of the sequencing by ligation process showing, for each base of the read sequence, the positions of the interrogations (2 for each position). 19
2.10 Examples of graphs. Both the graphs have 5 nodes and 7 edges, but in Figure 2.10(a) the edges are unordered (i.e. $(u, v) = (v, u)$) and they are represented as simple lines. Instead in Figure 2.10(b) the edges are ordered (i.e. $(u, v) \neq (v, u)$) and they are represented as arrows indicating the direction. . . . . 26
2.11 Example of *De Bruijn graph* on the binary alphabet, with $k = 4$. Nodes represent all the possible substrings of length $k - 1$ and there is an edge between two nodes if there is a $k$-mer that has the first $(k - 1)$-mer as prefix and the second $(k - 1)$-mer as suffix. The $k$-mer is obtained by the fusion of the $(k - 1)$-mers at the two ends. In this example edges are labeled by the symbols that must be concatenated to the first $(k-1)$-mer, to obtain the $k$-mer. . . . . 28
2.12 Example of rooted tree. The node $r$ is the root of the tree, the nodes $n_3$, $n_4$, $n_5$, $n_7$, $n_9$, $n_{10}$ and $n_{11}$ are the leaves and $n_1$, $n_2$, $n_6$ and $n_8$ are the internal nodes. As an example, $n_6$ is the parent of $n_8$ and $n_2$ is one of the children of $n_1$ (the other one is $n_5$). . . . . 29
2.13 Example of tree traversals. Given the tree, this example shows the order in which the nodes are visited, for each of the three types of traversal. . . 30

3.1 Genome Independent assembly. Reads are split into $k$-mers (1) and the *De Bruijn graph* is built, with the possibility of having sequencing errors or SNPs and also deletions or introns, that generate different branches (2). The graph is then contracted by collapsing all its linear paths (3). The contracted graph is traversed (4) and each isoform is assembled from an identified path (5). . . . . 38
3.2 Example of expressed gene. Given a gene in which its isoforms are composed of blocks (that for simplicity are associated to exons: $A$, $B$, $C$ and $D$) extracted from the genome, we represent the sequence $B$ of blocks based on the (genomic) position. The expressed gene $\mathcal{G} = \langle B, F \rangle$ is constructed from the block sequence $B$ and its isoform composition. . . . . 42
3.3 Examples of Isoform Graphs. These two examples show the isoform composition of a gene (1) with the corresponding block sequence (2) and their isoform graphs (3). Capital letters correspond to exons. In (a) is represented a skipping of the two consecutive exons $B$ and $C$ of the second isoform w.r.t. the first one. In (b) is represented an alternative donor site variant between exons $A$ and $A'$ and two mutually exclusive exons $C$ and $D$. Notice that, in this figure, the isoforms represented are classical. . . . 43
3.4 Alternative splicing graph compatible with the reads of the expressed gene of Figure 3.3(b) . . . . . 44

3.5 Example of read classification. The example shows two transcripts, where the first one is composed of 3 blocks ($A$, $B$ and $C$) and the second one of 2 blocks ($A$ and $C$). Moreover, reads coming from those blocks are partitioned into unspliced (white rectangles) and spliced (gray rectangles). Notice that the two reads sharing the same suffix of block $A$ (of length $k$) and those two sharing the same prefix of block $C$ (of length $k$) are labeled with $suf(A, k)$ and $pre(C, k)$, respectively. . . . . 47
3.6 Sn and PPV values at vertex level in the first 3.6(a) and in the second 3.6(b) experiments. Each point represents a gene. . . . . 57
3.7 Sn and PPV values at arc level in the first 3.7(a) and in the second 3.7(b) experiments. Each point represents a gene. . . . . 58
3.8 Gene POGZ. The correct isoform graph (on the left) and the predicted isoform graph (on the right) predicted in the third experiment. Number in the nodes represent the lengths of the blocks. The difference between the two graphs consists of the gray vertex that is missing in the predicted graph. . . . . 59

4.1 Example of taxonomic tree. The example shows a rooted tree with maximum depth 8 (7 taxonomic ranks plus the Root), underling a subset of nodes and their associated taxonomic ranks. . . . . 67
4.2 Leaf set partition used for the *Penalty Score* calculation in TANGO. In this example, given a read $R_i \in R$ the matches are represented as circled leaves, i.e. $M_i = \{S_1, S_5, S_6, S_7\}$. The leaves of $T_i$, which is the subtree rooted at the LCA of $M_i$, can be partitioned into 4 disjointed subsets ($TP_{i,j}$, $FP_{i,j}$, $TN_{i,j}$ and $FN_{i,j}$) for each node $j$ in $T_i$. $T_{i,j}$ represents the subtree of $T_i$ induced by the choice of node $j$ as representative of $M_i$. . . . 72
4.3 Example of tree with its representation in *Newick* format. . . . . 75
4.4 Example of "binarization process". On the left there is an example of tree in which a node $r$ has 4 children (named $n_1$, $n_2$, $n_3$ and $n_4$). On the right there is the same tree after the binarization, in which the circled node are the ones added in the process. . . . . 76
4.5 An example of rooted tree $T$ is shown in 4.5(a), where the circled leaves are the ones in the set $S$, i.e. $S = \{3, 4, 9, 11\}$. In 4.5(b) the skeleton tree obtained from Definition 4.2, starting from the tree $T$ and the subset of leaves $S$. . . . . 82
4.6 Example of tree and the corresponding data structure obtain by taking the nodes as ordered from left to right (according to the picture). In particular, for each node we use three pointers: *parent*, *first-child* and *next-sibling*. Notice the following special cases: the *Root* node (which has itself as parent), the leaves (which do not have a first child) and the last siblings (which do not have the next sibling). . . . . 87

# List of Tables


2.1 Roche/454 Life Sciences NGS instruments. . . . . 15
2.2 Illumina/Solexa NGS instruments. . . . . 17
2.3 Life Technologies/SOLiD NGS instruments. . . . . 20
2.4 Error rate comparison of different sequencing platforms. . . . . 22
2.5 Quality score and base calling accuracy. . . . . 22

4.1 List of the main databases for 16S ribosomal RNA (rRNA) and the number of Bacteria and Archaea (high-quality) sequences present in the current version of the database. . . . . 73
4.2 Statistics on the taxonomic trees. In the specific, the second column reports the type of the tree which can be $n$-ary or binary; the third and fourth columns report the number of leaves and total nodes, respectively; the last column reports the maximum depth of the tree. . . . . 77
4.3 Statistics on the reconstructed RDP taxonomy. In the specific, for each depth of the tree, the number of nodes (resp. leaves) is reported. . . . . 78

A.1 Details of the first experiment of Section 3.5. . . . . 96
A.2 Details of the second experiment of Section 3.5. . . . . 100

# Chapter 1

# Introduction

Due to the continuously growing amount of produced data, methods in computer science are becoming more and more important in biological studies. More specifically, the advent of *Next-Generation Sequencing* (NGS) techniques has opened new challenges and perspectives that were not even thinkable some years ago. One of the key points of these new methods is that they are able to produce a huge quantity of data at much lower costs, with respect to the previous sequencing techniques. Moreover, the growth rate of these data is higher than the one of semiconductors, meaning that approaches based on Moore's law do not work. This also means that algorithms and programs that were used in the past are no longer applicable in this new "context" and so new solutions must be found. This is mainly due to two main reasons. On one hand, data are of different nature. In fact, the produced sequences are much shorter than the ones of the classical methods. On the other hand, also the volume of data is changed, as the new methods can produce millions or billions of sequences, that is orders of magnitude more than before. Hence, efficient procedures are required to process NGS data. Moreover, new approaches are necessary to face the continuously emerging computational problems, that are fundamental to take advantage of the produced data. Two of the fields in bioinformatics that are most influenced, by the introduction of the Next-Generation Sequencing methods, are *transcriptomics* and *metagenomics*. In this thesis we address two central problems of these latter fields.

The central goal in transcriptome analysis is the identification and quantification of all full-length transcripts (or isoforms) of genes, produced by *alternative splicing*. This mechanism has an important role in the regulation of the protein expressions that influences several cellular and developmental processes. Moreover, recent studies point out that alternative splicing may also play a crucial role in the differentiation of stem cells, thus opening a new perspective in transcriptome analysis [1]. Understanding such

a mechanism can lead to new discoveries in many fields. Therefore, a lot of effort has been invested in the study of methods for alternative splicing analysis and full-length transcript prediction, which, however, operate on data produced by the old Sanger technology. On the other hand, NGS technologies have opened new possibilities in the study of the alternative splicing, since data from different individuals and cells are available for large scale analysis. With the specific goal of analyzing NGS data, some computational methods for assembling full-length transcripts or predicting splice sites from such data have been recently proposed. However, these tools by alone are not able to provide an overview of alternative splicing events of a specific set of genes. In particular, a challenging task is the comparison of transcripts derived from NGS data to predict AS variants and to summarize the data into a biological meaningful set of AS events. Another issue involves the prediction of AS events from NGS data without a specific reference genome, as the currently available tools can perform this task with limited accuracy, even with the presence of a reference.

In this thesis, to tackle the previously mentioned issues:

1. we face the problem of providing a representation of the alternative splicing variants, in absence of the reference genome, by using the data coming from NGS techniques;
2. we provide an efficient procedure to construct such a representation of all the alternative splicing events.

The solution we propose for the first problem is the definition of the *isoform graph*, which is a graph representation that summarize all the full-length transcripts of a gene. We investigate the conditions under which it is possible to construct such a graph, by using only RNA-Seq reads, which are the sequences obtained when sequencing a transcript with NGS methods. These conditions guarantee the possibility to obtain the correct graph from the NGS data. We analyze each condition in detail, in order to explore the possibility of reconstructing the isoform graph in absence of a reference genome.

For the second problem we propose a solution to construct the isoform graph, also when the necessary conditions, for its complete reconstruction, are not satisfied. In the latter case the obtained graph is an approximation of the correct one. The algorithm we propose is a three-step procedure in which the reads are first partitioned into two subsets and then assembled to create the graph. We also provide an experimental validation of this approach, which is based on an efficient implementation of the algorithm, that uses the RNA-Seq reads coming from one or more genes. More precisely, in order to verify its accuracy and its stability, we have performed different tests that reflect

different conditions of the input set. We show the scalability of our implementation to huge amount of data. Limiting computational (time and space) resources used by our algorithm is a main aim of ours, when compared to other tools of transcriptome analysis. Our algorithmic approach works in time that is linear in the number of reads, with space requirements bounded by the size of the hash tables used to memorize the sequences. Finally, our computational approach for representing alternative splicing variants is different from current methods of transcriptome analysis that focus on using RNA-Seq data for reconstructing the set of transcripts of a gene and estimating their abundance. More specifically, we aim to summarize genome-wide RNA-Seq data into graphs, each representing an expressed gene and the alternative splicing events occurring in the specific processed sample. On the contrary, current tools do not give a concise result, such as a structure for each gene, nor they provide a easy-to-understand list of alternative splicing events for a gene.

Part of this work has been presented in the following international venues:

- **S. Beretta**, P. Bonizzoni, G. Della Vedova, R. Rizzi, *Reconstructing Isoform Graphs from RNA-Seq data.*, 2012, IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2012, Philadelphia PA, USA [2]
- Y. Pirola, R. Rizzi, **S. Beretta**, E. Picardi, G. Pesole, G. Della Vedova, P. Bonizzoni, *PIntronNext: a fast method for detecting the gene structure due to alternative splicing via ESTs, mRNAs, and RNA-Seq data.*, EURASNET Symposium on Regulation of Gene Expression through RNA Splicing, 2012, Trieste, Italy
- **S. Beretta**, P. Bonizzoni, G. Della Vedova, R. Rizzi, *Alternative Splicing from RNA-seq Data without the Genome.*, ISMB/ECCB - 8th Special Interest Group meeting on Alternative Splicing (AS-SIG), 2011, Vienna, Austria
- **S. Beretta**, P. Bonizzoni, G. Della Vedova, R. Rizzi, *Identification of Alternative Splicing variants from RNA-seq Data.*, Next Generation Sequencing Workshop, 2011, Bari, Italy

The second part of this thesis is focused on a fundamental problem in metagenomics. The aim of this continuously growing field of research is the study of uncultured organisms to understand the true diversity of microbes, their functions, cooperation and evolution, in environments such as soil, water, ancient remains of animals, or the digestive system of animals and humans. With the advent of NGS technologies, which do not require cloning or PCR amplification, there has been an increase in the number of metagenomic projects. As anticipated, the use of NGS methods creates new problems and challenges in bioinformatics, which has to handle and analyze these datasets in an efficient and

useful way. A better understanding of the microbial world is one of the main goals of metagenomic studies and, in order to accomplish this challenge, the comparison among multiple datasets is a required task. Despite the improvement of various techniques, there is still a need for new computational approaches and algorithmic methods to find a solution to emerging problems related to NGS data analysis. One of these problems is the assignment of reads to a reference taxonomy. This task is crucial in identifying the species present in a sample and also to classify them. More specifically, starting from a set of NGS reads coming from a metagenomic sample, the objective is to assign them to a reference taxonomy, in order to understand the composition of the sample. In order to solve this problem, a commonly adopted technique is the alignment of reads to the taxonomy, followed by the correct assignment of ambiguous reads (i.e. the ones that have multiple valid matches) in the same taxonomy. This latter procedure is usually based on computing the *Lowest Common Ancestor* (LCA) of all the alignments of the read.

In the thesis we address some challenging open questions on the problem of assigning reads to a taxonomy, mainly:

1. having a fast and at the same time accurate procedure to process reads and assign them to the tree;
2. making possible the use of taxonomy that differ in the ranking of species.

The two above mentioned tasks are accomplished in this thesis by designing novel procedures in the TANGO method. TANGO is an example of software that bases the assignment on the calculation of a penalty score function for each of the candidate nodes of the taxonomy. In particular, we improve the time efficiency and accuracy of TANGO by realizing an fast procedure for the computation of the minimum penalty score value of the candidate nodes, also supported by a more efficient implementation of the software. To do this, we take advantage of an algorithm for inferring the so called *skeleton tree*, given a tree and a subset of its leaves. We prove that it is possible to compute the penalty score in an optimal way by using the previously mentioned tree, and we provide an algorithmic procedure to accomplish this task. The latter method is implemented by using a simple data structure to represent the tree, which contributes to the improvement of performance of the overall method. The second main issue mentioned before has been addresses by developing a method to contract the taxonomic trees keeping only specific ranks, in order to facilitate the comparison among different taxonomies. Our contributions provide an improvement to the problem of assigning the reads to a reference taxonomy, which is a basic step in a metagenomic analysis.

Part of this work has been included in:

- D. Alonso-Alemany, A. Barre, **S. Beretta**, P. Bonizzoni, M. Nikolski, G. Valiente, *Further Steps in TANGO: Improved Taxonomic Assignment in Metagenomics.*, 2012, (Submitted)

In the rest of this thesis all the results mentioned above are explained in detail. In **Chapter 2** the biological notions, necessary for the understanding of the following descriptions, are introduced. In addition to this, the basic algorithmic concepts and notations are formalized, providing a complete background. **Chapter 3** starts with a brief introduction and describes the current approaches for the reconstruction of transcripts, from NGS data. Then the concept of isoform graph is explained and the necessary and sufficient conditions to construct such a graph, are provided. An algorithm for the computation of the isoform graph is given, with also a particular attention on its implementation. After that, an experimental validation, aimed to verify the accuracy of the predictions and also the stability of this method, is described. Finally, some conclusions and future developments of this approach are proposed. **Chapter 4** reports the work in metagenomics, performed during my period abroad at the *Algorithms, Bioinformatics, Complexity and Formal Methods Research Group of the Technical University of Catalonia in Barcelona, Spain*, in collaboration with Daniel Alonso-Alemany, under the supervision of Prof. Gabriel Valiente. More specifically, after an introduction to metagenomics, the problem of the taxonomic assignment is described, with a particular attention on the solution provided by the TANGO software. After that, we describe the proposed method to contract the main different available taxonomies. We also provide a description of the new data structures used to represent trees. The focus is then moved to the minimum penalty score calculation, and in particular, to the proof of this latter problem that involves the skeleton tree. Finally, all the improvements of the TANGO software are described and the chapter is concluded by underlining some possible developments of this work.

# Chapter 2

# Background

In this chapter basic notions concerning the biological and algorithmic background are given. After the introduction of fundamental biological concepts, the attention is focused on the sequencing process by describing classical and new techniques. In particular, the Next-Generation Sequencing platforms are shown, discussing the methods they adopt. The produced data are then analyzed and the specific case of RNA-Seq is explained. Finally some algorithmic concepts and formal notations used in the rest of the thesis are introduced (for a more detailed explanation see [3]).

## 2.1 DNA and RNA

The deoxyribonucleic acid (DNA) molecule is the carrier of the genetic information in our cells. In fact all the information it contains are passed from organisms to their offspring during the process of reproduction. More specifically, each DNA molecule consists of the four nucleotides Adenine (A), Cytosine (C), Guanine (G), and Thymine (T), with backbones made of sugars and phosphate groups joined by ester bonds. It is organized as a double-helix and the two strands of the DNA molecule are complementary to each other. The base-pairing is fixed: A is always complementary to T, and G is always complementary to C. This double-helix model was proposed in 1953 by James Watson, an American scientist, and Francis Crick, a British researcher, and it has not been changed much since then. This discovery was made by studying the X-ray diffraction patterns, and this allowed to the two scientists to build models and to figure out the double-helix structure of DNA, a structure that enables it to carry biological information from one generation to the following.

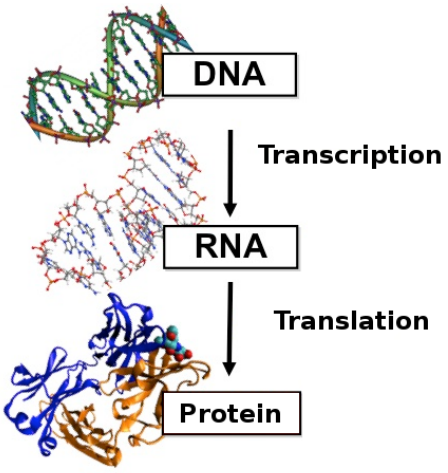



FIGURE 2.1: The central dogma of molecular biology. The DNA codes for the production of messenger RNA during transcription, which is then translated to proteins.

The complete genome is composed of *chromosomes* which are a set of different DNA molecules. Eukaryotes, i.e. organisms that have cells with a structure made by membranes, store the chromosomes in the nuclei of their cells. For example, in humans, the genome has a total of more than 3 billion of nucleotides that are organized in 23 chromosomes, each of which appears in two copies in each cell. An important functional unit in a chromosome is the gene, which is described by the so called *central dogma of molecular biology*.

The central dogma in molecular biology (shown in Figure 2.1) says that portions of DNA, called gene regions, are *transcribed* into ribonucleic acid (RNA), which is then *translated* into proteins. The RNA molecules differ from the DNA ones because they are often single strained and the Thymine (T) nucleotide is replaced by Uracil (U). In addition to this, the half-life of RNA molecules is shorter than that of DNA molecules. As mentioned before, RNA has a central role in protein synthesis, in fact a particular type of RNA, called messenger RNA, carries information from DNA to structures called ribosomes. These ribosomes, that are made from proteins and ribosomal RNAs, can read messenger RNAs and translate the information they carry into proteins. There are many types of RNAs with other roles and functions, in particular they can regulate the expression of some genes, and they are also present in the genomes of most viruses.

## 2.2 Gene and Alternative Splicing

From a molecular point of view, a *gene* is commonly defined as the entire nucleic acid sequence that is necessary for the synthesis of a functional polypeptide. According to this definition, in a gene there are not only the nucleotides that encode the amino acid

sequence of a protein (usually referred to as the *coding region*), but there are also all the DNA sequences required for the synthesis of a particular RNA transcripts. The concept of gene has evolved and has become more complex since it was first proposed. There are various definitions of this term, but all the initial descriptions were focused on the ability to determine a particular characteristic of an organism and also the heritability of this characteristic. A modern definition of a gene is *"a locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, transcribed regions, and or other functional sequence regions"* [4, 5].

On the other side, the computational biology community has adopted a "definition" of gene as a formal language to describe the gene structure as a grammar, by using a precise syntax of upstream regulation, exon, and introns. More specifically, a gene is composed of *exons* and *introns*. The formers, which are usually referred as coding parts, are the essential parts of the messenger RNA (mRNA), that is the template sequence for the protein, whereas introns, which are usually referred as non-coding parts, mostly have a regulatory role. Following the central dogma, the complete gene region is first transcribed into a precursor mRNA (pre-mRNA) molecule that consists of exons and introns. Transcription is the first stage of the expression of genes into proteins.

During the *transcription* process, some proteins, called transcription factors, bind in or near the promoter region that resides directly upstream of the gene. These proteins have the controlling role of the transcription. At this point, the *splicing* process is initiated by RNA binding proteins, called splicing factors, and the intron sequences of the pre-mRNA are removed. This is typical in eukaryotes, in which the messenger RNA is subject to this process, in order have the correct production of proteins through translation. The recognition of intron-exon boundaries is facilitated through the detection of short sequences called splice sites.

At the end of the transcription, a process, called polyadenylation, is involved. Polyadenylation consists in the addition of a poly-A tail, which is a series of Adenine nucleotides, to a RNA molecule. The process of polyadenylation is initiated by the binding of proteins to the polyadenylation site of an exon in the pre-mRNA. After the splicing and the polyadenylation of the pre-mRNA, the final mRNA is obtained. The poly-A tail added in the polyadenylation process is important for the transport of the mRNA to the cell cytoplasm and also for the control of the half-life of the mRNA.

By varying the exon composition of the same messenger RNA, the splicing process can produce a set of unique proteins. This process is called *alternative splicing* (AS) and is the mechanism by which a single pre-mRNA can produce different mRNA variants, by extending, shortening, skipping, or including exon, or retaining intron sequences. The most recent studies indicate that alternative splicing is a major mechanism generating

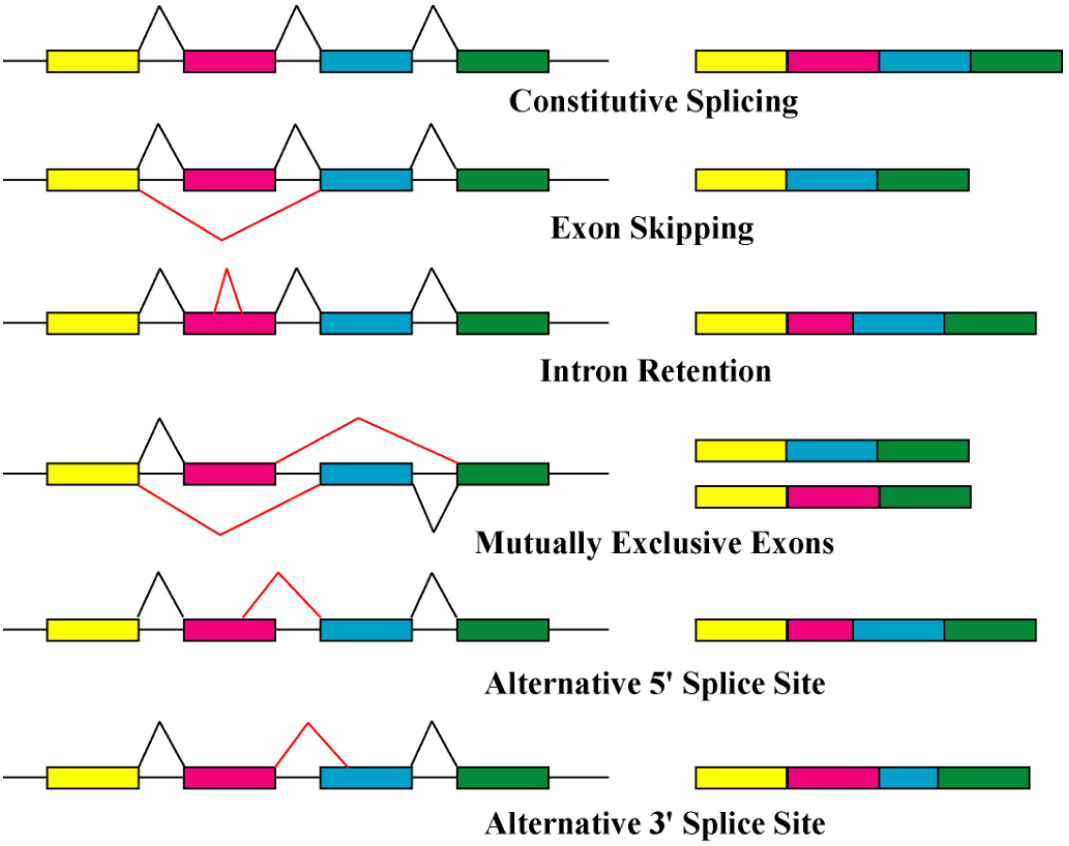



FIGURE 2.2: Basic types of alternative splicing events.

functional diversity in humans and vertebrates, as at least 90% of human genes exhibit splicing variants.

In a typical mRNA (which is composed of several exons), there are different ways in which the splicing pattern can be altered (see Figure 2.2). Most exons are constitutive and they are always spliced or included in the final mRNA. A regulated exon that is sometimes included and sometimes excluded from the mRNA is called a *skipped exon* (or *cassette exon*). In certain cases, multiple cassette exons are *mutually exclusive*, producing mRNAs that always include one of several possible exon choices but no more. Exons can also be lengthened or shortened by altering the position of one of their splice sites. One sees both *alternative* $5'$ and *alternative* $3'$ splice sites. The $5'$-terminal exons of an mRNA can be switched through the use of alternative promoters and alternative splicing. Similarly, the $3'$-terminal exons can be switched by combining alternative splicing with alternative polyadenylation sites. Finally, some important regulatory events are controlled by the failure in removing an intron, and such a splicing pattern is called *intron retention*.

The combination of these AS events generates a large variability at the post-transcriptional level, accounting for an organism's proteome complexity [6]. Changes in the splicing site

choice can have all manner of effects, especially on the encoded protein. For example small changes in the peptide sequence can alter ligand binding, enzymatic activity, allosteric regulation, or protein localization. In other genes, the synthesis of a whole polypeptide, or a large domain within it, can depend on a particular splicing pattern. Genetic switches, based on alternative splicing, are important in many cellular and developmental processes, including sex determination, apoptosis, axon guidance, cell excitation and contraction, and many others. In addition, many diseases (e.g. cancer) have been related to alterations in the splicing machinery, highlighting the relevance of AS to therapy.

## 2.3 Sequencing

Sequencing is the determination of the precise sequence of nucleotides in a molecule. Since the discovery of the structure of DNA by Watson and Crick in 1953 [7], an enormous effort has been done for decoding genome sequences of many organisms, including humans. If finding DNA composition was the discovery of the exact substance holding our genetic makeup information, DNA sequencing is the discovery of the process that will allow us to read that information.

### 2.3.1 Sanger Technique

Genomic sequencing began with the development, by Frederic Sanger, of the Sanger sequencing, in the 1970s. This method, also known as dideoxynucleotide or chain termination method [8], has been the only widely used technology for over three decades. In this technique, the DNA is used as a template to generate multiple copies of the same molecule that differ from each other in length, by a single base. The DNAs are then separated based on their size, and the bases at the end are identified, recreating the original DNA sequence (see Figure 2.3).

More specifically, in this method a single strained piece of DNA is amplified many times by the addition of dideoxy-nucleotides instead of deoxy-nucleotides. In fact, when the DNA is amplified, new deoxy-nucleotides (dNTPs) are usually added as the strand of DNA grows, but in the Sanger method this bases are replaced by special ones, called dideoxy-nucleotides (ddNTPs). There are two important differences that characterize the ddNTPs with respect to the similar dNTPs: they have fluorescent tags attached to them (a different tag for each of the 4 ddNTPs) and a crucial atom is missing, so that it prevents new bases from being added to a DNA strand, after that a ddNTP is added. This means that if during a DNA strand growth a ddNTP is inserted, the

synthesis process is stopped because no other nucleotides can be added after that, on the strand. If this amplification process is repeated for many cycles, as a result, all the possible lengths of DNA will be represented, and every piece of the synthesized DNA will contain a fluorescent label at its terminus (see Figure 2.3(a)).

At this point, by using a gel electrophoresis, the amplified DNA can be separated according to its size. In this way, the DNA is separated from the smallest to the largest, and while the fluorescent DNA reaches the bottom of the gel, a laser can pick up the fluorescence of each piece of DNA. Each ddNTP (ddA, ddC, ddG, ddT) emits a different fluorescent signal that can be recorded on a computer, indicating the presence of a specific ddNTP at the terminus. Since every possible size of DNA strand is present (each one with its terminating ddNTP), the DNA strand has a fluorescent ddNTP at every position, and this also means that every nucleotide in the strand can be determined. At this point a computer program can convert the data, read by the laser, into a coloured graph, showing the determined sequence (see Figure 2.3(b)).

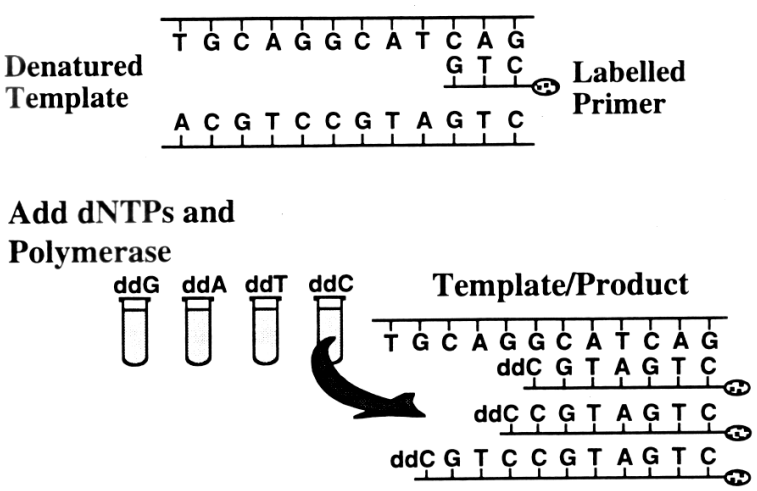
In the past, instead of using a computer, the resulting gel needed to be analyzed manually. This was a time consuming step, in fact the separation of the DNA strands by electrophoresis required the use of radioisotopes for labeling ddNTPs, one for each ddNTP (four different reactions). Today, thanks to all the improvements in fluorescent labels and in gel electrophoresis, DNA sequencing is fully automated (including the read out of the final sequence) and it is also faster and more accurate than before.

Sanger sequencing revolutionized the way in which the biology was studied, providing a way to learn the most basic genetic information. However, despite large efforts to improve the efficiency and the throughput of this technique, such as the development of automated capillary electrophoresis and computer driven assembly programs, the Sanger sequencing remained costly and time consuming, especially for applications involving the sequencing of whole genomes of organisms, such as humans.

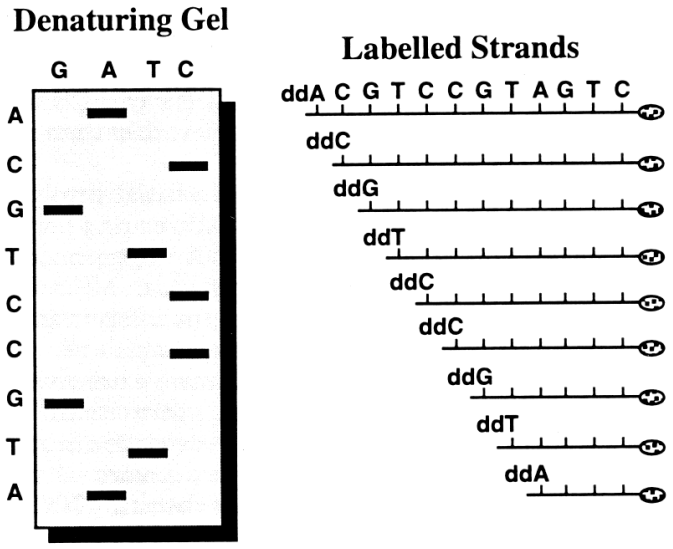
### 2.3.2 Next Generation Sequencing

In the past few years a “revolution” occurred in the field of sequencing. This was due to the development of a number of new high-throughput sequencing technologies, known as Next-Generation Sequencing (NGS) technologies, that are now widely adopted [9].

These technologies have fundamentally changed the way in which we think about genetic and genomic research, opening new perspectives and new directions that were not achievable or even thinkable with the Sanger method. They have provided the opportunity for a global investigation of multiple genomes and transcriptomes, in an extremely




(a) Polymerase reaction.




(b) Electrophoretic separation.

FIGURE 2.3: Sanger sequencing process. Figure 2.3(a) illustrates the products of the polymerase reaction, taking place in the ddCTP tube. The polymerase extends the labeled primer, randomly incorporating either a normal dCTP base or a modified ddCTP base. At every position where a ddCTP is inserted, the polymerization terminates; the final result is a population of fragments. The length of each fragment represents the relative distance from the modified base to the primer. Figure 2.3(b) shows the electrophoretic separation of the products of each of the four reaction tubes (ddG, ddA, ddT, and ddC), run in individual lines. The bands on the gel represent the respective fragments shown to the right. The complement of the original template (read from bottom to top) is given on the left margin of the sequencing gel.

efficient and timely manner at much lower costs, if compared with Sanger-based sequencing methods. One of the main advantages offered by NGS technologies is their ability to produce an incredible volume of data, cheaply, that in some cases exceeds one billion of short reads per instrument run. Applications that have already benefitted from these technologies, include: polymorphism discovery [10], non-coding RNA discover [11], large-scale chromatin immunoprecipitation [12], gene-expression profiling [13], mutation mapping and whole transcriptome analysis.

One of the common technological features that is shared among all the available NGS platforms is the massively parallel sequencing of DNA molecules (single or clonally amplified) that are spatially separated in a flow cell. In fact, the sequencing is performed by repeated cycles of polymerase-mediated nucleotide extensions or, in one format, by iterative cycles of oligonucleotide ligation.

In the following, the main available NGS systems will be described, analyzing technologies and features of the instruments used for the sequencing process [14]. The most diffuse platforms are the ones produced by *Roche/454 Life Sciences*, *Illumina* and *Life Technologies*. In addition to these ones, there are also two emerging technologies: *Ion Torrent* and *Pacific Biosciences*.

### Roche/454 Life Science

*454 Life Sciences*, founded in 2000 by Jonathan Rothberg, developed the first commercially available NGS platform, the GS 20, that was launched in 2005. This platform combined the single-molecule emulsion PCR with pyrosequencing, and was used by Margulies and colleagues to perform a shotgun sequencing of the entire 580,069bp of the Mycoplasma genitalia genome, resulting in a 96% coverage and 99.96% accuracy with a single GS 20 run. In 2007, *Roche Applied Science* acquired 454 Life Sciences, and introduced a second version of the 454 instrument, called GS FLX. In this new platform, that shared the same core technology of the GS 20, the flow cell is referred to as a "picotiter well" plate, which is made from a fused fiber-optic bundle.

The *library* of template DNAs used for sequencing is prepared by first cutting the molecule, and then end-repairing and ligating the obtained fragments (several hundred base pairs long) to adapter oligonucleotides. The library is then diluted to a single-molecule concentration, denatured, and hybridized to individual beads, containing sequences complementary to adapter oligonucleotides. By using an *emulsion PCR*, the beads are then compartmentalized into water-in-oil mixture, where the clonal expansions of single DNA molecules bound to the beads. The beads are deposited into individual picotiter-plate wells, and combined with sequencing enzymes (*PTP loading*).

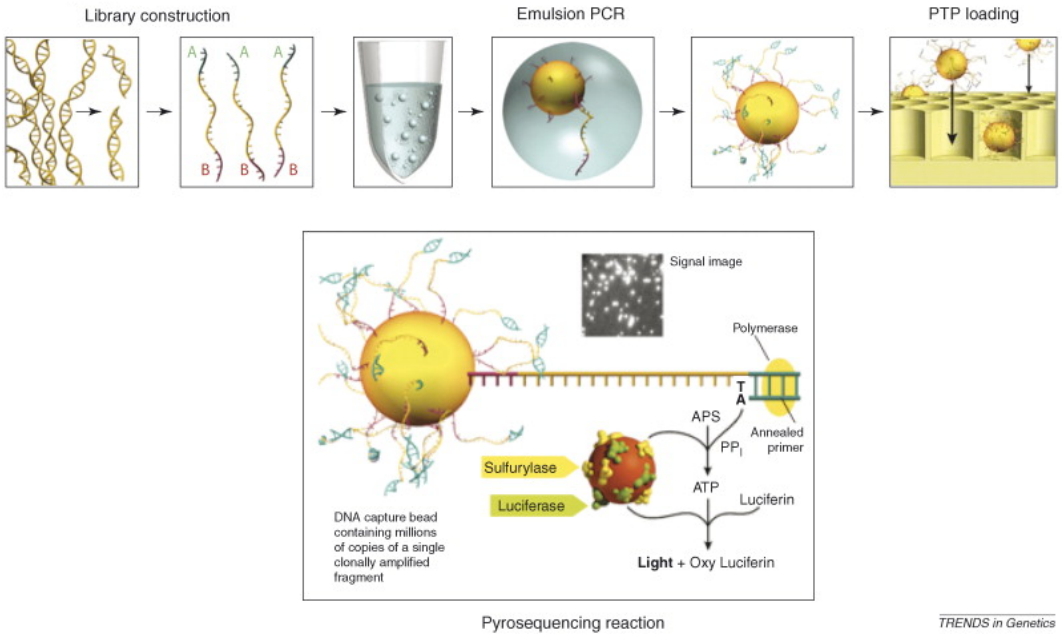



FIGURE 2.4: Roche 454 sequencing. In the library construction, the DNA is fragmented and ligated to adapter oligonucleotides, and then the fragments are clonally amplified by emulsion PCR. The beads are then loaded into picotiter-plate wells, where iterative pyrosequencing process is performed.

After that, the *pyrosequencing* is performed by successive flow additions of the 4 dNTPs: the four DNA nucleotides are added sequentially, in a fixed order across the picotiter-plate device, during a sequencing run. When a nucleotide flows, millions of copies of DNA bound to each of the beads and can be sequenced in parallel. In this way, each nucleotide complementary to the template strand is added into a well, and so the polymerase extends the existing DNA strand with one nucleotide. A camera records the light signal emitted by the addition of one (or more) nucleotide(s) (see Figure 2.4).

The two currently available systems by *Roche/454 Life Sciences* are the GS FLX+ and the GS Jr., which generate up to $1 \times 10^6$ and $1 \times 10^5$ reads per run, respectively. More specifically, there are two instruments using the first system (GS FLX+), that are: the GS FLX Titanium XLR70 and the newest GS FLX Titanium XL+. These two instruments produce reads of length 450 and 700 base pairs respectively, with a typical throughput of 450Mb in 10 hours for the first instrument, and 700Mb in 23 hours for the second, both maintaining an error rate $< 1\%$. On the other hand, the GS Jr. system produces reads of length $\sim 400$ base pairs, for a throughput of $\sim 35$Mb in 10 hours, with an error rate of $< 1\%$. Table 2.1 summarizes all the previous information on *Roche/454 Life Sciences* instruments.

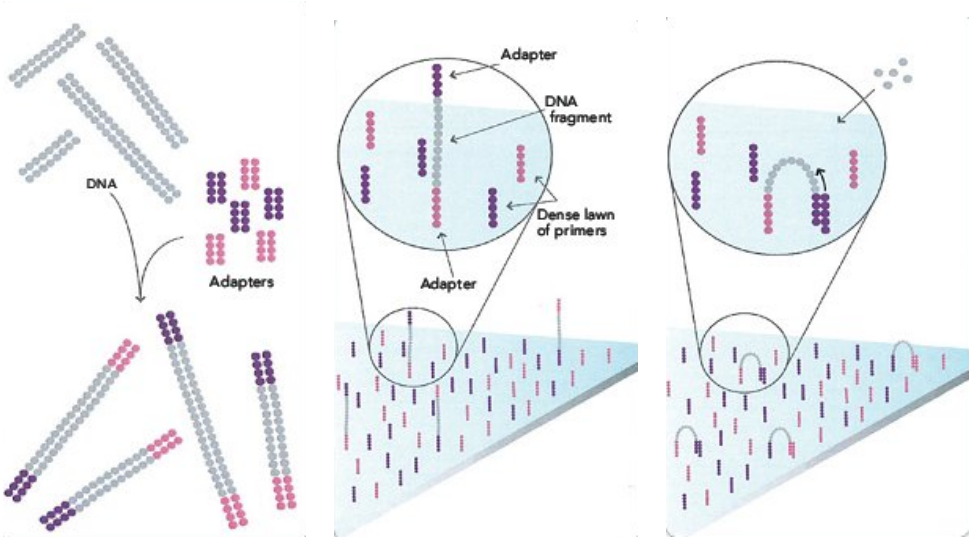
| **Instrument** | **Read length** (bp) | **Throughput** (Mb/run) | **Run time** (hours) |
|---|---|---|---|
| GS FLX Titanium XLR70 | 450 | 450 | 10 |
| GS FLX Titanium XL+ | 700 | 700 | 23 |
| GS Jr. system | $\sim 400$ | $\sim 35$ | 10 |

TABLE 2.1: Roche/454 Life Sciences NGS instruments.

### Illumina/Solexa

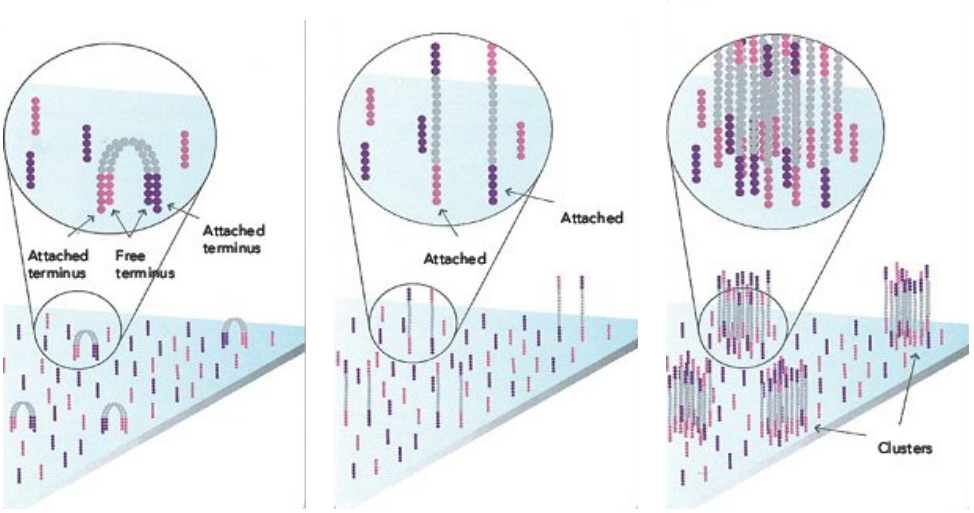
*Solexa* was founded in 1998 by two British chemists, Shankar Balasubramanian and David Klenerman, and the first short read sequencing platform, called Solexa Genome Analyzer, was launched in 2006. In the same year the company was acquired by *Illumina*.

The process of sequencing in the Genome Analyzer system is done by cutting the DNA molecule into small fragments (several hundred base pairs long) and, after that, oligonucleotide adapters are added to the ends of the these fragments to prepare them for the binding to the flow cell (see Figure 2.5(a)). This latter cell consists of an optically transparent slide with 8 individual lanes on the surface, on which oligonucleotide anchors are bound. When the single-stranded template DNAs are added to the flow cell, they are immobilized by hybridization to these anchors (see Figure 2.5(b)). Once the strand is attached to an anchor primer, it is "arched" over and hybridized to an adjacent anchor oligonucleotide (that has a complementary primer) by a "bridge" amplification in the flow cell (see Figure 2.5(c)). Starting from the primer on the surface, the DNA strand is replicated in order to create more copies (see Figure 2.6(a)), that are than denatured (see Figure 2.6(b)). At this point, starting from a single-strand DNA template, multiple amplification cycles are performed, in order to obtain a "cluster" (of about a thousand copies) of clonally amplified DNA templates (see Figure 2.6(c)). Before starting the sequencing process, the clusters are washed, leaving only the forward strands (to make the sequencing more efficient). To start the sequencing process, primers complementary to the adapter sequences, polymerase and a mixture of 4 differently colored fluorescent reversible dye terminators are added to the mix. In this way, the added primers are bound to the primers of the strand and the terminators are incorporated, according to sequence complementarity of each strand present in the clonal cluster (see Figure 2.7(a)). As bases are incorporated, a laser is used to activate the fluorescence (see Figure 2.7(b)) and the color is read by a computer, obtaining the sequence from many clusters (see Figure 2.7(c)). This iterative process, called *sequencing by synthesis*, takes more than 2 days to generate a read sequence, but, since there are millions of clusters in each flow cell, the overall output is $>$ 1 billion base pairs (Gb) per run.



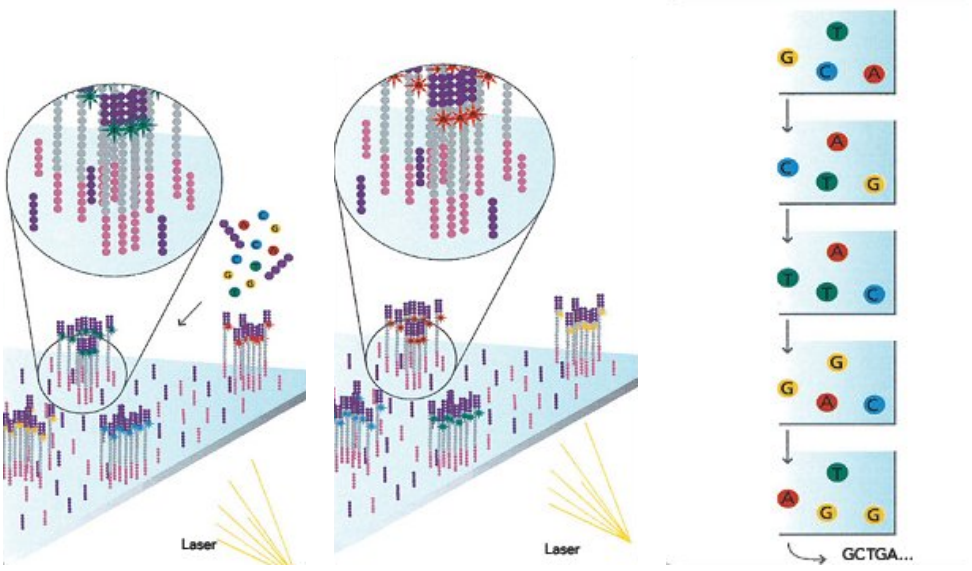

(a) DNA sample preparation. (b) Binding to the surface. (c) Bridge amplification.

FIGURE 2.5: Illumina sequencing 1/3. The DNA sample is fragmented and the adapters are bound to the ends (Figure 2.5(a)); the obtained single-strained fragments are attached to the flow cell surface (Figure 2.5(b)) and then they are "bridged" (Figure 2.5(c)).




(a) Double-strained fragments. (b) Denaturation. (c) Amplification.

FIGURE 2.6: Illumina sequencing 2/3. The fragments become double-strained (Figure 2.6(a)) and then they are denatured. This process leaves anchored single-strained fragments (Figure 2.6(b)). After a complete amplification, millions of clusters are formed on the flow cell surface (Figure 2.6(c)).




(a) First base determination. (b) Iterative sequencing process. (c) Sequence reading.

FIGURE 2.7: Illumina sequencing 3/3. The clusters are washed, leaving only the forward strands and, to initiate the sequencing process, primer, polymerase and a 4 colored terminators are added. The primers are bound to the primers of the strand and, after that, the first terminator is incorporated (Figure 2.7(a)). By iterating the incorporation process, all the sequence nucleotides are added and the laser activates the fluorescence (Figure 2.7(b)). During the incorporation a computer reads the image and determines the sequence of bases (Figure 2.7(c)).

| **Instrument** | **Read length** (bp) | **Throughput** (Gb/run) | **Run time** (days) |
|---|---|---|---|
| HiSeq 2000 | $2 \times 100$ | 600 | 11 |
| GAIIx | $2 \times 150$ | 95 | 14 |
| MiSeq | $2 \times 150$ | 2 | 1 |

TABLE 2.2: Illumina/Solexa NGS instruments.

The newest platforms can analyze higher cluster densities and have been improved in the sequencing chemistry allowing to obtain longer reads. Currently, there are 3 types of available instruments by *Illumina*, all producing *paired-end* read (see Section 2.4). More specifically, the HiSeq 2000 system can generate about 600 Gb of $2 \times 100$ base pair reads for run, taking an overall time of about 11 days. There are also other models of this instrument that have the possibility to perform "rapid runs", in order to obtain a smaller amount of reads in less time. Similarly to the previous instrument, the Genome Analyzer IIx (GAIIx) can produce up to 95 Gb of $2 \times 150$ base pair reads per single flow cell, taking about 14 days. Finally, the MiSeq instrument can produce reads of the same size of the previous one ($2 \times 150$ base pairs), but with a throughput of 2 Gb in about one day. The information about these instruments are summarized in Table 2.2.

### Life Technologies

*Life Technologies* was formed in 2008 by the merging of *Invitrogen* and *Applied Biosystem*; to be more precise, the former company bought the latter, and the two became Life Technologies. It was in 2007 that Applied Biosystem refined the technology of *SOLiD* (Supported Oligonucleotide Ligation and Detection) System 2.0 platform, previously developed in the laboratory of George Church, and released its first NGS system, the SOLiD instrument.

As for Roche/454 technology, the sample preparation consists of DNA fragments that are then ligated to oligonucleotide adapters; after that, the fragments are attached to beads and are clonally amplified by an emulsion PCR. Finally, beads with clonally amplified template are immobilized onto a derivitized-glass flow-cell surface, where the sequencing process starts. The major difference between SOLiD and the other NGS platforms is its *sequencing by ligation* (instead of sequencing by synthesis). The sequencing by ligation process starts by annealing a sequencing primer, complementary to the adapter at the "adapter–template" junction, and adding a mixture of all the 16 possible probes. In the specific, probes involved in this sequencing process are used to detect two adjacent bases (16 possible combinations), but there are only 4 different fluorescent dyes. Because of this, each base in the sequence is interrogated twice in order to obtain the correct original sequence (see Figure 2.9). Once the primer is annealed, the correct di-base probe, i.e. the one with the first two bases complementary to the first two bases of the template sequence, binds to this latter template and ligates to the primer (see Figure 2.8(1)). Each probe is an octamer, which consists of (3'-to-5' direction) 2 probe-specific bases followed by 6 degenerate bases (denoted as *nnnzzz*) with one of the 4 fluorescent labels linked to the 5' end. After the ligation of the probe, the fluorescence signal is recorded (see Figure 2.8(2)) and then the last 3 degenerated bases are cleaved and washed in order to obtain the new 5' phosphate group that is used to ligate the probe in the next cycle (see Figure 2.8(3)). To extend the first primer, seven cycles of ligation, referred to as a *round*, are performed (see Figure 2.8(4)) and then this synthesized strand is denatured. At this point, the new round is started, and a new sequencing primer, which is shifted of one base with respect to the previous one, is annealed. This latter has an offset of $(n-1)$ with respect to the original template sequence (see Figure 2.8(5)). The overall sequencing process is composed of five rounds, each of which is performed with a new primer and with successive offsets ($n-2$, $n-3$ and $n-4$) (see Figure 2.8(6)).

One of the main features of this sequencing process is that the perfect annealing of the probes is controlled by 2 bases of oligonucleotides. Thus, this method is usually more accurate and specific than the sequencing by synthesis approaches. Currently, there are 2 available NGS instruments by *Life Technologies*. The first, the SOLiD 4 system, can

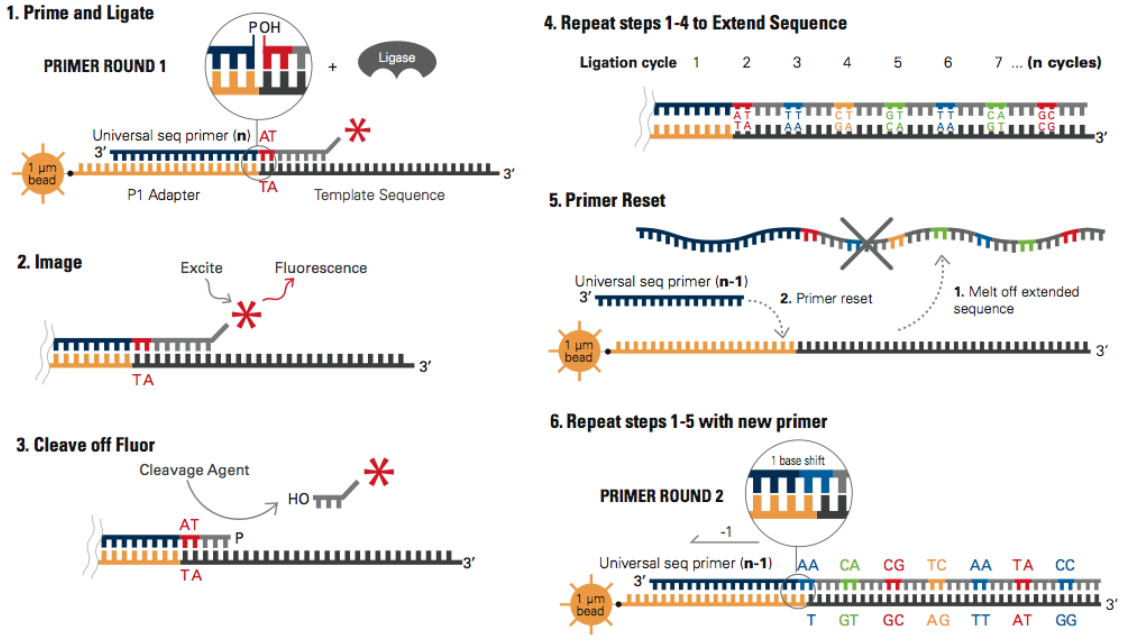



FIGURE 2.8: Sequencing by ligation. In (1) the primer is annealed to the adaptor and the first two bases of the probe are bound to the first two bases of the sequence. The fluorescence signal is read (2) and the last bases of the probe are cleaved, so that the new phosphate group is created (3). This ligation process is repeated for seven cycles (4), in order to extend the primer. Finally, this latter primer is melted off and a new one is annealed (5). The previous steps are repeated for all the new primers, with different offsets (6).

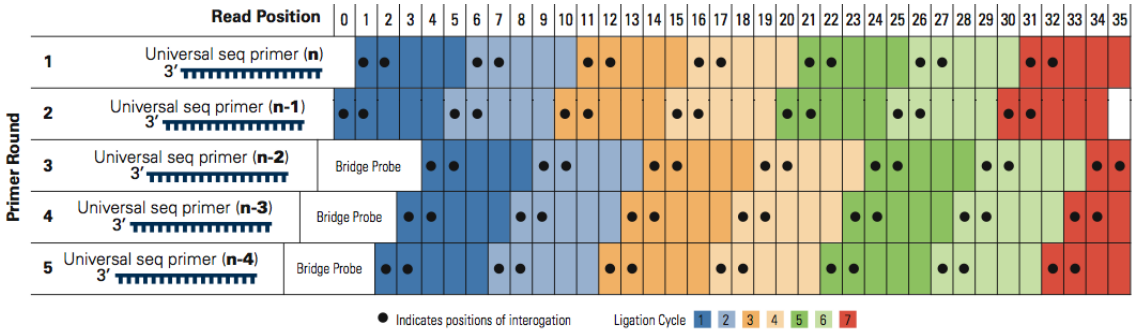



FIGURE 2.9: Summary of the sequencing by ligation process showing, for each base of the read sequence, the positions of the interrogations (2 for each position).

produce up to 100 Gb of $2 \times 50$ mate pair reads, in about 12 days. In a similar way, the second one, called SOLiD 5500xl system, can produce up to 180 Gb of $75 \times 35$ base pairs (paired-end) or $60 \times 60$ base pairs (mate-paired), in about 7 days. All the previous information on *Life Technologies* instruments are summarized in Table 2.3.

| **Instrument** | **Read length** (bp) | **Throughput** (Gb/run) | **Run time** (days) |
|---|---|---|---|
| SOLiD 4 | $2 \times 50$ | 100 | 12 |
| SOLiD 550xl | $75 \times 75$ (paired)<br>$60 \times 60$ (mate) | 180 | 7 |

TABLE 2.3: Life Technologies/SOLiD NGS instruments.

**Emerging Technologies**

In addition to the previously described methods, there are also some other new emerging technologies, in the sequencing field. In the following, two of them will be briefly presented. The first one is the so called *Ion Torrent* technology, in which the information coming from the chemical sequence is translated into digital form, without any fluorescence. In fact, this method relies on a well known biochemical process: in nature, when a nucleotide is incorporated into a strand of DNA by a polymerase, a hydrogen ion ($H^+$) is released as a byproduct. This hydrogen ion carries a charge that the system's ion sensor can detect. This sensor is composed of a high-density array micro-machined wells that allow to perform biochemical processes in a parallel manner. In this way, when a nucleotide is incorporated into a DNA strand, it releases a hydrogen ion which charges the $pH$ of the solution. This charge can be detected by the ion sensors and translated into digital form. Any nucleotide that is added to a DNA template is detected as a voltage change, and if a nucleotide is not a match for a particular template, no voltage change will be detected and no base will be called for that template. On the other hand, if there are two identical bases on the DNA strand, the voltage is doubled, and the chip records two identical base calls. Thanks to this new technology, it is possible to reach a throughput of 1 Gb in a few hours, depending on the used chip. Moreover, this technology is still under development in order to improve the read quality and increase the length of the produced sequences.

In addition to the previous one, *Pacific Biosciences* developed a single-molecule real-time (SMRT) sequencing system. This technology uses a progressive sequencing by synthesis in which the nucleotides, incorporated during the polymerase, have a fluorescent dye attached to the poly-phosphate chain (phospholinked-nucleotides). Such dyes are different for each of the 4 bases. The sequencing is performed on the SMRT cells, which contain thousand of zero-mode waveguides (ZMWs), providing a visualization chamber. In this way, when a base is incorporated, the fluorescent nucleotide is brought into the polymerase's active site and a high-resolution camera, in the ZMW, records this fluorescence signal. In addition to this, since the sequencing library is prepared in such a way that the resulting molecule is circular, it is possible to sequence the template using a scheme, called circular consensus sequencing, that improves the accuracy of the base calls, by

using the redundant sequencing information. This system gives the longest reads, in fact, the average read length per run is around 1.5 Kb. These, are two of the currently emerging technologies, that are continuously evolving in the field of sequencing, and are usually referred to as *Third Generation Sequencing* (TGS) technologies.

## 2.4 Data

NGS technologies have overcame the limitations of the previous sequencing methods by providing a new "kind" of data, suited for answering a wide range of biological questions. More specifically, one of the features that is shared among all the previously illustrated platforms, is that they produce a huge quantity of short sequences (*short reads*) that deeply cover the sequenced molecule. In the following, the main features of NGS technologies will be described.

The first, and probably the most relevant observation, is done by comparing the produced NGS reads to the ones of classical methods, such as Sanger sequencing, in terms of sequence length. In fact, depending on the adopted platform, the produced sequences can range from 35 base pairs of the Illumina machines, to several hundred base pairs of the Roche/454 systems, which are still shorter than the sequences, few thousand base pairs long, of the Sanger method. Another key point of this new generation of systems is the throughput, that is much higher than before. Due to such a massive amount of involved data, the management and the analysis of the data require dedicated software and also high-performance and capacity computing resources. This shift, from a low number of long reads to a huge number of short reads, can be also observed from a coverage point of view, which is the average number of reads that represent (cover) a given nucleotide, in the original sequence. In the classical methods, the typical coverage was 1x - 2x, meaning that each base was present in at most two reads. On the other hand, with the advent of the NGS technologies, it is now possible to reach a coverage of 10x or 100x of the given sequence, which correspond to an improvement of one or two orders of magnitude.

It is not easy to directly compare the different NGS platforms, especially from an error rate point of view. In fact, for example, most of the times the value reported by the companies is based on sequence reads of particular templates, that are favourable for their platform. However, in almost all the previously introduced instruments (except the one by Pacific Biosciences), the errors increase near the end of each platform's maximum read length. Indeed, maximum read length is limited by error tolerance. Although the error rate comparison is difficult for other reasons, in Table 2.4 are reported the "reasonable approximations" that can be used to compare different platforms [15].

| **Instrument** | **Primary Errors** | **Single-pass Error Rate (%)** | **Final Error Rate (%)** |
|---|---|---|---|
| 454 - all Models | Indel | 1 | 1 |
| Illumina - all models | Substitution | $\sim 0.1$ | $\sim 0.1$ |
| SOLiD 5500xl | A-T bias | $\sim 5$ | $\leqslant 0.1$ |
| Ion Torrent - all chips | Indel | $\sim 1$ | $\sim 1$ |
| Pacific Biosciences | CG Deletion | $\sim 15$ | $\leqslant 15$ |

TABLE 2.4: Error rate comparison of different sequencing platforms.

| **Phred Quality Score** | **Probability of Incorrect Base Call** | **Base Call Accuracy** |
|---|---|---|
| 10 | 1 in 10 | 90% |
| 20 | 1 in 100 | 99% |
| 30 | 1 in 1,000 | 99.9% |
| 40 | 1 in 10,000 | 99.99% |
| 50 | 1 in 100,000 | 99.999% |

TABLE 2.5: Quality score and base calling accuracy.

From Table 2.4, it is possible to notice that the error rate varies from 0.1% of the SOLiD 5500xl and Illumina systems, to 15% of the Pacific Biosciences one. These values are referred to the final error rate, which can be different from the one obtained in a single passage. This is the case of SOLiD, that has the second highest raw error rate ($\sim 5\%$), but, by allowing a double or triple encoding of each base (i.e. each base is sequenced independently two or three times, with inconsistent data becoming inaccessible to users), it achieves its low error rate ($\leqslant 0.1\%$). In a similar way, Illumina refers its error rate not on the entire data, reaching a $\sim 0.1\%$; also Pacific Biosciences suggests of reading each template multiple times, in order to overcome high error rates and to achieve a consensus sequence with a low error rate, but it gives users the option to obtain single-pass data. Finally, in addition to raw error rates of sequencing reads, each platform has also its own biases. To this purpose, each sequencing platform usually provides the *quality* of the produced data, by adding an extra information in the read file (or in a separated one), about the quality of each base present in the sequences. For each nucleotide, it is used the *Phred quality score* as a measure of reliability. This measure, denoted as $Q$, is logarithmically related to the base-calling error $P$:

$$Q = -10\ log_{10}P$$

where $Q$ is the phred quality score and $P$ is the probability that the base call is incorrect. This means that, the smaller the $P$, the higher the $Q$. From another point of view, $1-P$ is the probability of correctness of the base, so if $P = 0.01$, then the quality score $Q = 20$ and the probability that is correct is 0.99 (i.e. 99%) (see Table 2.5).

The "old way" to provide the information about the sequences and the associated quality values was to provide two separated files. The first one, in *FASTA* format, in which there were the sequences, and the other one, containing the phred values:

**FASTA Sequence file** :

```
>SEQ_ID
agtcTGATATGCTgtacCTATTATAATTCTAGGCGCTCATGCCCGCGGATATCGTAGCTA
```

**Quality file** :

```
>SEQ_ID
10 15 20 22 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50 50 50 20 50 50 50 14 50 50 50 11 20 25 25 30 30 20 15 20
35 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
```

In the specific, in a FASTA file there are two lines for each entry, i.e. for each read: the first one contains the read id (prefixed with >) and the second one contains the sequence of the read. The quality file adopts the same format, but instead of the nucleotides, the second line is a sequence of numeric values, one for each base of the read in the FASTA file, indicating the quality score. To have a more compact way to provide the information about read and quality scores, a new format has been adopted. This "new way", called *FASTQ*, incorporates both the information in a single file. In the specific, it is organized as:

**FASTQ file** :

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*(((***+))%%%++)(%%%%).1***(-+*(''))**55CCF>>>>>>CCCCCCC65
```

In the first line of each entry, the sequence identifier starts with the '@' symbol (instead of '>') and it is followed by the sequence in another line (as in the FASTA file). After that, there is a separator line started with a symbol (which usually is '+'), and finally, there is a line containing the encoded quality values (one symbol per nucleotide). The fourth line encodes the quality values of the sequence of nucleotides present in the second line, and obviously, it must contain the same number of symbols as the number of letters of the sequence. The quality encoding is done by using letters and symbols to represent numbers, and the range of adopted symbols depends on the used platform. The following example refers to the Illumina range of symbols, with associated quality scores from 0 to 41:

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J
|                   |                   |                   |                   |
Q0                 Q10                 Q20                 Q30                 Q40
```

For this reason, one of the first steps in an analysis pipeline that uses NGS data is the sequencing filter, based on the phred values, in order to discard the reads with poor qualities or to cut them at a certain length (read trimming).

As a final remark, all the NGS platforms, in addition to the "single" short read sequences, offer the possibility to produce *paired-end* reads, i.e. sequences that are derived from both the ends of the fragment, in the library. Obviously, there are some differences among read pairs, depending from the used platform. In particular, it is possible to distinguish between:

- *Paired-ends*, that are a linear fragment sequenced at both the ends in two separate reactions;
- *Mate-pairs*, that are a circularized fragment, usually $> 1$ Kb long, sequenced by a single reaction or by two separate end reads.

In general, paired-end reads offer some advantages, e.g. for the sequencing of large and complex genomes, because they can be more accurately placed (mapped), than the single ended short reads and so they can be used to disambiguate some complex situations. It is also possible to distinguish reads that are coming from cDNAs, that carry information about RNA molecules. In this case they are called *RNA-Seq* reads.

### 2.4.1 RNA-Seq

The RNA sequencing method, called RNA-Seq, is the use of the previously described technologies, for the sequencing RNA molecules [16]. This technique has revolutionized the way in which transcriptomes are analyzed, providing a new approach to map and quantify them. As said, by using NGS technologies, a population of RNAs is converted into a library of cDNA fragments by attaching adaptors to the ends of each sequence. These fragments are then sequenced, obtaining a set of short reads (or paired-ends reads).

RNA-Seq has opened new prospectives, like the possibility of identifying the positions of the transcription boundaries. In fact, novel transcribes regions and introns for existing genes can be annotated, and moreover, some of the aspects of the existing gene annotations can be modified and corrected. In addition to this, the information about

the connection of two or more exons can be obtained by using short and long reads. For these reasons, RNA-Seq is particularly suited for studying complex transcriptomes, to reveal sequence variations. Furthermore, from a quantitative point of view, RNA-Seq can be used to determine the expression levels of RNA molecules, in a more accurate way, than in the previous methods. Considering all the mentioned advantages, RNA-Seq has generated a global view of the transcriptome and its composition for a number of species (known or unknown) and has revealed new splicing isoforms of many genes.

The annotation of alternative splicing variants and events, to differentiate and compare organisms, is part of the central goal in transcriptome analysis of identifying and quantifying all full-length transcripts. RNA-Seq will help to reach this purpose. It can also be applied to the biomedical field, in order to compare normal and diseased tissues to understand the physiological changes between them. Another challenge, involving RNA-Seq, is its use to target complex transcriptomes, in order to identify rare RNA isoforms from the genes. In the near future, RNA-Seq is expected to replace microarrays for many applications involving the structure and the dynamic of transcriptomes.

## 2.5 Definitions

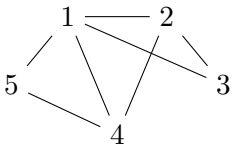
### 2.5.1 Strings

Let $s = s_1 s_2 \cdots s_{|s|}$ be a sequence of characters over the alphabet $\Sigma$, that is a *string*. The *length* of the string $s$ is denoted by $|s|$ and the *size* of $\Sigma$ is denoted by $|\Sigma|$. If considering DNA molecules, the typical alphabet is $\Sigma = \{A, C, G, T\}$, where $x \in \Sigma$ is a nucleotide and the length is measured in the *number of nucleotides* (nt), or *base pairs* (bp) when referring to the double stranded molecule. The $i^{th}$ character of a string $s$ is denoted by $s[i]$. Then $s[i : j]$ denotes the substring $s_i s_{i+1} \cdots s_j$ of $s$, while $s[: i]$ and $s[j :]$ denote respectively the *prefix* of $s$ consisting of $i$ symbols and the *suffix* of $s$ starting with the $j$-th symbol of $s$.

We denote with $\text{pre}(s, i)$ and $\text{suf}(s, i)$ respectively the prefix and the suffix of length $i$ of $s$. Among all prefixes and suffixes, we are especially interested into $\text{LH}(s) = \text{pre}(s, |s|/2)$ and $\text{RH}(s) = \text{suf}(s, |s|/2)$ which are called the *left half* and the *right half* of $s$.

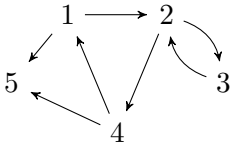
Given two strings $s_1$ and $s_2$, the *overlap* $ov(s_1, s_2)$ is the length of the longest suffix of $s_1$ that is also a prefix of $s_2$. The *fusion* of $s_1$ and $s_2$, denoted by $\varphi(s_1, s_2)$, is the string $s_1[: |s_1| - ov(s_1, s_2)]s_2$ obtained by concatenating $s_1$ and $s_2$ after removing from $s_1$ its longest suffix that is also a prefix of $s_2$. We extend the notion of fusion to a

$$G = ( \quad V = \{1,2,3,4,5\}, \quad E = \{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(4,5)\} \quad )$$




(a) Example of Undirected Graph.

$$G = ( \quad V = \{1,2,3,4,5\}, \quad E = \{(1,2),(1,5),(2,3),(2,4),(3,2),(4,1),(4,5)\} \quad )$$




(b) Example of Directed Graph.

FIGURE 2.10: Examples of graphs. Both the graphs have 5 nodes and 7 edges, but in Figure 2.10(a) the edges are unordered (i.e. $(u,v) = (v,u)$) and they are represented as simple lines. Instead in Figure 2.10(b) the edges are ordered (i.e. $(u,v) \neq (v,u)$) and they are represented as arrows indicating the direction.

sequence of strings $\langle s_1, \ldots, s_k \rangle$ as $\varphi(\langle s_1, \ldots, s_k \rangle) = \varphi(s_1, \varphi(\langle s_2, \ldots, s_k \rangle))$ if $k > 2$, and $\varphi(\langle s_1, s_2 \rangle) = \varphi(s_1, s_2)$.

### 2.5.2 Graphs and Trees

#### Graphs

A *graph* $G$ is an ordered pair $(V, E)$, where $V$ is a finite set called the *vertex set* and $E$ is called the *edge set*. Given two vertexes (or nodes) $u, v \in V$, an edge (or arc) is a pair $(u, v) \in E$. An edge $e = (u, v)$ is *directed* if one endpoint is the "head" and the other the "tail" (making $e$ *ordered*, i.e. $(u, v) \neq (v, u)$). In such a case the graph $G$ is called *directed graph* (or *digraph*). Otherwise, if there is no direction in the edges, the graph is called *undirected graph* (see Figure 2.10 for an example of graphs). If $(u, v)$ is an edge in a directed graph $G = (V, E)$, $e$ is *incident from* (or *outgoing*) vertex $u$ and is *incident to* (or *ingoing*) vertex $v$. If $(u, v)$ is an edge in an undirected graph $G = (V, E)$, $e$ is *incident* to vertexes $u$ and $v$. Given a vertex $v \in V$, the *indegree* of $v$ is the number of its ingoing edges and the *outdegree* is the number of its outgoing edges. In a directed graph the *degree* of a vertex is the sum of its indegree and its outdegree edges, instead in an undirected graph the degree of a vertex is the number of edges incident on it.

A *path* (or *walk*) of length $k$ from a vertex $u$ to a vertex $u'$ in a graph $G = (V, E)$ is a sequence $\langle v_0, v_1, \ldots, v_k \rangle$ of vertexes such that $u = v_0$, $u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. The length of the path is the number of edges in the path. A path forms a *cycle* if its endpoints $u$ and $u'$ are the same, i.e. $u = u'$. A graph is called *acyclic* if it contains no cycles. A graph that is directed and acyclic is called *directed acyclic*

*graph* (DAG). A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a set $V' \subseteq V$, the subgraph induced by $V'$ is the graph $G' = (V', E')$, where $E' = \{(u, v) \in E : u, v \in V'\}$. An undirected graph is *connected* if every pair of vertexes is connected by a path, and a *connected component* is a maximal connected subgraph of the graph. A connected graph has one connected component.

An interesting and well studied type of graphs are the so called *De Bruijn graphs*. This latter graphs were initially developed to find the shortest cyclic sequence of letters from an alphabet, in which every possible string of length $k$ appears as a substring in the cyclic sequence. To construct such a graph, given an alphabet $\Sigma$, all possible $(k-1)$-mers (substrings of length $k-1$) are assigned to nodes, and there exists a direct edge between two $(k-1)$-mers, $x$ and $y$, if there is some $k$-mer whose prefix is $x$ and whose suffix is $y$. In this way, the $k$-mers are represented by the edges of the graph. All the possible substrings of length $k$ of an alphabet $\Sigma$ are $|\Sigma|^k$. For example, if dealing with the DNA, the usual alphabet is $\Sigma = \{A, C, G, T\}$ and so all the possible substrings of length $k$ are $4^k$. In the binary alphabet (i.e. $\Sigma = \{0, 1\}$) all the possible substrings of length 3 are ($2^3 = 8$): 000, 001, 010, 011, 100, 101, 110, 111. The corresponding *De Bruijn graph* with $k = 4$, that has all the previous substrings as nodes, is shown in Figure 2.11. Each $k$-mer can be obtained by taking the $(k-1)$-mer at the starting node and overlapping the $(k-1)$-mer at the ending node. It must be also noticed that two consecutive nodes (i.e. nodes that are connected by an edge) have an overlap of $k-2$ (i.e. the suffix of length $k-2$ of the first one is equal to the prefix of length $k-2$ of the second one).

This type of graphs have been recently used for the assembly of sequences from NGS data in order to provide an efficient approach to solve that computational problem. One of the differences, with respect to the formulation of the problem for which the *De Bruijn graphs* were initially developed, is that, when dealing with short reads, not all the possible substrings of length $k$ of a given alphabet are present. The advantage of using *De Bruijn graphs* is that the sequence can be reconstructed by an Eulerian cycle (in the example in Figure 2.11 the shortest circular sequence, that contains all the 4-mers as substrings, is 0000111101100101).

### Trees

A (free) *tree* is a connected, acyclic, undirected graph. Let $G = (V, E)$ be an undirected graph. The following statements are equivalent.

1. $G$ is a (free) tree.

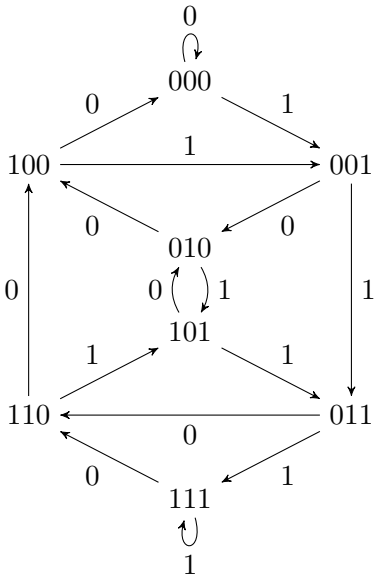



FIGURE 2.11: Example of *De Bruijn graph* on the binary alphabet, with $k = 4$. Nodes represent all the possible substrings of length $k-1$ and there is an edge between two nodes if there is a $k$-mer that has the first $(k-1)$-mer as prefix and the second $(k-1)$-mer as suffix. The $k$-mer is obtained by the fusion of the $(k-1)$-mers at the two ends. In this example edges are labeled by the symbols that must be concatenated to the first $(k-1)$-mer, to obtain the $k$-mer.

2. Any two vertexes in $G$ are connected by a unique path.
3. $G$ is connected, but if any edge is removed from $E$, the resulting graph is disconnected.
4. $G$ is connected, and $|E| = |V| - 1$.
5. $G$ is acyclic, and $|E| = |V| - 1$.
6. $G$ is acyclic, but if any edge is added to $E$, the resulting graph contains a cycle.

A *rooted tree* is a (free) tree in which there is a distinguished vertex called *root*. Let $x$ be a node in a rooted tree $T$ with root $r$. Any node $y$ on the unique path from $r$ to $x$ is called an *ancestor* of $x$ (and *proper ancestor* if $x \neq y$). If $y$ is an ancestor of $x$, then $x$ is a *descendant* of $y$ (and *proper descendant* if $y$ is a proper ancestor of $x$). The *subtree* of $T$, rooted at $x$, is the tree induced by descendants of $x$, rooted at $x$, and it is referred to as $T_x$.

If the last edge on a path from the root $r$ of a tree $T$ to a node $x$ is $(y, x)$, then $y$ is the *parent* of $x$, and $y$ is a *child* of $y$. The root is the only node in $T$ with no parent. If two nodes have the same parent, they are *siblings*. A node with no children is a *leaf*. A node that is not a leaf is an *internal node* (see Figure 2.12 for an example). In a rooted tree $T$, the number of children of a node $x$ is called the *degree* of $x$ in $T$. The

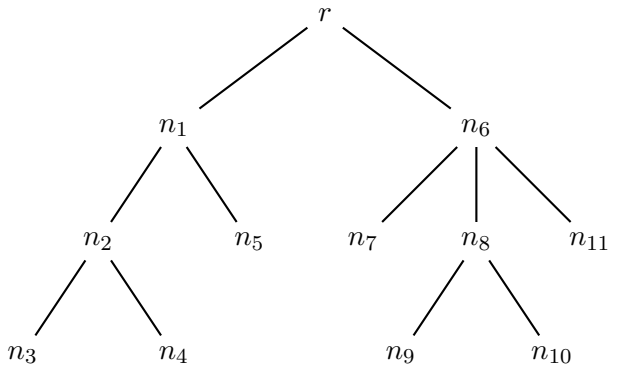


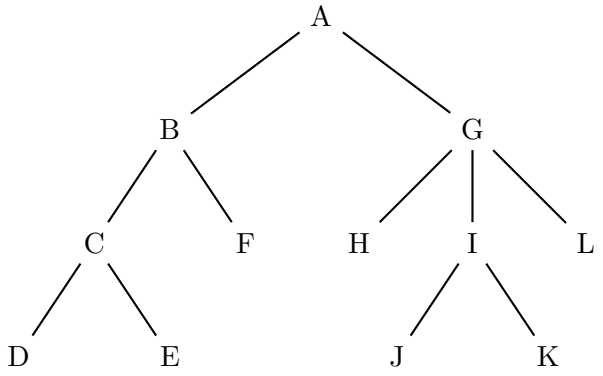

FIGURE 2.12: Example of rooted tree. The node $r$ is the root of the tree, the nodes $n_3$, $n_4$, $n_5$, $n_7$, $n_9$, $n_{10}$ and $n_{11}$ are the leaves and $n_1$, $n_2$, $n_6$ and $n_8$ are the internal nodes. As an example, $n_6$ is the parent of $n_8$ and $n_2$ is one of the children of $n_1$ (the other one is $n_5$).

length of the path from the root $r$ to a node $x$ is the *depth* of $x$ in $T$. The largest depth of any node in $T$ is the *height* of $T$. An *ordered tree* is a rooted tree in which the children of each node are ordered, i.e. if a node has $k$ children, then there is the first child,..., the $k$-th child. In general, rooted trees are referred as $n$-ary trees, in which $n$ represents the maximum number of children of each node. From this point of view, one of the most studied type of trees are the *binary trees*, in which every node has at most 2 children. Given a tree $T$, it is also possible to identify its set of nodes as $N(T)$ and its set of edges as $E(T)$. In addition to this, $L(T)$ represent the set of all the leaves, i.e. $L(T) = \{n \in N(T) : n \text{ is a leaf}\}$, while $I(T)$ denotes its set of internal nodes, i.e. $I(T) = \{n \in N(T) \setminus L(T)\}$.

Given a rooted tree $T$ with root $r$ and two nodes $x$ and $y$, the *Lowest Common Ancestor* (LCA) of $x$ and $y$, denoted as $LCA(x, y)$, is the lowest node in the tree (the farthest from the root $r$) that has both $x$ and $y$ as descendants.

There are several algorithms to traverse a rooted tree and the difference among them is the order in which they visit the nodes. In particular, it is possible two distinguish between two main approaches: *breadth-first* traversal and *depth-first* traversal. In the breadth-first traversal of a rooted tree $T$ with root $r$, the nodes are visited according to their depth in $T$. The first visited node is the root $r$ (which is at zero depth), then all the nodes at depth one (i.e. the children of $r$) are visited, then the nodes at depth two (i.e. the children of the children of $r$) are visited, and so on until the leaves are reached. On the other hand, the depth-first traversal can be performed in *preorder* or in *postorder*. In the former, starting from the root, a node is visited and then, for all the subtrees rooted at each of its children, a preorder visit is performed. The recursive definition is the following:

1. Visit the root; and then




Breadth-first: A, B, G, C, F, H, I, L, D, E, J, K

Depth-first (preorder): A, B, C, D, E, F, G, H, I, J, K, L

Depth-first (postorder): D, E, C, F, B, H, J, K, I, L, G, A

FIGURE 2.13: Example of tree traversals. Given the tree, this example shows the order in which the nodes are visited, for each of the three types of traversal.

2. do a preorder traversal at each of the subtrees of the root.

Indeed, in a postorder traversal, the node is visited last, and before doing that, a postorder traversal is performed for the subtrees rooted at each of the children of the considered node. The recursive definition is the following:

1. Do a postorder traversal each of the subtrees of the root; and then
2. visit the root.

In this latter case the root is visited after all the other nodes. For an example of the three described traversals, see Figure 2.13.

# Chapter 3

# Reconstructing Isoform Graphs from RNA-Seq data

Next-generation sequencing (NGS) technologies allow new methodologies for alternative splicing (AS) analysis. Current computational methods for AS from NGS data are mainly focused on predicting splice site junctions or de novo assembly of full-length transcripts. These methods are computationally expensive and produce a huge number of full-length transcripts or splice junctions, spanning the whole genome of organisms. Thus summarizing such data into the different gene structures and AS events of the expressed genes is a hard task.

To face this issue, in this work we investigate the computational problem of reconstructing from NGS data, in absence of the genome, the structure for each gene, which is represented by the *isoform graph*: we introduce such graph and we show that it uniquely summarizes the gene transcripts. We define the computational problem of reconstructing the isoform graph from a set of RNA-Seq reads and we provide some conditions that must be met to allow such reconstruction. Finally, we describe an efficient algorithmic approach to solve this latter problem, validating our approach with both a theoretical and an experimental analysis.

Part of the work presented in this chapter has been presented in international venues, including the ISMB/ECCB - 8th Special Interest Group meeting on Alternative Splicing (AS-SIG) in Vienna, and the Next Generation Sequencing Workshop in Bari. Moreover, it has been published in [2].

## 3.1 Introduction

Challenging tasks of transcriptome analysis via RNA-Seq data analysis [17, 18] are reconstructing full-length transcripts (or isoforms) of genes and their expression levels. Moreover, the annotation of alternative splicing variants and AS events, to differentiate and compare organisms, is part of the central goal in transcriptome analysis of identifying and quantifying all full-length transcripts. However, the extraction of splicing variants or significant AS events from the different transcripts produced by a set of genes requires to compare hundreds of thousands of full-length transcripts. Graph representations of splice variants, such as splicing graphs [19], have emerged as a convenient approach to summarize several transcripts from a gene into the putative gene structure they support. The current notions of splicing graphs rely on some sort of gene annotations, such as the annotation of full-length transcripts by their constitutive exons on the genome.

In this work, we first define the notion of *isoform graph* which is a gene structure representation of genome annotated full-length transcripts of a gene. The isoform graph is a variant of the notion of splicing graph that has been originally introduced in [20] in a slightly different setting. When using only RNA-Seq data, the genome annotation cannot be given, and thus it is quite natural to investigate and characterize the notion of splicing graph which naturally arises when a reference genome is not known or available. Thus, in the work we focus on the following main question: *under which conditions the reconstruction of a gene structure can be efficiently accomplished using only information provided by RNA-Seq data?*

In order to face this problem, we give some necessary or sufficient conditions to infer the isoform graph, we introduce an optimization problem that guides towards finding a good approximation of the isoform graph and finally we describe an efficient heuristic for our problem on data violating the conditions necessary to exactly infer the isoform graph.

The novelty of our approach relies on the fact that it allows the reconstruction of the splicing graph in absence of the genome, and thus it is applicable also to data where the genomic sequence is not available, or is highly fragmented or altered data, as found in cancer genomes. Moreover we focus on methods that can effectively be used for a genome-wide analysis on a standard workstation. Such goal implies that we have to avoid computing the complete assembly of full-length transcripts or listing the set of splice site junctions, as those tasks are usually computationally intensive when performed on the whole transcriptome, i.e. when the input data consists of RNA-Seq sampled from the transcripts of several genes.

In the work, we aim to advance towards the understanding of the possibilities and limitations of computing the distinct gene structures from which genome-wide RNA-Seq or short reads data have been extracted. In this sense our approach aims to keep separate gene structures in the reconstruction from genome wide RNA-Seq, even in absence of a reference.

Our proposed approach is validated from both theoretical and experimental points of view. First we will prove that some conditions must be met in order to guarantee the correct reconstruction of the isoform graph from RNA-Seq data. Then we describe a simple and efficient algorithm that reconstructs the isoform graph under some more restrictive conditions. At the same time, a more refined algorithm (sharing the main ideas of the basic one) is able to handle instances where the aforementioned conditions do not hold due to, for example, errors in the reads or lower coverage that typically affect real data.

We show experimentally, as well as theoretically, the scalability of our implementation to huge quantity of data. In fact limiting the time and space computational resources used by our algorithm is a main aim of ours, when compared to other tools of transcriptome analysis. More precisely, our algorithmic approach works in time that is linear in the number of reads, while having space requirements bounded by the size of hashing tables used to memorize reads.

Moreover, we show that our method is able to distinguish among different gene structures though processing a set of reads from various genes, limiting the process of fusion of graphs structures from distinct genes due to the presence of repeated sequences.

The theoretical and experimental analysis have pointed out limitations that are inherent the input data. As such, we plan to further improve our algorithms and our implementations to be able to deal with the different situations coming from these limitations.

It must be underlined that our computational approach to AS is different from current methods of transcriptome analysis that focus on using RNA-Seq data for reconstructing the set of transcripts (or isoforms) expressed by a gene, and estimating their abundance. In fact, we aim on summarizing genome-wide RNA-Seq data into graphs, each representing an expressed gene and the alternative splicing events occurring in the specific processed sample. On the contrary, current tools do not give a concise result, such as a structure for each gene, nor they provide a easy-to-understand list of AS events for a gene.

Observe that, in absence of a reference, the information given by the predicted full-length transcripts spanning the whole genes does not imply a direct procedure to built the isoform graph. In fact, the annotation against a genome is required to compute such

a graph, as we need to determine from which gene a given transcript is originated, may lead to a complex and time consuming procedure.

As said before, our work is not focused on full-length transcripts reconstruction, such as Cufflinks [21], and Scripture [22] or de novo assembly methods that build a *De Brujin* graph such as TransAbyss [23], Velvet [24], and Trinity [25]. These tools are computationally expensive and are able to find only the majority of the annotated isoforms while providing a large amount of non annotated full-length transcripts that would need to be experimentally validated.

### 3.1.1 Transcriptome Reconstruction

The advent of RNA-Seq methods has opened new opportunities in the field of trascriptomics (as underlined in Section 2.4.1). For this reason, in the recent years, new computational challenges have emerged and new solutions have been proposed in order to tackle these recent problems. In particular, it is possible to identify three main "categories" of addressed computational challenges [26]: read mapping, transcriptome reconstruction and expression quantification. Although these different analysis can be performed singularly, RNA-Seq data studies often require to use methods from all three categories, in fact, the built pipelines, for RNA-Seq based analysis, combine the output of some programs as input of others.

More specifically, we mainly focus on the transcriptome reconstruction, describing the different adopted approaches. The reconstruction of the transcriptomes from RNA-Seq data is done by assembling the reads or their alignments, depending on the adopted strategy, into contigs. There are some main factors that make this task much difficult. First of all, since different transcripts have different expression levels, it can happen that for the less expressed ones there are only few a reads coming from them, making the reconstruction hard. Moreover, due to the fact that reads have short length, it is not easy to determine where each sequence is coming from, also because a gene can have very similar transcripts. To overcome these issues, there have been developed some tools that falls into two main categories, depending on the fact that they use or not a reference sequence (usually the genome) to guide the assembly.

#### Genome-guided Reconstruction

As pointed out in [27], the methods for the *genome-guided* reconstruction (also called *reference based* or *ab initio* assembly) involve the use of a splice aligner, in fact the introduction of RNA-Seq data has opened new issues, like the possibility that a sequence

(read) spans a splice junction. When using RNA-Seq sequences it is possible to perform alignments against a transcriptome but also a genome. In the first case (referred to as *unspliced alignment*), the whole sequence is searched into the reference, and this is the same as mapping a read coming from a DNA molecule into a genome. On the contrary, in the second case (referred to as *spliced alignment*), the read can be split and the obtained substrings can be aligned to different positions in the reference. This is for example the case of reads that span a splice junction between two exons that are separated by an intron in the genome. For this reason it is necessary to introduce a *gap* in the alignment. It must be clarified that also the unspliced aligners can introduce small gaps in order to deal with deletions or insertions few (usually one or two) bases long.

In particular, the unspliced aligners are usually based on the *seed* technique or on the *Burrows-Wheeler transform*, which is used to index the reference sequence. The advantage in using this transform is that it is particularly efficient for searching for exact matches in fact, by using some optimized implementation, an alignment against the human genome can be performed on a standard laptop. Finally, by performing a backtracking it is possible to deal with errors, but in this case the performance decreases exponentially. Two examples of programs that use the BWT are Bowtie [28] and BWA [29]. Indeed, seed based methods can be useful when for example the original transcriptome is not available and so the mapping must be done against the genome or a distant transcriptome (from another species). On the other hand, spliced aligners are capable of align reads with long gaps. Most of the used software are based on two main approaches, called *exon-first* and *seed and extend*. The former method is a two steps process, in which the reads are first aligned using an unspliced aligner, in order to identify the ones that are "entirely contained" into an exon and then the remaining ones are split into substrings and mapped singly. The assumption is that reads including one or more splice junctions require a gapped alignment. Example of programs that adopt this approach are MapSplice [30], SpliceMap [31] and TopHat [32]. In the second method, called "seed and extend", reads are cut into small substrings that are aligned into the genome. When a candidate match is found, the mapping is extended (using for example the Smith-Waterman algorithm) until the splice junction is correctly identified and the exact spliced alignment of the read is found. GSNAP [33] and QPALMA [34] are two software that use this approach. In general, exon-first methods are faster than the others and require less computational resources but, in contrast, seed and extend methods perform better than exon-first approaches when most of the mapping reads are on splice junctions.

As we said, for the genome-guided reconstruction, reads are usually aligned against the reference genome to obtain the mapping across introns. After that, by using the overlapping reads in each position, a graph is built and finally, all the isoform are composed

by a graph traversal. There are different strategies used to build and traverse the graph representing the isoforms. In particular, Scripture [22] builds a "connectivity graph" in which two bases in the genome are connected if they are immediate neighbors either in the genomic sequence itself or within a spliced read. At this point, the transcripts are created by scoring each path in the graph based on a statistical segmentation approach in order to keep only the relevant ones. This is why Scripture may produce a larger set of transcripts for each locus. In a similar way, Cufflinks [21] generates an "overlap graph" in which the spliced reads (split into fragments) are divided into non-overlapping loci, and each locus is assembled independently from the others. To do this, the fragments that are compatible and that overlap in the genome (based on the position) are connected, so that each fragment has one node in the graph and there is an edge between every pair of compatible fragments. In this graph, each path corresponds to a set of mutually compatible fragments that can be contracted to a single isoform. In this way, Cufflinks finds the minimum path cover of the overlap graph. This idea is based on Dilworth's Theorem which states that the number of mutually incompatible reads is the same as the minimum number of transcripts needed to "explain" all the fragments. Cufflinks implements a proof of Dilworth's Theorem that produces a minimal set of paths that cover all the fragments in the overlap graph, by finding the largest set of reads with the property that not two of them could have originated from the same isoform. Since it is possible to have more than one minimal set of isoforms, Cufflinks uses a statistical model on the coverage of the paths by weighting the edges in the graph. Both the previous programs can be efficiently executed on a standard laptop and guarantee good performances, also in presence of low expressed transcripts.

#### Genome-Independent Reconstruction

The second category of transcriptome assembly methods is called *genome-independent* (also referred to as *de novo* strategy). The difference, with respect to the previously described approaches, is that in this case the reads are directly assembled to build the transcripts. To do this, (almost all) the genome independent methods use the *De Bruijn graphs*, which give an efficient way to represent a sequence in terms of all its possible substrings of length $k$, called $k$-mers (see Section 2.5.2). In the specific, given a sequence $S$ and an integer $k$, $S$ is split into overlapping $k$-mers (all its substrings of length $k$) and the *De Bruijn graph* is built with $k$-mers as nodes and there is an edge between two $k$-mers if they have an overlap of $k-1$ (i.e. they are consecutive in the string $S$). In this way, there exists an edge between two nodes $x$ and $y$, if $x$ (after removing its first digit) is prefix of $y$, and $y$ (after removing its last digit) is suffix of $x$. Here, there is an example of *De Bruijn graph* with $k = 4$:

$$S : \text{acctgacgtaac}$$
$$\Downarrow$$
*De Bruijn graph*:
$$\text{acct} \rightarrow \text{cctg} \rightarrow \text{ctga} \rightarrow \text{tgac} \rightarrow \text{gacg} \rightarrow \text{acgt} \rightarrow \text{cgta} \rightarrow \text{gtaa} \rightarrow \text{taac}$$

A key point of the *De Bruijn graphs* is that, to build the original sequence, it is necessary to find an *Eulerian path* in the graph (i.e. a path that visits every edge exactly once). By using this property, it is possible to use the *De Bruijn graphs* to assemble (overlapping) sequences in an efficient way, making the choice of $k$ crucial. There are also some problems in the use of these graphs for the assembly of sequences, like for example sequencing errors that generate spurious branches or cycles. Another problem in finding the "assembled sequence" is that it can happen that an Eulerian path is not possible and so a path that visits at least every edge once is chosen. A common strategy, used by the genome independent assemblers, is to build a *De Bruijn graph* from all the reads by splitting the reads into $k$-mers and then contracting all the linear paths. After that, the transcripts are composed by traversing this collapsed graph, as shown in Figure 3.1 from [27].

TransABySS [23] uses the previously described procedure and, in order to prune the spurious branches, it uses the paired-end sequences. More specifically, by using the empirical distribution (coverage), branches that are not supported by reads and paired-ends are deleted. In a similar way, Velvet [24] performs a series of operations to remove errors in the graph. It must be underlined that Velvet is a "generic" *de novo* assembler and it is not specifically designed for transcriptome reconstruction. Finally, a recently developed tool, called Trinity [25], that uses a three steps process based on *De Bruijn graphs*, is becoming one of the most used *de novo* assembly programs. In the first step of the Trinity software, named *Inchworm*, a dictionary with all the $k$-mers extracted from the reads is constructed, filtering out the erroneous and the less frequent ones (e.g. singletons). After this, contigs are assembled in a greedy way: starting from the most frequent $k$-mer, the contig is extended, in both the directions, by extracting (and deleting from the dictionary) the most frequent $k$-mers, which has a $k-1$ overlap. The extension process is repeated until a $k-1$ overlapping $k$-mer exists. New contigs are created until there are available $k$-mers in the dictionary. As pointed out by the authors, this approach is much more efficient than computing a full graph from all reads at once, and it quickly provides a meaningful intermediate output of the contigs that are strongly supported by many $k$-mers in the reads. In the second step, called *Chrysalis*, the assembled contigs are partitioned into subsets that are likely to be derived from alternative splice transcripts. These contigs are grouped if there is an overlap of $k-1$ bases among them and if there is a minimal number of reads that span the junction

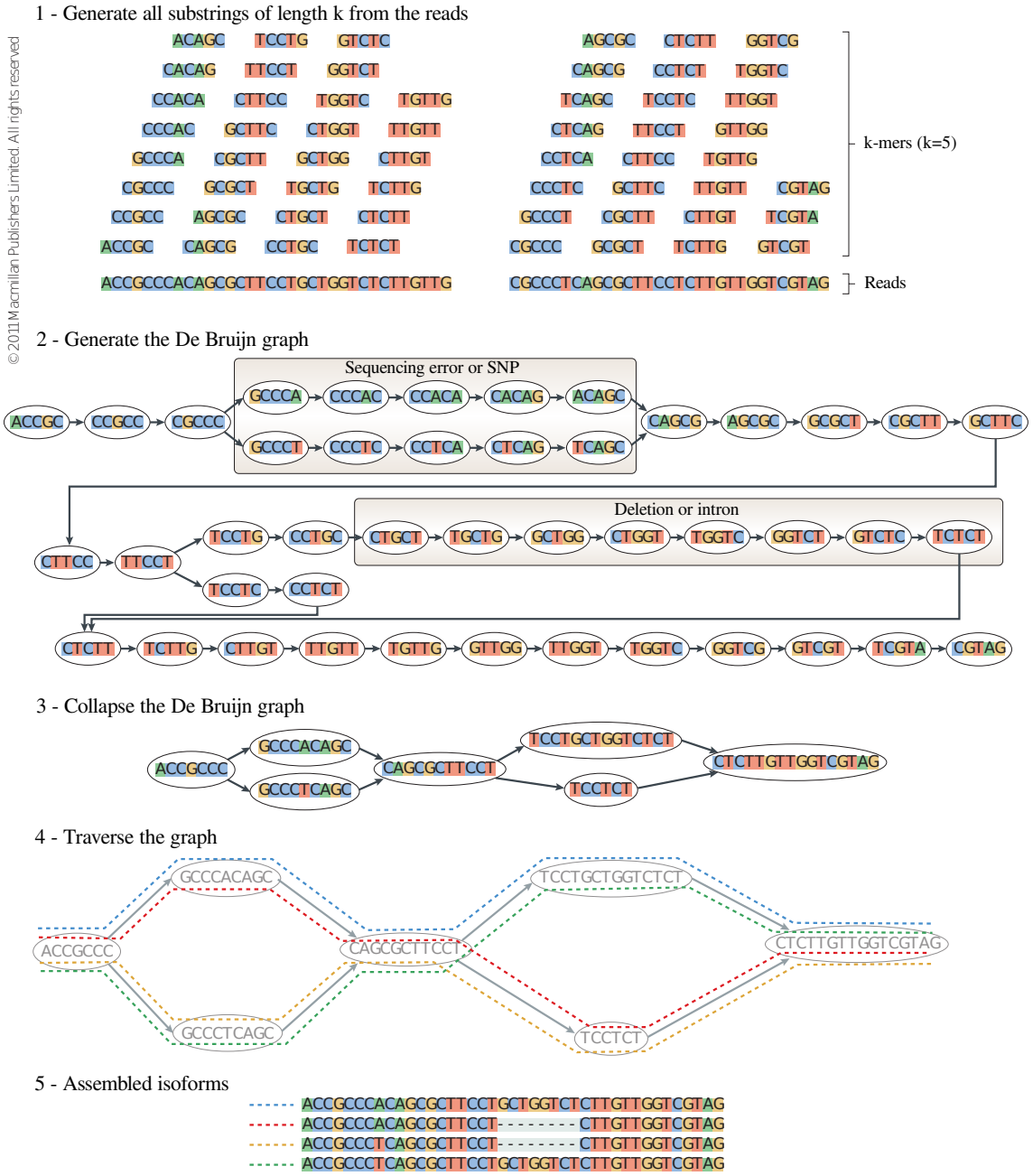



FIGURE 3.1: Genome Independent assembly. Reads are split into $k$-mers (1) and the *De Bruijn graph* is built, with the possibility of having sequencing errors or SNPs and also deletions or introns, that generate different branches (2). The graph is then contracted by collapsing all its linear paths (3). The contracted graph is traversed (4) and each isoform is assembled from an identified path (5).

among the contigs. Finally, for each cluster, a *De Bruijn graph* is built by assigning to each of the edges, a weight corresponding to the number of $k$-mer sets supporting it. In the last step, named *Butterfly*, the graph is contracted and then transcripts are assembled by finding paths and looking for reads (and paired-end) supporting them. Since genome independent approaches are usually memory and time consuming tasks (because of the huge number of reads to assemble), clustering the graph into independent subgraphs allows to speed up the procedure by executing it in parallel.

It is also possible to combine the two complementary approaches, in order to take advantage of the sensitivity of the genome guided methods and the ability in identifying novel transcripts of the genome independent ones. To this purpose, it is possible to align the reads into a reference genome (when available) and then assemble the remaining ones (or assemble the obtained transcripts by using a genome guided procedure). On the other hand, it is also possible to assemble the reads in order to obtain the transcripts and then map these latter contigs into the genome (maybe coming from a distant species), as a validation. The choice depends on what it is available and on the scope of the analysis.

As a last remark, it must be said that some of the previously introduced programs allow to quantify the expression levels of the (reconstructed) isoforms. To this purpose there have been introduced some metrics because in general there are some issues in estimating the expression. In fact, when sequencing a sample, longer transcripts will produce more reads compared to shorter transcripts and also, in different samples, the number and the quality of the reads could vary a lot. A widely adopted measure to quantify the gene expressions, is the reads per kilobase of exon model per million mapped reads (RPKM) [35].

### 3.1.2 Splicing Graph

A conceptual tool that has been used to investigate the reconstruction of full-length transcripts from ESTs (Expressed Sequence Tags) [19, 36] or RNA-Seq data [20] is the *splicing graph* which, give a "compact" view of the alternative splicing variants.

Before the advent of the NGS technologies, one of the most used strategy for the identification and quantification of the different full-length transcripts was the "single pass" sequencing of random cDNA clones. Thanks to this technique, also know as Expressed Sequence Tag (EST), it was possible to produce short partial sequences of a specific transcript. Due to the alternative splicing mechanism, the different isoforms produced from a RNA molecule can share part of their sequence. This was reflected also in the ESTs, making the reconstruction of the original full-length transcripts or the identification of the original isoform that has generated a specific EST, difficult. To face this

problem, a graph data structure was defined, in order to describe the relationship among different isoforms and the exon connections that compose them. This data structure, called splicing graph, instead of representing each transcript, gives a global representation of all the potential splicing variants, starting from the available EST and cDNA data. The splicing graph was defined in [19] as follows. Let $\{s_1, \ldots, s_n\}$ be the set of all RNA transcripts for a given gene of interest. Each transcript $s_i$ corresponds to a set of genomic positions $V_i$ with $V_i \neq V_j$ for $i \neq j$. Define the set of all transcribed positions $V = \bigcup_{i=1}^{n} V_i$ as the union of all sets $V_i$. The splicing graph $G$ is the directed graph on the set of transcribed positions $V$ that contains an edge $(v, w)$ if and only if $v$ and $w$ are consecutive positions in one of the transcripts $s_i$. Every transcript $s_i$ can be viewed as a path in the splicing graph $G$ and the whole graph $G$ is the union of $n$ such paths. Finally, in order to obtain a more compact representation of the splicing graph, the vertexes with *indegree*=*outdegree*=1 are usually collapsed. The splicing graph can be built starting from a set $S = \{s_1, \ldots, s_n\}$ of ESTs by considering the set of all its possible $k$-mers, named $Spec_k(S)$. In this way, the vertex set $V$ is obtained by the set of all the possible $(k-1)$-mers (i.e. $V = Spec_{k-1}(S)$) and for each $k$-mer $x_1 \ldots x_k \in Spec_k(S)$ there exists an edge $e = (x_1 \ldots x_{k-1}, x_2 \ldots x_k)$ between vertexes $(x_1 \ldots x_{k-1})$ and $(x_2 \ldots x_k)$.

From the above construction process, it must be noticed that errors in the input sequences can produce fake branches in the graph or can add spurious edges among vertexes. For this reason, it is necessary to use error correction tools that consider also the presence of SNPs (that can cause bulges in the graph). In fact, it is important to distinguish between errors and nucleotide variants for the correct identification of valid paths in the graph.

A more recent study involving the splicing graph was made to address the same problem as before, but starting from NGS data [20]. In particular, the authors face the problem of characterizing the *transcript variants* and their abundances, from a set of short reads (and paired-ends). In this case, the definition of the splicing graph, used to formulate such a problem, is the following.

**Definition 3.1.** Given a set $S_0$ of transcripts coming from an RNA (*variants*), the *splicing graph* is a directed acyclic graph whose vertexes are the maximal sequences of adjacent exons or exon fragments that always appear together in $S_0$ (called *blocks*), and whose directed edges join two blocks if the corresponding junctions is in at least one variant of $S_0$.

This definition introduces the concept of block. In this way, a variant (i.e. an isoform) is a directed path in the graph that corresponds to a sequence of blocks. It must be also noticed that the opposite is not always true, i.e. not all the paths in the splicing

graph are isoform of the input set. This means that there are valid directed paths that are not real variants. As anticipated before, this representation was used to estimate the transcript abundances. More specifically, this was done by formulating a system of linear equations, in which the abundances of block junctions are expressed as sum of the abundances of transcripts in which the blocks appear.

## 3.2 The Isoform Graph and the SGR Problem

In this work we use a notion of a splicing graph that is close to the one in [20], where splicing graphs provide a representation of variants. Since our main goal is the reconstruction of the splicing graph from the nucleotide sequences of a set of short reads without the knowledge of a genomic sequence, some definitions will be slightly different from [20] where the notion of abundance of reads spanning some splicing junction sites is central. Moreover our goal is to study transcripts data originating from different tissues or samples, where the expression level of each single transcript greatly varies among samples. Therefore introducing abundance into consideration is likely to introduce a number of complications in the model as well as in the algorithms, while increasing the presence of possible confounding factors.

Informally, a splicing graph is the graph representation of a gene structure, inferred from a set of RNA-Seq data, where isoforms correspond to paths of the splicing graphs, while splicing events correspond to specific subgraphs.

In this work we consider discrete genomic regions (i.e. a gene or a set of genes) and their full-length isoforms or transcript products of the genes along these regions. Let us recall that the genomic region of a gene consists of a sequence of coding regions (exons) alternating with non-coding ones (introns). A gene isoform is a concatenation of some of the coding regions of the gene respecting their order in the genomic region. Alternative splicing regulates how different coding regions are included to produce different full-length isoforms or transcripts, which are modeled here as sequences of *blocks*. Formally, a *block* consists of a string, typically taken over the alphabet $\Sigma = \{a, c, g, t\}$, and an integer called rank, such that no two blocks share the same rank. The purpose of introducing the rank of a block is to disambiguate between blocks sharing the same nucleotide sequence (i.e. string) and to give an order among blocks, reproducing the order of exons in the genomic region.

Given a block $b$, we denote by $s(b)$ and $r(b)$ its string and rank respectively. In our framework a *gene coding region* is a sequence (that is, an ordered set) $B = \langle b_1, b_2, \cdots, b_n \rangle$ of blocks with $r(b_i) = i$ for each $i$. Then, the *string coding region for* $B$ is the string

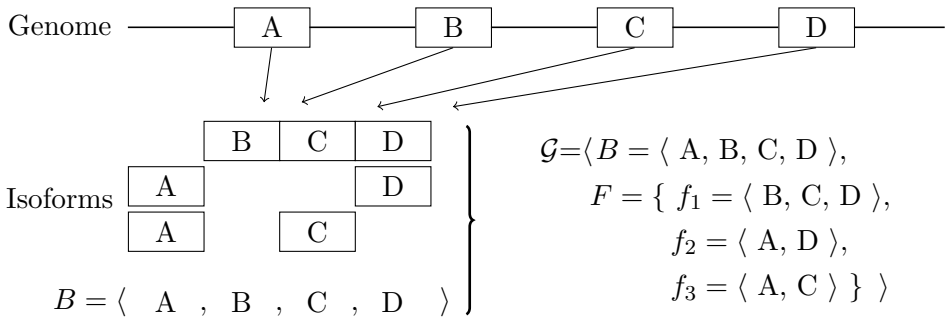


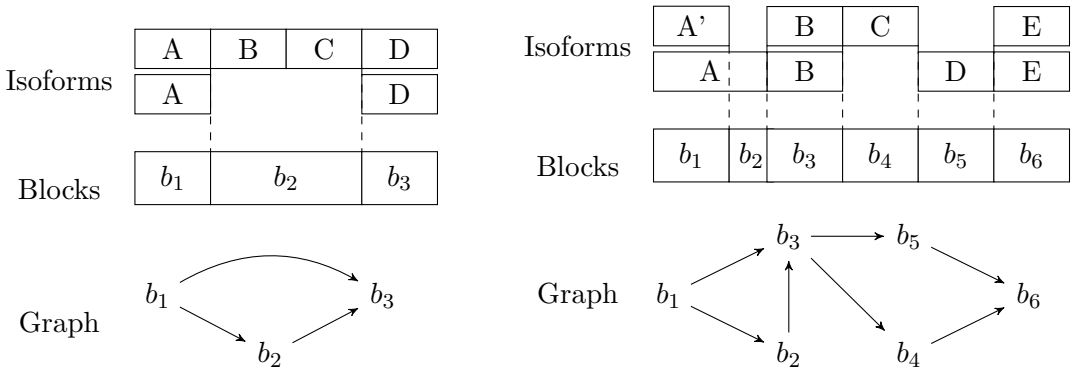

FIGURE 3.2: Example of expressed gene. Given a gene in which its isoforms are composed of blocks (that for simplicity are associated to exons: $A$, $B$, $C$ and $D$) extracted from the genome, we represent the sequence $B$ of blocks based on the (genomic) position. The expressed gene $\mathcal{G} = \langle B, F \rangle$ is constructed from the block sequence $B$ and its isoform composition.

$s(b_1)s(b_2)\cdots s(b_n)$ obtained by orderly concatenating the strings of the blocks in $B$. Intuitively a gene coding region is the sequence of all the coding regions on the whole genomic sequence for the studied gene. We define a *block isoform* $f$ *compatible* with $B$, as a subsequence of $B$, that is $f = \langle b_{i_1}, \cdots, b_{i_k} \rangle$ where $i_j < i_{j+1}$ for $1 \leqslant j < k$. We distinguish between classical isoforms (defined on exons or genomic regions) and block isoforms (defined on blocks). Nonetheless, we will use interchangeably the terms isoforms and block isoforms whenever no ambiguity arises. By a slight abuse of language we define the string of $f$, denoted by $s(f)$, as the concatenation of the strings of the blocks of $f$.

**Definition 3.2.** An *expressed gene* is a pair $\langle B, F \rangle$, where $B$ is a gene coding region, $F$ is a set of block isoforms compatible with $B$ where (1) each block of $B$ appears in some isoform of $F$, and (2) for each pair $(b_i, b_j)$ of blocks of $B$, appearing consecutively in some isoform of $F$, there exists a isoform $f \in F$ s.t. exactly one of $b_i$ or $b_j$ appears in $f$.

We point out that Definition 3.2 is mostly compatible with that of [20], where a *block* is defined as a maximal sequence of adjacent exons, or exons fragments, that always appear together in a set of isoforms or variants. Therefore their approach downplays the relevance of blocks with the same string. Observe that Definition 3.2 implies that the set $B$ of blocks of a string coding region of an expressed region $\langle B, F \rangle$ is unique and is a minimum-cardinality set explaining all isoforms in $F$. Thus, the pair $\langle B, F \rangle$ describes a specific gene (see Figure 3.2).

In fact, from Definition 3.2, given two consecutive blocks $b, b' \in B$, if for all the isoforms $f \in F$ either both $b$ and $b'$ occur in $f$, or neither $b$ nor $b'$ occur in $f$, then such blocks $b$ and $b'$ can be concatenated into a new block $b''$ that can replace $b, b'$ in $B$ obtaining a new set of blocks $B'$ of cardinality $|B| - 1$, that preserves the characteristics of $B$.




(a) Exon Skipping.

(b) Alternative Donor Site and Mutually Exclusive Exons.

FIGURE 3.3: Examples of Isoform Graphs. These two examples show the isoform composition of a gene (1) with the corresponding block sequence (2) and their isoform graphs (3). Capital letters correspond to exons. In (a) is represented a skipping of the two consecutive exons $B$ and $C$ of the second isoform w.r.t. the first one. In (b) is represented an alternative donor site variant between exons $A$ and $A'$ and two mutually exclusive exons $C$ and $D$. Notice that, in this figure, the isoforms represented are classical.

Now assume that an expressed region for a set $F$ of isoforms is not unique, but there exists two sets of blocks $B_1$ and $B_2$ such that are gene representations for $F$ and satisfy the definition of expressed region given in Definition 3.2. Then there must exists a block $b_1 \in B_1 - B_2$ and a block $b_2 \in B_2 - B_1$. Clearly there exists an isoform $f_1 \in F$ such that $b_1$ is a block of $f_1$. Then, since $b_1 \notin B_2$ but $f_1 \in F$, it must be that there exists a different composition of $f_1$ that uses block $b_2$ instead of block $b_1$. But this fact holds if and only if a block $b_0$ of $f_1$ includes another block $b$ of $f_1$. But being blocks not-overlapping, it must be that $b_0$ consists of two blocks, thus contradicting the minimality of blocks proved before.

The uniqueness of blocks of an expressed gene allows us to define the associated graph representation, or isoform graph. Given an expressed gene $\mathcal{G} = \langle B, F \rangle$, the *isoform graph* of $\mathcal{G}$ is a directed graph $G_I = \langle B, E \rangle$, where an ordered pair $\langle b_i, b_j \rangle$ is an arc of $E$, iff $b_i$ and $b_j$ are consecutive in at least an isoform of $F$. Notice that $G_I$ is a directed acyclic graph, since the sequence $B$ is also a topological order of $G_I$. Notice that isoforms correspond to paths in $G_I$. Moreover, all the alternative splicing events of the considered transcripts (such as exon skipping, mutually exclusive exons, ...) correspond to specific subgraphs of $G_I$ (see Figure 3.3).

Our first aim of the work is to characterize when and how accurately the isoform graph of an expressed gene can be reconstructed from a set of substrings (i.e. RNA-Seq data) of the isoforms of the gene. More precisely, we want to investigate the following two main questions.

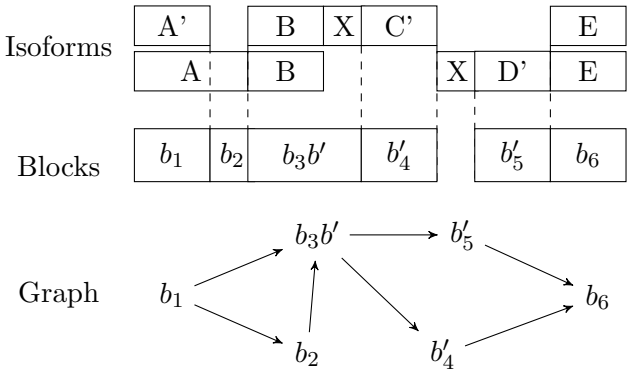



FIGURE 3.4: Alternative splicing graph compatible with the reads of the expressed gene of Figure 3.3(b)

*Question 1:* What are the conditions under which the isoform graph of a gene can be reconstructed from a sample of RNA-Seqs (without putting any bounds on the computational resources)?

*Question 2:* Can we build efficiently such a graph or an approximation of it?

Notice that the isoform graph is the real gene structure that we would like to infer from data but, at the same time, we must understand that the transcript data might not be sufficient to determine the isoform graph, as we have no information on the genomic sequence and on the blocks in particular. Therefore we aim at computing a slightly less informative kind of graph: the *splicing graph*, which is a directed graph where each vertex $v$ is labeled by a string $s(v)$. Notice that the splicing graph gives no assurance that a vertex is a block, not does it contain any indication regarding whether (and where) the string labeling a vertex appears in the genomic region.

For instance, let us consider the isoform graph of Figure 3.3(b). Assume that $s(b_4)$ and $s(b_5)$ share a common prefix $s(b')$, that is the exons C and D can be respectively written as XC' and XD'. Then if no information about the block positions and rank on the genomic sequence is provided as input data, the splicing graph of Figure 3.4 could be as plausible as the isoform graph of Figure 3.3(b).

This observation leads us to the notion of splicing graph *compatible* with a set of isoforms.

**Definition 3.3.** Let $\langle B, F\rangle$ be an expressed gene, and let $G = \langle V, E\rangle$ be a splicing graph. Then $G$ is *compatible* with $F$ if, for each isoform $f \in F$, there exists a path $p = \langle w_1, \ldots, w_k\rangle$ of $G$ such that $s(f) = s(w_1) \cdots s(w_k)$.

Since we have no information on the blocks of the expressed gene, computing any graph that is compatible with the isoform graph is an acceptable answer to our problem. We need some more definitions related to the fact that we investigate the problem of

reconstructing a splicing graph compatible with a set of isoforms only from RNA-Seqs obtained from the gene transcripts.

Let $\langle B, F\rangle$ be an unknown expressed gene. Then, a *RNA-Seq read* (or simply *read*) extracted from $\langle B, F\rangle$, is a substring of the string $s(f)$ of some isoform $f \in F$. Notice that we know only the nucleotide sequence of each read. Just as we have introduced the notion of splicing graph compatible with a set of isoforms, we can define the notion of splicing graph compatible with a set of reads.

**Definition 3.4.** Let $R$ be a set of reads extracted from an expressed gene $\langle B, F\rangle$, and let $G = \langle V, E\rangle$ be a splicing graph. Then $G$ is *compatible* with $R$ if, for each read $f \in R$, there exists a path $p = \langle w_1, \ldots, w_k\rangle$ of $G$ such that $r$ is a *substring* of $s(w_1) \cdots s(w_k)$.

*Problem* 1. **Splicing Graph Reconstruction (SGR) Problem**
**Input**: a set $R$ of reads, all of length $l$, extracted from an unknown expressed gene $\langle B, F\rangle$.
**Output**: a splicing graph compatible with $R$.

Clearly SGR can only be a preliminary version of the problem, as we are actually interested into finding a splicing graph that is most similar to the isoform graph associated to $\langle B, F\rangle$. Therefore we need to introduce some criteria to rank all splicing graphs compatible with $R$. The parsimonious principle leads us to a natural objective function (albeit we do not claim it is the only possibility): to minimize the sum of the lengths of strings associated to the vertices (mimicking the search for the shortest possible string coding region). In the rest of paper the SGR problem will ask for a splicing graph minimizing such sum.

## 3.3 Unique solutions to SGR

In this section we will show some conditions must be satisfied if we want to be able to optimally solve the SGR problem. Notice that, given an isoform graph $G_I$ there is a splicing graph $G_S$ naturally associated to it, where the two graphs $G_I$ and $G_S$ are isomorphic (except for the labeling) and the label of each vertex of $G_S$ is the string of the corresponding vertex of $G_I$.

Let $R$ be an instance of SGR originating from an expressed gene $\langle B, F\rangle$. Then $R$ is a *solvable* instance if: (i) for each three blocks $b$, $b_1$ and $b_2$ s.t. $b$ and $b_1$ are consecutive in an isoform, $b$ and $b_2$ are consecutive in another isoform, then $b_1$ and $b_2$ begin with different characters. Also, for each three blocks $b$, $b_1$ and $b_2$ s.t. $b_1$ and $b$ are consecutive in an isoform, $b_2$ and $b$ are consecutive in another isoform, then $b_1$ and $b_2$ end with different

characters; (ii) for each subsequence $B_1$ of $B$, the string $s(B_1)$ does not contain two identical substrings of length $l/2$ (where $l$ is the length of the reads). We will show here that our theoretical analysis can focus on solvable instances, since for each condition of solvable instance we will show an example where there exists an optimal splicing graph – different from the isoform graph – compatible with the instance.

Condition (i). Let $B = \{b, b_1, b_2\}$, and let $F = \{\langle b, b_1\rangle, \langle b, b_2\rangle\}$. Moreover $s(b_1)[1] = s(b_2)[1] = x$, that is the strings of both blocks $b_1$ and $b_2$ begins with the symbol $x$, which does not appear in any other position of the string coding region. Consider now the expressed gene $B' = \{b', b'_1, b'_2\}$, and let $F' = \{\langle b', b'_1\rangle, \langle b', b'_2\rangle\}$, where $s(b') = s(b)x$, $s(b'_1) = s(b'_1)[2:]$, and $s(b'_2) = s(b'_2)[2:]$ (informally, the symbol $x$ is removed from $b_1$ and $b_2$ and attached to the end of $b$). It is immediate to notice that $|s(b)| + |s(b_1)| + |s(b_2)| > |s(b')| + |s(b'_1)| + |s(b'_2)|$ and any set of reads that can be extracted from one gene can also be extracted from the other. A similar argument holds for the condition on the final character of the string of each block.

Condition (ii). Let us consider three blocks $b_1$, $b_2$ and $b_3$ such that $b_2$ and $b_3$ contains the same $l/2$-long substring $z$, that is $s(b_2) = p_2 z s_2$ and $s(b_3) = p_3 z s_3$. There are two isoforms: $\langle b_1, b_2\rangle$ and $\langle b_1, b_3\rangle$. Besides the isoform graph, there is another splicing graph $v_1, \ldots, v_5$ where $s(v_1) = s(b_1)p_2$, $s(v_2) = s(b_1)p_3$, $s(v_3) = z$, $s(v_4) = p_2$, $s(v_5) = p_3$ that is compatible with any set of reads extracted from $\langle B, F\rangle$. The arcs of this splicing graphs are $(v_1, v_2)$ and $(v_1, v_3)$. Notice that the sum of lengths of the labels of this new splicing graph is smaller than that of the isoform graph.

## 3.4 Methods

In order to investigate the two main questions stated before, we propose a method for solving the SGR problem. Our approach to compute the isoform graph $G_S$ first identifies the vertex set $B_S$ and then the edge set $E_S$ of $G_S$. Moreover we focus on fast and simple methods that can possibly scale to genome-wide data. For ease of exposition, the discussion of the method assumes that reads have no errors, and $l = 64$.

The basic idea of our method is that we can find two disjoint subsets $R_1$, and $R_2$ of the input set $R$ of reads, where reads of $R_1$, called *unspliced*, can be assembled to form the nodes in $B_S$, while reads of $R_2$, called *spliced*, are an evidence of a junction between two blocks (that is an arc of $G_S$).

We will discuss how our method deals with problems as errors, low coverage.

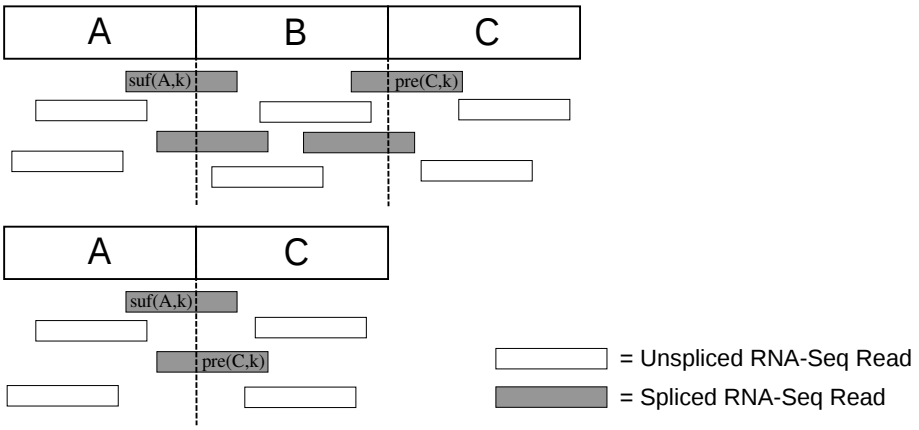



FIGURE 3.5: Example of read classification. The example shows two transcripts, where the first one is composed of 3 blocks ($A$, $B$ and $C$) and the second one of 2 blocks ($A$ and $C$). Moreover, reads coming from those blocks are partitioned into unspliced (white rectangles) and spliced (gray rectangles). Notice that the two reads sharing the same suffix of block $A$ (of length $k$) and those two sharing the same prefix of block $C$ (of length $k$) are labeled with $suf(A, k)$ and $pre(C, k)$, respectively.

The second main tenet of our algorithm is that each read is encoded by a 128-bit binary number, divided into a *left fingerprint* and a *right fingerprint* (respectively the leftmost and the rightmost 64 bits of the encoding). Then two reads $r_1$ and $r_2$ overlap for at least $l/2 = 32$ base pairs iff the right fingerprint of $r_1$ is a substring of the encoding of $r_2$ (a similar condition holds for the left fingerprint of $r_2$). Bit-level operations allows to look for such a substring very quickly.

**Definition 3.5.** Let $r$ be a read of $R$. Then $r$ is *spliced* if there exists another $r' \in R$, $s(r) \neq s(r')$, such that $\text{pre}(r, k) = \text{pre}(r', k)$ or $\text{suf}(r, k) = \text{suf}(r', k)$, for $l/2 \leqslant k$. Moreover a read $r$ is *perfectly spliced* if there exists another $r' \in R$, $s(r) \neq s(r')$, such that the longest common prefix (or suffix) of $r$ and $r'$ is exactly of length $l/2$. A read that is not spliced is called *unspliced*.

More specifically, from Definition 3.5, in an ideal situation, unspliced reads represent the substrings of the blocks, while spliced reads identify junctions among them (see Figure 3.5).

In our framework, a junction site between two blocks $b_1$ and $b_3$, that appear consecutively within an isoform, is detected when we find a third block $b_2$ that, in some isoform, appears immediately after $b_1$ or immediately before $b_3$. For illustrative purposes, let us consider the case when $b_2$ appears immediately after $b_1$ in an isoform (Figure 3.3(a)). The strongest possible signal of such junction consists of two reads $r_1$ and $r_2$ such that $ov(s(b_1), r_1) = ov(r_1, s(b_3) = l/2$ and $ov(s(b_2), r_2) = ov(r_2, s(b_3)) = l/2$, that is $r_1$ is cut into halves by the junction separating $b_1$ and $b_3$, while $r_2$ is cut into halves by the junction separating $b_2$ and $b_3$ (i.e. $r_1$ and $r_2$ are perfectly spliced). In a less-than-ideal

scenario, we still can find two reads $r_1$ and $r_2$ sharing a common prefix (or suffix) that is longer than $l/2$, in which case the two reads are spliced.

Notice that all reads extracted from the same block can be sorted so that any two consecutive reads have large overlap. More formally, we define a *chain* as a sequence $C = \langle r_1, r_2, \cdots, r_n \rangle$ of unspliced reads where $ov(r_i, r_{i+1}) = l/2$ for $1 \leqslant i < n$ (notice that the $\mathrm{RH}(r_i) = \mathrm{LH}(r_{i+1})$, which allows for a very fast computation). Let $C$ be a chain. Then the string of $C$ is the string $s(C) = \varphi(C)$, moreover $C$ is *maximal* if no supersequence of $C$ is also a chain. Under ideal conditions (i.e. no errors and high coverage) $s(C)$ is exactly the string of a block. Coherently with our reasoning, a perfectly spliced read $r$ is called a *link* for the pair of chains $(C, C')$, if $\mathrm{LH}(r) = \mathrm{suf}(s(C), l/2)$ and $\mathrm{RH}(r) = \mathrm{pre}(s(C'), l/2)$. In this case we also say that $C$ and $C'$ are respectively *left-linked* and *right-linked* by $r$.

Given a set $R$ of reads extracted from the isoforms $F$ of an unknown expressed region $\langle B, F \rangle$, our algorithm outputs a likely isoform graph $G_R = (\mathcal{C}, E_R)$, where $\mathcal{C} = \{C_1, \cdots, C_n\}$ is a set of maximal chains that can be derived from $R$, and $(C_i, C_j) \in E_R$ iff there exists in $R$ a link for $(C_i, C_j)$. The remainder of this section is devoted to show how we compute such graph efficiently even under less-than-ideal conditions. The algorithm is organized into three steps that are detailed below. In the first step we build a data structure to store the reads in $R$. We use two hash tables which guarantee a fast access to the input reads. The second step creates the nodes of $G_R$ by composing the maximal chains of the unspliced reads of $R$. In the last step of the creation of $G_R$, the maximal chains obtained in the second step are linked.

**Step 1: Fingerprinting of RNA-Seq reads**

For each read longer than 64bp we extract some substrings of 64bp that are representative of the original read. Depending on the adopted strategy, it is possible to extract all the 64bp substrings or only the leftmost and rightmost ones. Then each 64bp read is unambiguously encoded by a 128-bit binary number, exploiting the fact that we can encode each symbol with 2 bits as follows: $\mathrm{enc}(a) = 0 = 00_2$, $\mathrm{enc}(c) = 1 = 01_2$, $\mathrm{enc}(g) = 2 = 10_2$, $\mathrm{enc}(t) = 3 = 11_2$. Since such encoding is a one-to-one mapping between reads and numbers between 0 and $2^{128} - 1$, we will use interchangeably a string and its fingerprint.

Moreover, given a read $r$, we define $\phi_1(r)$ (also called *left fingerprint*) and $\phi_2(r)$ (also called *right fingerprint*) respectively as the leftmost and the rightmost 64 bits of the encoding of $r$.

The main data structures are two tables $\mathcal{L}_l$ and $\mathcal{L}_r$, both of which are indexed by 64-bit fingerprints. More precisely, $\mathcal{L}_l$ has an entry indexed by each left fingerprint, while $\mathcal{L}_r$ has an entry indexed by each right fingerprint. The entry of $\mathcal{L}_l$, associated to the left fingerprint $f_l$, consists of a list of all the right fingerprints $f_r$ such that the concatenation $f_l f_r$ is a read in the input set. The role of $\mathcal{L}_r$ is symmetrical. The purpose of those tables is that they allow for a very fast labeling of each read into unspliced or perfectly spliced reads. In fact, a read $r$ is unspliced iff both the entry of $\mathcal{L}_l$ indexed by its left fingerprint and the entry of $\mathcal{L}_r$ indexed by its right fingerprint are lists with only one element. In this way, if a collision is detected during the insertion of a fingerprint in a table, the read is labeled as spliced. Moreover, let $f_l$ be a left fingerprint of some reads, let $f_{r,1}$ and $f_{r,2}$ be two fingerprints in the list of $\mathcal{L}_l$ indexed by $f_l$, such that the first character of $f_{r,1}$ is different from that of $f_{r,2}$. Then the two reads $f_l f_{r,1}$ and $f_l f_{r,2}$ are perfectly spliced. Also, constructing $\mathcal{L}_l$ and $\mathcal{L}_r$ requires time proportional to the number of the input reads.

### Step 2: Building the set $\mathcal{C}$ of Maximal Chains

The procedure *BuildChains* described in Algorithm 1 takes as input a set $R$ of RNA-Seq reads and produces the set $\mathcal{C}$ of all maximal chains that can be obtained from $R$. Let $R_1 \subseteq R$ be the set of the *unspliced* reads. The algorithm selects any read $r$ of $R_1$ and tries to find a *right extension* of $r$, that is another unspliced read $r_r$ such that $ov(r, r_r) = l/2$. Afterwards the algorithm iteratively looks for a right extension of $r_r$, until such a right extension no longer exists. Then, the algorithm iteratively looks for a left extension of $r$, while it is possible.

Also, the time required by this procedure is proportional to the number of unspliced reads. In fact, each unspliced read is considered only once, and finding the left or right extension of a read $r$ can be performed in constant time.

At the end, we will merge all pairs of chains whose strings have an overlap at least $l/2$ bases long, or one is a substring of the other. In this step, called *chain merging*, for decreasing values of $k$ ranging from 63 to 32, we determine if there exists a read $r_r$ such that $ov(r, r_r) = k$, exploiting the $\mathcal{L}_l$ table. This step is necessary to overcome to the absence of some unspliced reads, that can cause the shortening of chains. Moreover, by assembling reads that have a $l/2$ overlap, it happens that there will be more than one chain representing the same block. In ideal conditions, such chains are all the possible $(l/2) - 1$ shifts of the reads in the block. The chain merging this step is achieved by extracting all possible 32-base substrings of $r$ and looking in the $\mathcal{L}_l$ table for that

substring. If such a substring is found, the *fusion* of the two strings (representing the same block) is preformed.

We recall that the maximal chains are the vertices of the isoform graph we want to build.

**Algorithm 1**: BuildChains($R$)

**Data**: a set $R$ of RNA-Seq reads

1 $\mathcal{C} \leftarrow \varnothing$;
2 $R_1 \leftarrow \{r \in R | r$ is *unspliced*$\}$;
3 **while** $R_1 \neq \varnothing$ **do**
4 $r \leftarrow$ any read from $R_1$;
5 $R_1 \leftarrow R_1 \setminus \{r\}$;
6 $C \leftarrow \langle r \rangle$;
7 $r_1 \leftarrow r$;
`// Extend the chain on the right`
8 **while** $\exists$ *a right extension* $r_2 \in R_1$ *of* $r_1$ **do**
9 append $r_2$ to $C$;
10 $R_1 \leftarrow R_1 \setminus \{r_2\}$;
11 $r_1 \leftarrow r_2$;
`// Extend the chain on the left`
12 **while** $\exists$ *a left extension* $r_2 \in R_1$ *of* $r$ **do**
13 prepend $r_2$ to $C$;
14 $R_1 \leftarrow R_1 \setminus \{r_2\}$;
15 $r \leftarrow r_2$;
16 $\mathcal{C} \leftarrow \mathcal{C} \cup C$;
17 **return** $\mathcal{C}$;

#### Step 3: Linking Maximal Chains

Algorithm 2 computes the arcs of the output graph using the set $R_2$ of *perfectly spliced* reads and the set $\mathcal{C}$ of maximal chains computed in the previous step. More precisely, given a perfectly spliced read $r$, we denote with $\mathcal{D}(r)$ and $\mathcal{A}(r)$ the set of maximal chains that are, respectively, left-linked and right-linked by $r$. In other words, $r$ is a *link* for each pair of chains in $\mathcal{D}(r) \times \mathcal{A}(r)$.

Moreover each such pair will be an arc of the graph. All sets $\mathcal{D}(r)$ and $\mathcal{A}(r)$ are computed in the cycle at line 4 of Algorithm 2. The cycle at line 11 obtains the whole set $E_R$ of arcs of $G_R$. Algorithm 2 is greatly sped up if Algorithm 1 also stores the prefix and the suffix of length 32bp of each maximal chain in $\mathcal{C}$.

### 3.4.1 Isomorphism between predicted and true isoform graph

Let $R$ be an instance of SGR originating from an expressed region $\langle B, F \rangle$. We can prove that a simple polynomial time variant of our method computes a splicing graph $G_R$ that

**Algorithm 2**: LinkChains($R_2$, $\mathcal{C}$)

**Data**: a set $R_2$ of perfectly spliced reads, and the set $\mathcal{C}$ computed by Algorithm 1

$E_R \leftarrow \varnothing$;
**foreach** $r \in R_2$ **do**
    $\mathcal{D}(r) \leftarrow \varnothing$; $\mathcal{A}(r) \leftarrow \varnothing$;
**foreach** $C \in \mathcal{C}$ **do**
    $f \leftarrow \text{pre}(s(C), l/2)$;
    **foreach** $r \in R_2$ *such that* $\text{RH}(r) = f$ **do**
        $\mathcal{A}(r) \leftarrow C$;
    $f \leftarrow \text{suf}(s(C), l/2)$;
    **foreach** $r \in R_2$ *such that* $\text{LH}(r) = f$ **do**
        $\mathcal{D}(r) \leftarrow C$;
**foreach** $r \in R_2$ **do**
    **if** $\mathcal{D}(r) \neq \varnothing$ *and* $\mathcal{A}(r) \neq \varnothing$ **then**
        **foreach** $p \in \mathcal{D}(r) \times \mathcal{A}(r)$ **do**
            $E_R \leftarrow E_R \cup p$;
**return** $E_R$

is isomorphic to the true isoform graph, when $R$ is a good instance. More precisely, $R$ is a *good* instance if it is solvable and (iii) for each isoform $f$ there exists a sequence $r_1, \ldots, r_k$ of reads such that each position of $f$ is covered by some read in $r_1, \ldots, r_k$ (i.e. $s(f)$ is equal to the fusion of $r_1, \ldots, r_k$) and $|ov(r_i, r_{i+1})| \geqslant l/2$ for each $i < k$.

First of all, notice that two reads $r_1$ and $r_2$ with overlap at least $l/2$ can be extracted from the same isoform. Let us build a graph $G$ whose vertices are the reads and an arc goes from $r_1$ to $r_2$ if and only if $ov(r_1, r_2) \geqslant l/2$ and there exists no read $r_3$ such that $ov(r_1, r_3) \geqslant ov(r_1, r_2)$ and $ov(r_3, r_2) \geqslant l/2$. By the above observation and by condition (iii) there is a 1-to-1 mapping between maximal paths in such graph and isoforms and the string of an isoform is equal to the fusion of the reads of the corresponding path. Compute the set $R_1$ of all $l$-mers that are substrings of the string of some isoforms. Then classify all reads of $R_1$ into unspliced, perfectly spliced and (non-perfectly) spliced reads, just as in our method. Notice that the halves of each perfectly spliced read are the start or the end of a block. Assemble all unspliced reads into chains where two consecutive reads have overlap $l - 1$ (each unspliced read belongs to exactly one chain), using the putative start/end $l/2$-mers computed in the previous step to trim the sequences of each block. At the end, each perfectly spliced read links two blocks of the splicing graph.

### 3.4.2 Low coverage, errors and SNP detection

We will consider here what happens when the input instance does not satisfy the requirements of a good instance. There are at least two different situations that we will have to tackle: data errors and insufficient coverage.

One of the effects of the chain merging phase is that most errors are corrected. In fact the typical effect of a single-base error in a read $r$ is the misclassification of $r$ as spliced instead of unspliced, shortening or splitting some chains. Anyway, as long as there are only a few errors, there exist some overlapping error-free unspliced reads that span the same block as the erroneous read. Those unspliced reads allow for the correction of the error and the construction of the correct chain spanning the block.

Notice that the chain merging also lessens the impact of partial coverage – that is when we do not have all possible $l$-mers of a block. When working under full coverage, we can identify a sequence of reads spanning a block and such that two consecutive reads have overlap equal to $l-1$, while the chain merging step successfully reconstruct the blocks with reads overlapping with at least $l/2$ characters.

A similar idea is applied to pairs of reads $r_1$, $r_2$ with $ov(r_1, r_2) \geqslant l/2$ and that are likely to span more than one block. Those reads can be detected by analyzing the hash tables. In this case a set of additional reads, compatible with the fusion of $r_1$ and $r_2$ is added to the input set, obtaining an enriched set which includes the perfectly spliced reads required to correctly infer the junction, even when the original reads have low coverage.

Also the fact that the definition of perfectly spliced read asks for two reads with the same left (or right) fingerprint, makes our approach more resistant to errors, as a single error is not sufficient to generate an arc in the splicing graph.

Finally, we point out that our approach allows for SNP detection. The main problem is being able to distinguish between errors and SNPs: let us consider an example that illustrates a strategy for overcoming this problem. Let $e$ be an exon containing a SNP, that is $s(e)$ can be $yaz$ or $ybz$, where $y$ and $z$ are two strings and $a$, $b$ are two characters. Moreover notice that, since this situation is a SNP, roughly the same number of reads support $yaz$ as $ybz$. Therefore, as an effect of our read enrichment step, there are two reads $r_1$ and $r_2$ s.t. $r_1$ supports $yaz$ and $r_2$ supports $ybz$, and $\mathrm{LH}(r_1) = \mathrm{LH}(r_2)$ or $\mathrm{RH}(r_1) = \mathrm{RH}(r_2)$. Equivalently, $r_1$ and $r_2$ are two spliced reads supporting the SNP. This case can be easily and quickly found examining the list of reads sharing the left (or right) fingerprints and then looking for a set of reads supporting the SNP (again exploiting the fact that the fingerprint of a half of the reads in the set is known).

### 3.4.3 Repeated sequences: stability of graph $G_R$

When compared with approaches based on de Bruijn graphs, our method is stable w.r.t. repeated sequences shorter than $l/2$, that is our method is not negatively influenced by those short repeats.

Let us state formally this property. An algorithm to solve the SGR problem is *k-stable* if and only if we obtain a new isoform set $F'$ from $F$ by disambiguating each $k$-long substring that appears in more than one isoform but originates from different parts of the string coding region, then the graph obtained from $F$ is isomorphic to that obtained from $F'$.

Notice that de Brujin graphs are highly sensitive to this case, as they must merge $k$-long repetitions into a single vertex. On the other hand, our approach can avoid merging $(l/2-1)$-long repetitions, as the chain construction step is based on finding $l/2$-long identical substrings in the input reads. Validating this property is one of the features of our experimental analysis.

Let us consider the following example. Let $s_A = yvz$ and $s_B = uvw$ be respectively a block of gene $A$ and $B$, therefore $s_A$ and $s_A$ belong to two different weakly connected components of the isoform graph. Assume that $v$ is a sequence of length $k < l/2$, where $k$ is the parameter length used in the construction of de Brujin graphs (i.e. the vertices correspond to $k$-mers), and consider the case where reads coming from both genes are given in input to compute a splicing graph. If the instance is good, our approach is able to reconstruct the isoform graph, while a typical algorithm based on de Brujin graphs would have a single vertex for $x$. Notice also that the resulting graph would be acyclic, hence the commonly used technique of detecting cycles in the graph to determine if there are duplicated strings is not useful.

## 3.5 Experimental Results

We have run our experimental analysis on simulated (and error-free) RNA-Seq data obtained from the transcript isoforms annotated for a subset of 112 genes extracted from the 13 ENCODE regions used as training set in the EGASP competition (we refer the interested reader to [37] for the complete list of regions and genes). We have chosen those genes because they are well annotated and, at the same time, are considered quite hard to be analyzed by tools aiming to compute a gene structure, mainly due to the presence of repeated regions. Moreover, the presence of high similar genes makes this set very hard to be analyzed as a whole. Also, we decided to focus on a relatively small number of (hard to analyze) genes so that we could manually inspect the results to determine the causes of incorrect predictions.

A primary goal of our implementation is to use only a limited amount of memory, since this is the main problem with currently available programs [25]. In fact, we have run our program on a workstation with two quad-core Intel Xeon 2.8GHz processors and

12GB RAM. Even on the largest dataset, our program has never used more 30 seconds or more than 250MB of memory. Our implementation is available under AGPLv3 at `http://algolab.github.com/RNA-Seq-Graph/`.

Now let us detail the experiments. For each of the 112 ENCODE genes, the set of the annotated full-length transcripts has been retrieved from NCBI GenBank. From those transcripts we have extracted two sets of 64bp substrings corresponding to our simulated reads. The first set consists of all possibile 64-mers and corresponds to the maximum possible coverage (*full coverage* dataset), while the second set consists of a random set of 64-mers simulating an 8x coverage (*low coverage* dataset). In the first experiment we have analyzed the behavior on the full coverage dataset where data originating from each gene have been elaborated separately and independently. The second experiment is identical to the first, but on the low coverage dataset. Finally, the third experiment has been run on the whole full coverage dataset, that is all reads have been elaborated together and without any indication of the gene they were originating from. Notice that the whole full coverage dataset consists of 1.4 Million unique 64bp simulated reads. For each input gene, the true isoform graph has been reconstructed from the annotation.

To evaluate how much the splicing graph computed (denoted as $G_R$) is similar to the isoform graph (denoted as $G_S$), we have designed a general procedure to compare two labeled graphs, exploiting not only the topology of the graph, but also the labeling of each vertex with the goal of not penalizing small differences in the labels, like for example the lack of some bases at the end of a block (which corresponds to a correct detection of the AS events and a mostly irrelevant error in computing the nucleotide sequence of a block). The comparison of the graphs $G_R$ and $G_S$ is not straightforward, therefore we discuss next the procedure we have designed.

For clarity's sake, we consider that vertexes of both graphs are strings instead of blocks. Let $s, t$ be two strings. Then $s$ and $t$ are $p$-trim equivalent if it is possible to obtain the same string by removing from $s$ and $t$ a prefix and/or a suffix no longer than $p$ (notice that the removed prefixes and suffixes might be empty and can differ, in length and symbols, between $s$ and $t$). Let $v$ and $w$ be respectively a vertex of $G_R$ and of $G_S$. Then $v$ *maps* to $w$, if $v$ and $w$ are 5-trim equivalent. Moreover we say that $v$ *predicts* $w$ if $v$ maps to $w$ and no other vertex of $G_R$ maps to $w$. We can generalize those notions to arcs and graphs, where an arc $(v_1.v_2)$ of $G_R$ maps (resp. predicts) an arc $(w_1, w_2)$ of $G_S$ if $v_1$ maps to (resp. predicts) $w_1$ and $v_2$ maps to (resp. predicts) $w_2$. Finally, the graph $G_R$ *correctly predicts* $G_S$ if those two graphs have the same number of vertexes and arcs and each vertex/arc of $G_R$ predicts a vertex/arc of $G_S$.

In all experiments, the accuracy of our method is evaluated by two standard measures, *Sensitivity* (Sn) and *Positive Predictive Value* (PPV) considered at vertex and arc level.

Sensitivity is defined as the proportion of vertices (or arcs) of the isoform graph that have been correctly predicted by a vertex (or arc) of the computed splicing graph, while PPV is the proportion of the vertices (or arcs) of the splicing graph that correctly predict a vertex (or an arc) of the isoform graph. In the specific the sensitivity is defined as:

$$Sn = \frac{TP}{TP + FN}$$

and the positive predicted value as:

$$PPV = \frac{TP}{TP + FP}$$

where the *True Positives* ($TP$) are the vertexes (resp. arcs) that have been correctly predicted, the *False Negatives* ($FN$) are the correct vertexes (resp. arcs) that have not been correctly predicted and the *False Positives* ($FP$) are the not correct nodes (resp. arcs) that are predicted (as if they were).

The goal of the first experiment (each gene separately, full coverage) is to show the soundness of our approach, since obtaining satisfying results under full coverage is a requisite even for a prototype. More precisely our implementation has perfectly reconstructed the isoform graph of 43 genes (out of 112), that is Sn and PPV are 1 both at vertex and arc level. Notice that the input instances are usually not good instances, mostly due to the presence of short blocks or relatively long repeated regions, therefore we have no guarantee of being able to reconstruct the isoform graph. Moreover we have obtained average Sn and PPV values that are 0.86 and 0.92 at vertex level, respectively, and 0.72 and 0.82 at arc level, respectively. Also, the median values of Sn and PPV are 0.91 and 1 at vertex level, 0.83 and 0.93 at arc level, respectively.

The second experiment (separated gene, 8x coverage) is to study our method under a more realistic coverage. In this case, we have perfectly reconstructed the isoform graph of 39 genes (out of 112), and we have obtained average Sn and PPV values that are respectively 0.87, 0.91 at vertex level and 0.75, 0.81 at arc level. The median values of Sn and PPV are 0.93 and 1 at vertex level, 0.84 and 0.91 at arc level, respectively.

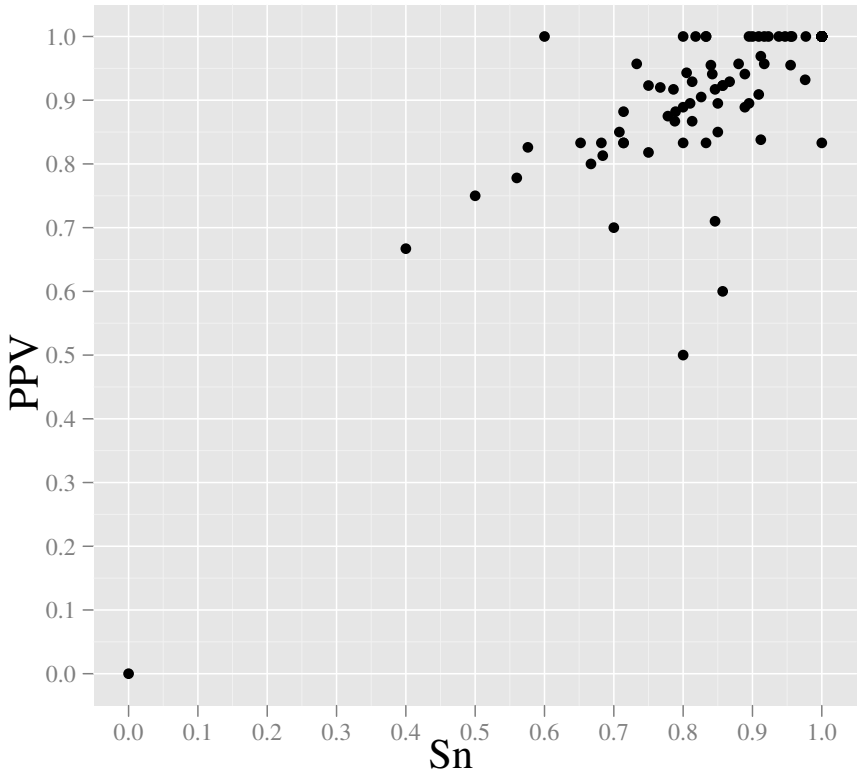
The details of the first two experiments are reported in Appendix A. More specifically, Table A.1 show the values of the 112 genes obtained in the first experiment, while Table A.2 reports the same values for the second experiment.

The graphics in Figure 3.6 give a detailed overview of the quality of the predictions in the first two experiments, in which each point correspond to a gene. More specifically, for each gene in the considered dataset, the Sn versus PPV values are plotted (Sn on the $x$ axis and PPV on the $y$ axis), for both nodes and arcs. Figures 3.6(a) and 3.6(b) show

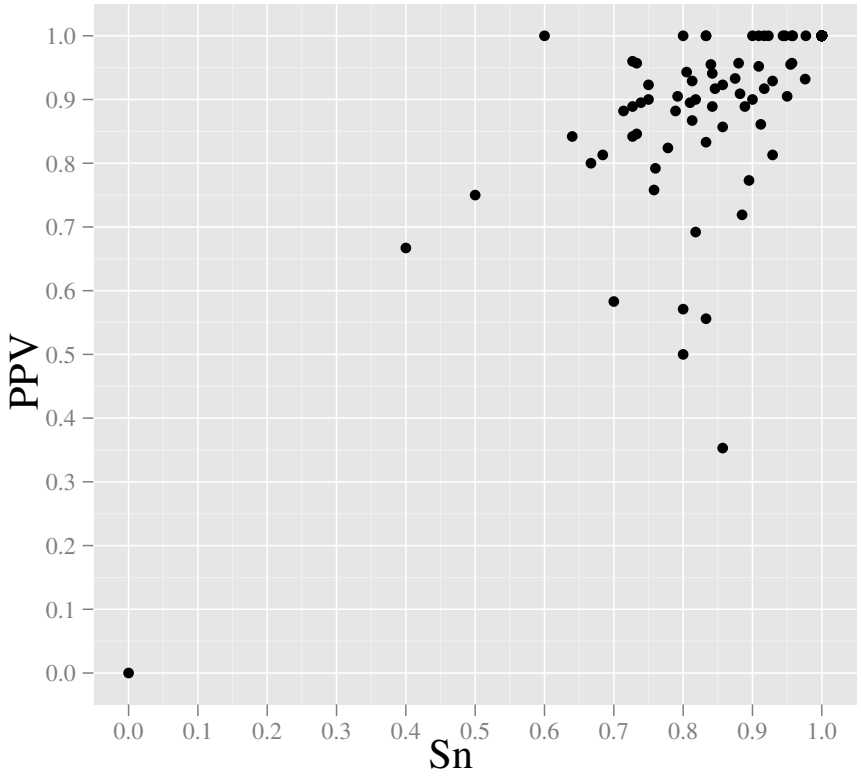
these values at node level in the first two experiments, underlining the goodness of the predictions. In fact, as it is possible to notice, the points are concentrated in the right-upper corner, which corresponds to values of Sn and PPV close to 1. In Figures 3.7(a) and 3.7(b) the same results at arc level are shown. In this latter case the points are more scattered in the graphics, with respect to the previous plots, but it is important to notice that the same "distribution" is conserved in both the graphics. This means that the reduction in the coverage does not have a big impact on the results of the prediction. The same holds for the nodes, in fact, also in this case, the distribution is the same in the graphics corresponding to the two experiments.

The main goal of the third experiment (*whole dataset*, full coverage) is to start the study of the scalability of our approach towards a genome-wide scale, determining if repetitions that occur in different genes are too high obstacles for our approach. A secondary goal is to determine if our implementation is stable, that is the computed splicing graph is not too different from the disjoint union of those computed in the first experiment. In this experiment the expected output of is a large isoform graph $G_I$ with 1,521 vertices and 1,966 arcs, with a 1-to-1 mapping between connected components and input genes. To determine such mapping, we have used a strategy based on BLAST [38], aligning labels of the vertexes and the genomic sequences. In particular, we have mapped all the obtained sequences (i.e. the nodes) into the original transcripts, keeping only the best matches reported by BLAST. After that, each connected component was considered separately, by checking the alignment of all its nodes, in order to assess the correct gene. At the same time, the connected component is inspected by a traversal to obtain the information on nodes and arcs. Finally, the identified structure is compared with the correct one (i.e. the one of the correct isoform graph). We report that only 7 connected component have been mapped to more than one gene – 4 of them are genes with very similar sequence composition (i.e. CTAG1A, CTAG1B and CTAG2).

In practice, such a 1-to-1 mapping exists for almost all components. Moreover only 17 genes have been associated to more than one connected components. Overall results are quite similar to those of the first experiment. In fact, the number of correctly identified vertices goes from 1,303 (first experiment) to 1,274 (third experiment). Similarly, the number of correctly identified arcs goes from 1,415 to 1,396 – the quality of our predictions is only barely influenced by the fact that the program is run on the data coming from 112 different genes. The overall vertex sensitivity is 0.837, the vertex positive predicted value is 0.778, while the arc sensitivity is 0.71 and the arc positive predicted value is 0.679. Figure 3.8 shows the isoform graph and the predicted graph for the gene POGZ.

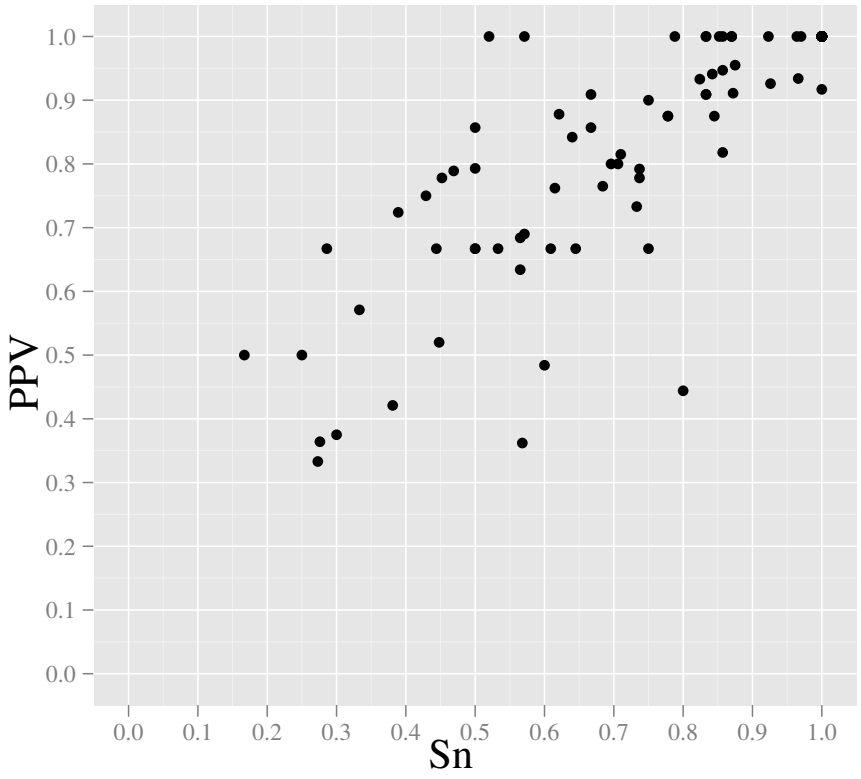



(a) Nodes First Experiment.

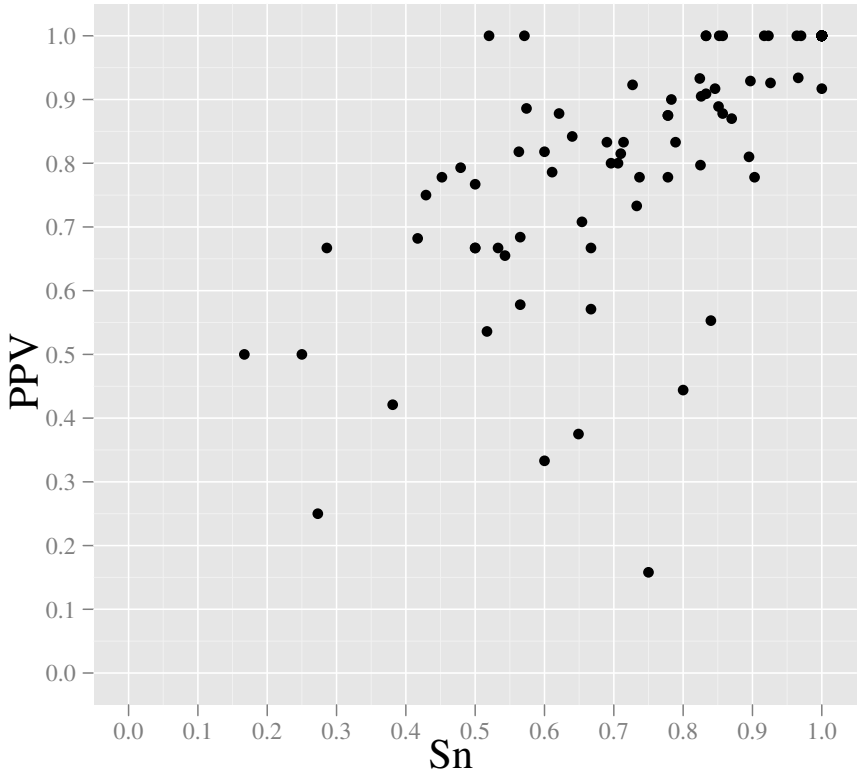



(b) Nodes Second Experiment.

FIGURE 3.6: Sn and PPV values at vertex level in the first 3.6(a) and in the second 3.6(b) experiments. Each point represents a gene.




(a) Arcs First Experiment.




(b) Arcs Second Experiment.

FIGURE 3.7: Sn and PPV values at arc level in the first 3.7(a) and in the second 3.7(b) experiments. Each point represents a gene.

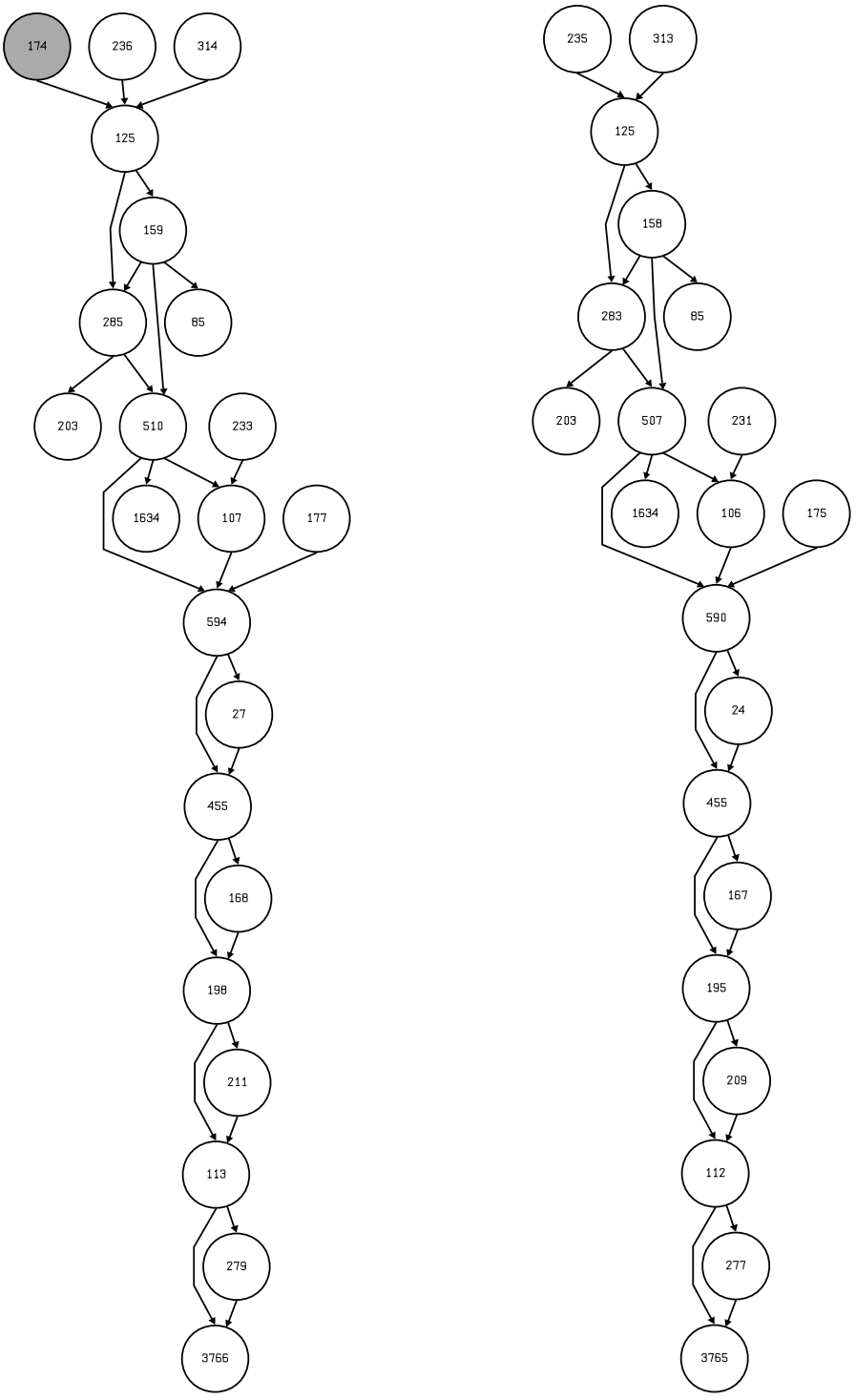



FIGURE 3.8: Gene POGZ. The correct isoform graph (on the left) and the predicted isoform graph (on the right) predicted in the third experiment. Number in the nodes represent the lengths of the blocks. The difference between the two graphs consists of the gray vertex that is missing in the predicted graph.

The final part of our analysis is a comparison with Trinity [25] – the most advanced available tool for full-length transcript reconstruction from RNA-Seqs without a reference genome, to determine how much it is stable. We have run Trinity on the two full coverage datasets, corresponding to the first and third experiments. Since Trinity computes transcripts and not the splicing graph, we use the variation of number of predicted full-lengths transcripts as a proxy for the (in)stability of the method. We observed that, for the two datasets, Trinity has predicted 2,689 and 1,694 full-length transcripts (on the genes from which the simulated read are generated, there are 1,020 annotated transcripts). The variation is significant and hints at a desired property of our algorithm that is not shared with other state-of-the-art tools.

## 3.6 Conclusions and Future Work

Next-Generation Sequencing techniques have revolutionized many fields of bioinformatics. In the last few years new challenges have emerged, especially in trascriptomics, where these new sequencing methods have opened new research directions. Due to its importance in many biological processes, alternative splicing is one of the most studied topics from both a computational and a functional point of view. In this thesis we have given contributions to the computational problem of predicting alternative splicing from NGS data.

In particular, we have faced the computational problem of reconstructing splicing variants from RNA-seq data in absence of a reference genome. To accomplish this task we have propose a graph representation of expressed isoforms and alternative splicing events in a sample and we have provided an algorithm for building such a graph for each gene that works in linear time in the size of the input data.

This work is focused on the analysis of this data structure and in particular on the identification of the conditions under which it is possible to construct a graph, called *isoform graph*, starting from RNA-Seq reads. More specifically, we have identified the necessary and sufficient conditions for a correct reconstruction of the graph in absence of a reference genome. We have also proposed a new efficient algorithm to compute isoform graphs that is particularly suited for elaborating RNA-Seq data in absence of a genomic sequence. We have proved that, under some assumptions, our algorithm, which is linear in the number of input reads, correctly computes the graph. We have shown that also when the conditions are not satisfied, our algorithm is able to reconstruct an approximation of the correct isoform graph, that represents the same splicing events. To do this, we have studied each condition singly, in order to verify that the algorithm is capable of finding the "correct" solution. At the same time, an extensive experimental

analysis on simulated data has confirmed the soundness of our approach and its efficiency. In order to verify the accuracy of the method, but also its stability and scalability, we have designed 3 experiments. The first 2 are performed by considering each gene separately, while in the third one, all the reads of the genes are mixed. In the first experiment we have tested the ability of the algorithm in reconstructing the graph in optimal conditions, when the coverage is high. In the second one, we have reduced the coverage in order to verify its predictions in conditions that are closer to the real ones. In the last experiment, we have tested the stability of the algorithm and also its scalability with an increased number of reads, by mixing reads coming from different genes.

Our theoretical analysis has identified cases that can be problematic for our approach – for instance because of short exons or relatively long repeated regions in the isoforms. Future work will be devoted to address those specific problems that do not allow the construction of a completely correct isoform graph.

Moreover, future work would be also devoted to the integration of the novel algorithm into the tool PIntron, which is a software for predicting alternative splicing from ESTs and transcripts data [39]. In fact, combining predictions of AS from traditional data with ones from NGS data would greatly improve our knowledge of alternative splicing phenomena in human genes. Moreover, the integration would be useful for the routine work of molecular biologists.

This work opens new research directions on alternative splicing investigation that take advantage of the graph structure, used to represent the alternative splicing variants. One research direction concerns the differential analysis of alternative splicing from NGS samples. More specifically, it is possible to design a procedure to compare graphs coming from different RNA-Seq experiments, in order to assess the differences in the structures. This is useful to study, for example, sick organisms in comparison with healthy ones, to check if there are changes in the alternative splicing variants of the two organisms. In addition to this, it is possible to use the same approach to monitor diseases, in particular if they affect the alternative splicing mechanisms, by adding or removing variants. Moreover, the most relevant events can be detected and used as reference in the comparison of the different experiments.

Another research direction concerns the use of RNA-Seq data for the detection of diseases that specifically affect alternative splicing events directly, without the assembly of full-length transcripts or the use of a reference. A useful development, which is closely related to the previous one, regards the extraction of alternative splicing events from the produced isoform graphs. In fact, recalling from Section 3.2, all the main alternative splicing events, such as exon skipping, mutually exclusive exons or alternative splicing sites, can be represented in the isoform graph as specific subgraphs. Inferring

these events could support the previous comparison and confirm the annotations of the considered genes.

This technique can be also used to inspect organisms for the first time, when obviously the genomic sequence is not available, in order to have a preliminary overview of the transcript variants of the sequenced genes. The produced graph can be used to support more specific studies based on the comparison with other organisms. In general we want to explore the possibility of not using a reference sequence for the analysis of RNA-Seq data and for comparing unknown transcripts.

Finally, we would like to remark the importance of having a convenient representation of all the alternative splicing variants of a gene. This graph structure, provided by the isoform graph, avoids all the problems related to the reconstruction of the isoforms and can be incorporated into a pipeline of transcript analysis to support and confirm some steps from a different point of view.

# Chapter 4

# Taxonomic Assignment to Multiple Trees in Metagenomics

The term *metagenomics* is used to refer to the study, at the genetic level, of the organisms present in an environmental sample. It is a powerful tool that can be used to analyze microbial communities, including the ones that can not be cultured in the laboratory. The aims of such a study are to characterize the different organisms present in a specific sample or also to group them, based on similarities. This is now possible due to the recent introduction of the NGS techniques, that allow to have information from all the genetic material of a sample. In this way, in metagenomic experiments, the reads are obtained by all the organisms present in the sequenced environment. One of the main tasks in a metagenomic analysis is the correct assignment of the reads (usually coming from a sequenced sample) to a reference taxonomy, in order to identify the species present in the sample and also to classify them. A common strategy to do this, is to align the input reads to the sequences of a database (that usually represents the species of the taxonomy) and then assign those reads to the correct rank. This is not a trivial operation for reads that match multiple sequences in the reference database with the same confidence in the alignment. There are several approaches to solve this problem that usually rely on the tree structure of the reference taxonomy.

TANGO [40] is an example of program that assigns reads into a reference taxonomy based on a parameter $q$ that influences a penalty score calculation for each of the possible candidate representatives of the taxonomy. In the specific, by choosing values of $q$ that are close to 0, the assignments are guided to lower ranks of the taxonomy (that are usually species). On the contrary, by choosing values of the parameter that are close to 1, the assignments are guided to higher ranks (close to the LCA of the matched sequences). This method introduces more flexibility than other approaches that always assign a

read to the LCA of the possible matches, since in this latter case there could be some descendants that are not in the match set (that are indeed considered by the penalty score calculation in TANGO). Although this approach performs better than other LCA-based methods, there is a common problem shared among all the taxonomy-based approaches: the taxonomic assignment (obviously) depends on the chosen taxonomy. This means that by changing the taxonomy, the assignment can change too, in particular if the two taxonomies do not agree. This fact has motivated our work. More specifically, our objective is to provide a support to different taxonomies used for the taxonomic assignment with TANGO, allowing the user to choose among them. A possible solution to this problem is to develop a mapping among taxonomic trees that gives the possibility to transfer the assignment on a tree into another one. This also avoids recomputing the assignment on the new tree. Furthermore, we have decided to improve the TANGO method both in the algorithm and the implementation, adding the possibility to manage different taxonomies. Although it is necessary to recompute the assignment to bring the results from a tree to another one, the improved version of the program is sensibly faster than the previous one, justifying this decision. In fact, we have made a new implementation of the software, based on different data structures (with respect to the ones used in the actual version) and it uses a new algorithm for the calculation of the penalty scores of the candidate nodes of the taxonomic tree. This latter algorithm is more efficient than the previous one, since it allows to obtain the same calculation of the penalty score function on a different tree, which is "smaller" than the one actually used (see Section 4.3). This optimal solution led us to develop a procedure that finds the minimum penalty score value (i.e. the correct taxonomic assignment) by looking at a subset of nodes. In addition to this, it also offers the possibility to perform the assignment on a different taxonomy with respect to the one used for the alignment of the reads.

In this chapter we start by presenting the main available taxonomies for the bacterial 16S rRNA used in metagenomic studies, analyzing the structures and the data formats. In fact, we are particularly interested in comparing them, in order to realize a mapping of nodes, that will be used to convert the input among different taxonomies. We have found some disagreements among tree representations of the most used reference databases. For this reason, we have implemented a function equivalent to the `merge_lineage` procedure[1] of the bioperl project to construct taxonomic trees starting from the lineages of the organisms, with the precise objective of doing the taxonomic assignment.

From this point of view, we add an intermediate step of tree contractions, that is useful to reduce the taxonomies to the same taxonomic ranks. Moreover, in the improved version of TANGO, we have added the support to different reference databases, offering more

[1] `http://doc.bioperl.org/bioperl-live/Bio/Tree/TreeFunctionsI.html#POD7`

flexibility in the choice of the taxonomy on which to perform the assignment. The other main point of this work is the realization of an optimal procedure to find the minimum value of the penalty score function, used in the taxonomic assignment (by TANGO). This latter procedure is based on a data structure, called skeleton tree, which is induced by the set of matches of a read, in the taxonomy. We use such a structure to compute the penalty score of the nodes in the tree, in an optimal way (see Lemma 4.5). Before that, we also prove its correctness in the calculation of such a function (see Lemma 4.4). The skeleton tree is based on the calculation of the LCA among pairs of nodes, which can be done in $O(1)$ time, after a pre-processing of the tree. We take advantage of this fact to realize the computation of the taxonomic assignments of the reads, in the taxonomy. After this, we describe the new data structures used for the representation of the tree and the new algorithm based on the skeleton tree, that are part of the improved version of TANGO. We also detail some implementing aspects, used for the realization of this task.

The research described in this chapter was carried out during my period abroad as a visiting student at the *Algorithms, Bioinformatics, Complexity and Formal Methods Research Group of the Technical University of Catalonia in Barcelona, Spain.* The work was realized in collaboration with Daniel Alonso-Alemany under the supervision of Prof. Gabriel Valiente.

## 4.1 Introduction to Metagenomics

Metagenomics is usually referred to the study of genetic material coming from an environmental sample [41]. Typically, in these samples are present communities of microbial organisms in their native environment that makes the sequencing difficult, expensive, or simply impossible. In fact, in many cases it is not possible to separate the organisms and sequence them individually. Studies of metagenomics are necessary for the discovery of novel genes and they can also help in the identification of novel species. These latter results give a better understanding of the composition of the microbial communities. Some closely related problems in this field are to assess the diversity and estimates the abundances of organisms present in an environmental sample. To do this, the reads are usually obtained from the 16S ribosomal RNA (rRNA) of the organisms and then they are compared with known databases (by clustering or assigning them to a reference taxonomy). The motivation is that the 16S rRNA genes (that have the length of about 1500bp) are conserved among organisms within a species while diverging across species [42, 43].

Taxonomies are usually represented as $n$-ary trees in which each depth correspond to a *taxonomic rank*. In this way it is possible to classify organisms and establish relations among them. The usually adopted characterization is based on 7 levels (i.e. taxonomic ranks), that are:

$$\text{Kingdom} \Rightarrow \text{Phylum} \Rightarrow \text{Class} \Rightarrow \text{Order} \Rightarrow \text{Family} \Rightarrow \text{Genus} \Rightarrow \text{Species}$$

that are usually abbreviated as *KPCOFGS*. In some cases further ranks have been introduced to refine the classification, like for example *sub* or *super* ranks (e.g. subfamily or superclass). Following the previously introduced characterization, species are usually the leaves of the tree (see Figure 4.1 for an example). The classification levels become more specific descending from the top to the bottom. In fact, there are many organisms that belong to the same kingdom, fewer that belong to the same phylum, and so on, with species being the most specific classification. For example humans are classified in the following way:

| | |
|---|---|
| Kingdom: | Animalia/Metazoa |
| Phylum: | Chordata |
| Class: | Mammalia |
| Order: | Primates |
| Family: | Hominidae |
| Genus: | Homo |
| Species: | Homo sapiens |

With the advent of NGS technologies it is now possible to have a direct access to some microbial species and to small genes that were not available with the previous techniques. In the specific, some genes that were small enough to be entirely contained into Sanger reads, can be now isolated with the produced short sequences. The 454 technology is the most widely used technique for metagenomics analysis (among all the previously introduced next-generation sequencing methods) [44]. The advantage of this method is that it produces sequences with length shorter than the ones produced by Sanger technique but these reads are long enough to identify genes. In addition to this, due to its high throughput, it helps in uncovering numerous new species with low abundance in an environmental sample. On the other hand, sequencing machines like the Illumina Genome Analyzer can achieve an even higher throughput, but the produced sequences are much shorter (see Section 2.3.2). This generates more uncertainty in the metagenomic analysis. For this reason, the adopted sequencing technology has a main role in the species accessibility and also in the resolution that can be achieved in the analysis.

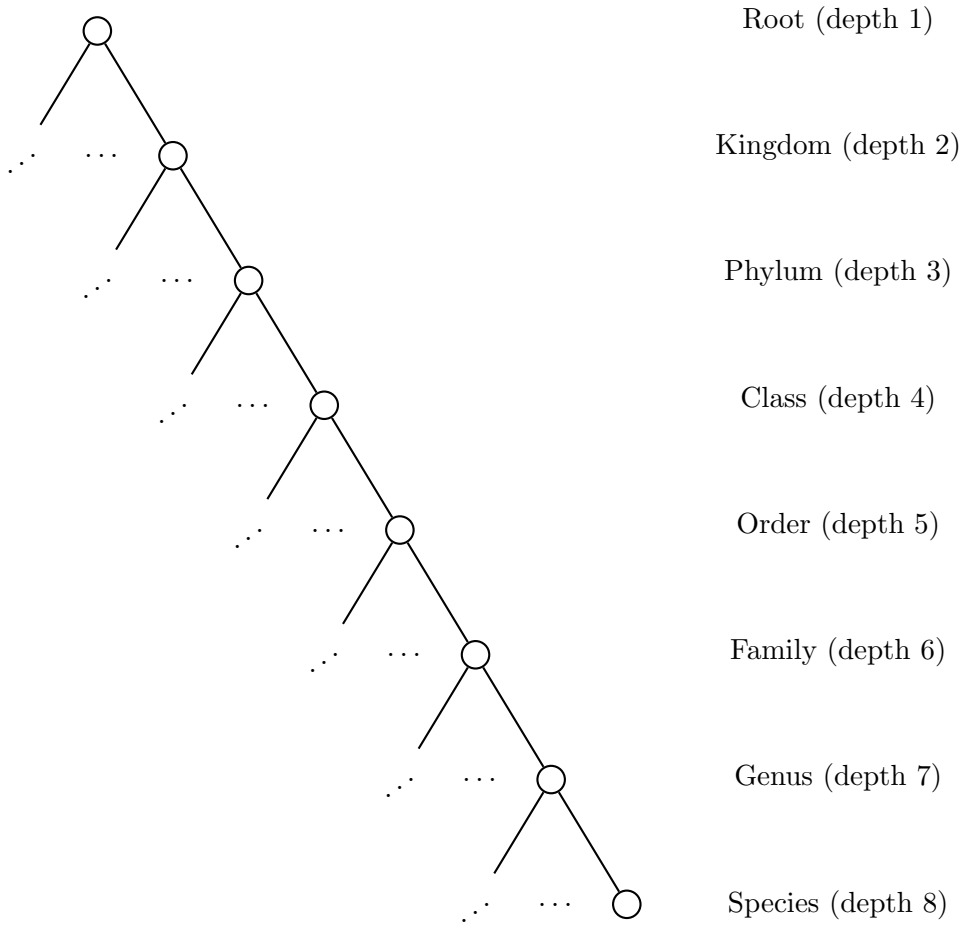



FIGURE 4.1: Example of taxonomic tree. The example shows a rooted tree with maximum depth 8 (7 taxonomic ranks plus the Root), underling a subset of nodes and their associated taxonomic ranks.

As pointed out in [45], in addition to the issues related to the use of NGS data (like short length reads and sequencing errors), there are other factors that make difficult and complex the metagenomic analysis. First of all, the samples can vary a lot in terms of complexities and distributions of the relative abundances. In fact, the number of species present can be different between two samples, influencing also the number of dominant ones. Another factor that must be considered is the way in which the reads are assigned to the taxonomy (in some analysis). There are several methods using different models and algorithms having advantages and disadvantages with respect to various measures of performance. The last but very important factor that influences metagenomics analysis is the taxonomy used as a reference for reads assignment. Since the chosen taxonomy influences all the analysis, so the way in which it is modeled poses constraints on the ability to discriminate or relate some species.

Since the beginning of the application of NGS technologies to metagenomics, several methods have been developed to classify (by aligning to a known reference taxonomy)

metagenomic data. As an example, NAST [43] performs a 16S rRNA reads alignment into a taxonomy (which is composed of a set of 16S rRNA reference sequences of the database). This program filters the reads and then performs an alignment with BLAST [38] against all the reference sequences. As underlined earlier, one of the first computational challenges in metagenomics is the alignment of reads against a taxonomic reference (e.g. database of 16S rRNA sequences of different organisms).

### 4.1.1 Taxonomic Alignment

There are several programs capable of mapping NGS data. Since these aligners have to deal with a huge amount of sequences (millions or billions), the primary requirement is speed. In fact, they can outperform the existing methods (like the previously introduced BLAST) by aligning up to millions of reads per hour. The problem of these approaches arises when considering mismatches because they usually consider only a limited number of differences in the alignment, making them less accurate if compared with BLAST. This latter tool is able to identify big differences between the short sequences and the genome used as reference in the mapping process. For example, by default Bowtie [28] allows 2 mismatched in the "seed" of the first $L$ base pairs of the read (where $L$ is set by default to 28) that corresponds to $\sim 93\%$ of similarity in the seed, and similarly BWA [29], by default, sets its maximum edit distance value to 4%, providing a similarity of 96%. Moreover, although they are usually capable of finding all the possible alignment of a read, the default behaviour is to report only one of those mapping; so, in addition to the previous problems, there is also the limitation related to the number of reported hits. This is because in the NGS aligners the performances decrease exponentially if they have to consider more possible solutions, so they have as a default behavior that of reporting one match (the best based on some criteria), becoming much slower if asked to find them all. For example, Bowtie has the possibility to look for all the valid alignments but by default it reports only the first one encountered. However, it is still unknown whether "new" mappers are suitable to address metagenomic problems, in particular when the reference is composed by many sequences very close to each other. This can cause problems for example in the assignment of those reads in the taxonomy because with only one match it is not possible to assess if there are other valid mappings (making its attribution in terms of a taxonomic tree ambiguous) or not.

In [45], some alignment programs are analyzed in order to underline properties related to metagenomics studies. In particular, the mapping of simulated "long" reads (e.g. 454 sequences) and simulated "short" reads (e.g. Illumina sequences) are considered. For the former alignments, BWA/SW [46] (that is a component of BWA still based on the

Burrows-Wheeler transform) and BLAT [47] (that is similar to BLAST) are compared, and for the latter reads the chosen programs are BWA [29] and GEM[48].

The results of such analysis showed that, for the longer reads, although the BWA/SW software is significantly faster than BLAT, this latter program can identify a higher number of correct alignments. Moreover, both these aligners have problems, in fact, the output produced by BLAT is huge when working on NGS data, while BWA/SW reports only one match for read. For the mapping of the shorter reads, GEM, differently from BWA, implements an exhaustive search, reporting all the matches of a sequence up to a specific number of mismatches. However, as pointed out by the authors, the ability of recovering the genomic location that originates the read is not the only important feature as far as metagenomics is concerned. In fact, it is also paramount to be able to correctly estimate and output the exact number of hits, in order to make the successive attribution of the read to the correct part of the taxonomy tree as precise as possible (see Section 4.1.2). From this point of view, BWA makes some misclassifications, and this underlines the importance for metagenomic studies, of having approaches that perform exhaustive searches.

### 4.1.2 Taxonomic Assignment with TANGO

One of the most important tasks in a metagenomic analysis is the assignment of the reads into a reference taxonomy (if available) or to cluster them based on the mapping too. It must be specified that the previous alignment phase is not necessary for all the assignment methods. To this purpose it is possible to distinguish between methods that rely on a taxonomy in order to assign the reads at the correct taxonomic rank using the mapping information and methods that try to group reads into cluster of related species. In this latter category there is the Quantitative Insights into Microbial Ecology (QIIME) [49]. This method does not require a genomic reference and groups reads into *Operational Taxonomic Units* (OTUs) by aligning the reads with different thresholds. These latter values are used to fix the identity among sequences and then for each cluster the longest read is selected as representative by performing a statistical analysis of the $k$-mer frequencies. Another software that makes a *non-taxonomic* assignment is MOTHUR [50] which uses reference sequences to perform a multiple alignment of the reads (by choosing a cutoff threshold to impose the similarity among species) followed by a clustering of those reads with different methods (like nearest neighbor, furthest neighbor or average neighbor). This software has also the possibility of assigning reads into a specified taxonomy (*taxonomic assignment*). Another common strategy to perform this latter task is to assign reads that have multiple hits (i.e. reads that can be assigned with equal significance to more than one species in the reference taxonomy) to

the lowest common ancestor of the set of species that match it. An example of tool that uses this method is MEGAN [51]. In the specific, after the alignment of the sequences with BLAST, the ambiguous reads (i.e. the one that have multiple matches) are assigned to the LCA of all the matched species, in the adopted taxonomy. In the specific, the adopted taxonomy in MEGAN is the NCBI taxonomy [52]. One of the problems in this strategy is that by using the LCA there could be false positives, i.e. there are species in the subtree rooted at the LCA the are not matched in the alignment. In fact, if considering the LCA of the match set of a read, not all of its descendant leaves are in that set. This last consideration motivated the development of TANGO.

TANGO [40] is a tool for the *Taxonomic Assignment in Metagenomics* that uses the LCA to refine the assignment process. From a computational point of view the problem is formulated as follows: given a reference taxonomy tree $T$ with leaf set $L(T)$ (i.e. the subset of nodes of $T$ that are leaves), each $\ell \in L(T)$ is a species and has an associated sequence. Given a set of reads $R$, for each $R_i \in R$ there exists a subset $M_i \subseteq L(T)$ of leaves such that each sequence (associated to nodes) in $M_i$ has at most $k$ mismatches with respect to $R_i$ (which are the set of *matches* or *hits* of an alignment program, see 4.1.1). As anticipated before, the objective is to identify for each $R_i \in R$, with $|M_i| \geqslant 1$, the best representing node in $T$ of all matches in $M_i$. Obviously, the choice for $R_i \in R$ such that $|M_i| = 1$ is trivial (*unambiguous reads*), but this is not true for the ones that have $|M_i| \geqslant 2$ (*ambiguous reads*). To solve this latter problem TANGO implement an algorithm, based on a parameter $q$, that calculate a *penalty score* of the nodes in $T$ (see Algorithm 3).

**Algorithm 3**: TANGO

**Data**: a reference taxonomy $T$, a set $R$ of reads with associated the set of matches $M_i$ for each $R_i \in R$ and the parameter $q \in [0, 1]$.

```
1  Assignments ← [ ];
2  foreach R_i ∈ R do
3      if |M_i| = 0 then
           // No assignment.
4          Assignments[i] ← ∅;
5      else if |M_i| = 1 then
           // The only leaf in M_i.
6          Assignments[i] ← {x ∈ M_i};
7      else
8          calculate the penalty score PS_{i,j} for all the nodes in the subtree T_i of T;
9          Assignments[i] ← {j | j ∈ T, have the smallest value of PS_{i,j}};
10 return Assignments;
```

In the specific, the key point of such algorithm is the calculation of the penalty score ($PS$) for the nodes of $T$. To this purpose, it is necessary to introduce some additional

concepts. Given a rooted tree $T$ and a read $R_i \in R$, let $T_i$ be the subtree of $T$ rooted at the LCA of $M_i$. In this way it is possible to introduce the following notations:

- $T_{i,j}$ = the subtree of $T_i$ rooted at node $j$;
- $TP_{i,j}$ (*True Positives*) = leaves in $T_{i,j}$ that belong to $M_i$;
- $FP_{i,j}$ (*False Positives*) = leaves in $T_{i,j}$ that do not belong to $M_i$;
- $TN_{i,j}$ (*True Negatives*) = leaves in $T_i \setminus T_{i,j}$ that do not belong to $M_i$;
- $FN_{i,j}$ (*False Negatives*) = leaves in $T_i \setminus T_{i,j}$ that belong to $M_i$.

In this way, given a read $R_i \in R$, it is possible to partition the leaf set of $T_i$ (i.e. the subtree induced by the LCA of $M_i$) into the 4 previously introduced subsets, for each node $j$ in the subtree $T_i$ (see Figure 4.2). Since $T_{i,j}$ represents the subtree of $T_i$ induced by the choice of node $j$ as representative of $M_i$, the algorithm selects the node that better represent each read in $R$. This is done by calculating the penalty score value for every node $j$ in the subtree $T_i$ and then by selecting the node that minimize such value. More specifically, given a node $j$ in the subtree $T_i$ and the parameter $q$, the penalty score is defined as:

$$PS_{i,j} = q \cdot \frac{|FN_{i,j}|}{|TP_{i,j}|} + (1-q) \cdot \frac{|FP_{i,j}|}{|TP_{i,j}|} \tag{4.1}$$

For all the nodes $j$ of $T$ that are not in $T_i$, $PS_{i,j} = \infty$. Also in the case in which $|TP_{i,j}| = 0$, $PS_{i,j} = \infty$. So, the representative of $M_i$ for every read $R_i \in R$ is selected as the node $j$ in $T_i$ such that the value of the penalty score $PS_{i,j}$ is minimum, with respect to the parameter $q$.

It is easy to notice that the choice depends on the value of the parameter $q \in [0, 1]$. In the specific, if $q = 0$, then:

$$PS_{i,j} = \frac{|FP_{i,j}|}{|TP_{i,j}|}$$

This implies that the minimum value of the penalty score is obtained by minimizing the false positive set. For this reason, the optimal assignment is done by selecting a leaf in $M_i$ for which the value of $|FP_{i,j}|$ (and consequently the value of $PS_{i,j}$) is equal to 0. On the other hand, if $q = 1$, then:

$$PS_{i,j} = \frac{|FN_{i,j}|}{|TP_{i,j}|}$$

This implies that the minimum value of the penalty score is obtained by minimizing the false negative set. This means that the optimal assignment is done by selecting the LCA of $M_i$ (i.e. the root of $T_i$) for which the value of $|FN_{i,j}|$ (and consequently the value

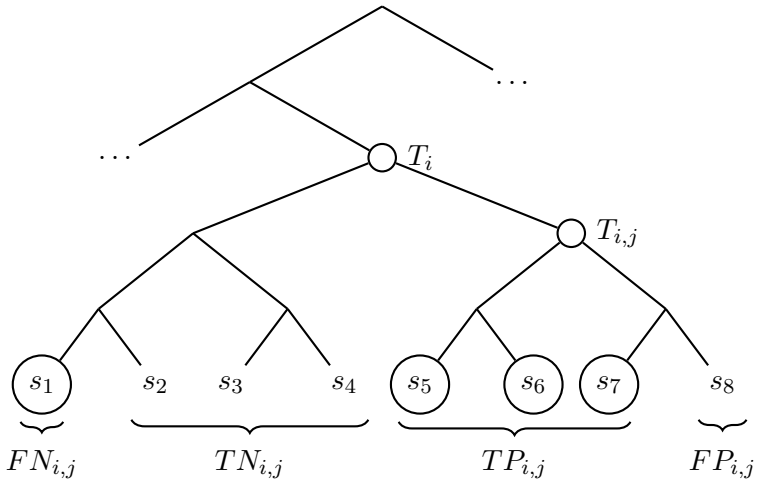



FIGURE 4.2: Leaf set partition used for the *Penalty Score* calculation in TANGO. In this example, given a read $R_i \in R$ the matches are represented as circled leaves, i.e. $M_i = \{S_1, S_5, S_6, S_7\}$. The leaves of $T_i$, which is the subtree rooted at the LCA of $M_i$, can be partitioned into 4 disjointed subsets ($TP_{i,j}$, $FP_{i,j}$, $TN_{i,j}$ and $FN_{i,j}$) for each node $j$ in $T_i$. $T_{i,j}$ represents the subtree of $T_i$ induced by the choice of node $j$ as representative of $M_i$.

of $PS_{i,j}$) is equal to 0. To summarize, the value of $q$ influences the choice of the best assignment in the following way: if $q = 0$ then the representative is selected among the leaves in $M_i$; if $q = 1$ the representative is the LCA of $M_i$; for all the values $0 < q < 1$ the representative is an internal node of the subtree $T_i$. To be more precise, in [40] the authors have shown that by selecting $q = 0.5$, the process of selection is equivalent to maximize the *F-measure* ($F_{i,j}$) which is defined as a combination of precision ($P_{i,j}$) and recall ($R_{i,j}$):

$$F_{i,j} = \frac{2 \cdot P_{i,j} \cdot R_{i,j}}{P_{i,j} + R_{i,j}}$$

Finally, the best representative node for an *ambiguous read* can be computed in an efficient way in $O(|T_i|)$ (i.e. the number of nodes in the subtree $T_i$) time, and, if a $O(|T|)$ time preprocessing is performed, it is possible to find the best $j$ in $T$ that minimizes the value of $PS_{i,j}$ for any $M_i \subseteq L$ in $O(|M_i|)$ time. An example of taxonomic assignment with TANGO is shown in [53].

## 4.2 Taxonomy Analysis

Metagenomics studies are mainly focused on bacterial communities, and the interest is due to the fact that Bacteria have important roles in the life of organisms. Several reference databases and taxonomies have been built in order to classify and determine the sequence and the relationship among Bacteria. To achieve this purpose, selected regions of the 16S ribosomal RNA (rRNA), that constitute optimal species molecular

| Database | N. Sequences (Taxon) |
|---|---|
| NCBI | 2,159,030 (Bacteria) |
| | 20,150 (Archaea) |
| RDP | 1,052,807 (Bacteria) |
| | 20,408 (Archaea) |
| Greengenes | 400,259 (Bacteria) |
| | 6,738 (Archaea) |
| SILVA | 629,125 (Bacteria) |
| | 38,721 (Archaea) |
| LTP | 8,931 (Bacteria) |
| | 348 (Archaea) |

TABLE 4.1: List of the main databases for 16S ribosomal RNA (rRNA) and the number of Bacteria and Archaea (high-quality) sequences present in the current version of the database.

markers, are usually sequenced (see [54] for a more detailed analysis about the reference databases used in metagenomics). Although the NCBI database is a huge source of sequence information, its content can have low quality and also erroneous sequence data. In addition to this, the amount of submitted data is continuously growing and this has contributed to the development of the previously mentioned reference databases. In fact, due to a so huge quantity of data, the quality and the annotation of such sequences are difficult of assess. For this reason, more specific and accurate resources have been developed. Currently, the main available reference databases for Bacteria and Archaea (reported in Table 4.1) are: *NCBI taxonomy*, *RDP*, *Greengenes*, *SILVA* and *LTP*.

### 4.2.1 16S Databases

**NCBI** taxonomy [52] is the most used database that provides a classification and a nomenclature for all the organisms present in it. At the moment of the writing of this thesis, there are 11,585 prokaryotic (of which 454 are Archaea and 11,131 are Bacteria) and 245,949 eukaryotic species in the NCBI database. This taxonomy is updated with user submissions that are confirmed by the literature, and all of this is done manually. The NCBI taxonomy is available at `http://www.ncbi.nlm.nih.gov/taxonomy/` and there is also a *ftp* site that includes dumps, to download such a taxonomy. Although this latter reference database is not specific for bacterial organisms, it is however used for that purpose (i.e. 16S based analysis).

The *Ribosomal Database Project* (**RDP**) [55] is the second example of reference database used in metagenomics. The current version of such taxonomy, which is RDP 10.29 (Release 10, Update 29), consists of 2,320,464 aligned and annotated 16S rRNA sequences. More specifically, these sequences are divided into Bacteria (which consists of 2,212,243

sequences) and Archaea (which consists of 107,969 sequences), plus 252 unclassified sequences. This taxonomy (which also provides a set of tools for taxonomic analysis) is available `http://rdp.cme.msu.edu/`.

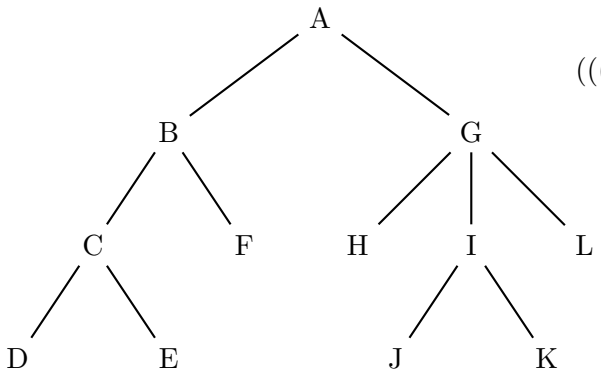
**Greengenes** [56] is a dedicated full-length 16S rRNA gene database that provides users with a curated taxonomy. This taxonomy consists of 1,049,116 aligned sequences with a length that is greater than 1,250nt and is based on a *De novo* phylogenetic tree calculated from sequences (previously filtered by quality). In this way, it is possible to infer relationships among the considered sequences. It is available for download and browsing at `http://greengenes.lbl.gov/`. It also provides an automatic tool for assigning names to novel (unclassified) clusters of sequences.

**SILVA** [57] database provides a classification for both *small subunits* (SSU), like the 16S rRNAs, and *long subunits* (LSU) quality checked rRNA sequences for all three domains of life (which are Bacteria, Archaea and Eukarya). The actual version (111) of SILVA, released in July 2012, has more than 3,500,000 available sequences (SSU and LSU). For the 16S rRNA it is possible to distinguish the *SSU Parc*, which contains only a quality filtered sequences and the *SSU Ref*, which contains high quality sequences (with length above 1,200nt for Bacteria and 900 for Archaea). In the current version of the taxonomy, which is available at `http://www.arb-silva.de/`, there are 3,194,778 SSU Parc (of which 2,651,771 are Bacteria and 129,147 are Archaea) and 739,633 SSU Ref (of which 629,125 are Bacteria and 38,721 are Archaea) sequences.

"The All-Species Living Tree" Project (**LTP**) [58–60] provides phylogenetic classification for the organisms that appear in the *Systematic and Applied Microbiology journal* after an accurate validation. The current version of the LTP taxonomy (which is the s108 of July 2012) contains 8,931 Bacteria and 348 Archaea 16S rRNA sequences (from SILVA database). This taxonomy is available at `http://www.arb-silva.de/projects/living-tree/`.

### 4.2.2 Tree Analysis

For each of the introduced databases, we have downloaded the taxonomy, which is usually represented as a tree, in order to analyze its structure and its nomenclature. It is important to underline that 16S databases do not always agree in the names used for the identification of the species, nor they always agree in the taxonomic structure. We have analyzed such taxonomies in order to provide a support for some of them in the improved version of the TANGO software.




*Newick* format:

$((((D,E)C,F)B,(H,(J,K)I,L)G)A);$

FIGURE 4.3: Example of tree with its representation in *Newick* format.

As a first step we have downloaded the taxonomic trees of the previous 16S databases, that for all but the NCBI, are available in *Newick* format. This latter format is widely adopted for the representation of trees, using parenthesis and commas (in its simplest definition). It also allows to add weights (that in the case of taxonomic trees can represent the evolutionary distances) on the edges. Interior nodes are represented by a pair of matched parentheses. Between them there are the representations of nodes that are immediately descended from that node, separated by commas. Finally, the names of the interior nodes follow the right parenthesis of that interior node. In this way, all the taxonomic trees can be represented by nested parenthesis as shown in Figure 4.3. On the other hand, instead of using this latter format, the NCBI taxonomy is available in *dump* format. In this case, it is used a column representation of the tree, i.e. each node is represented by an identifier and the identifier of its parent (which is unique in a tree). In this way, in the first column there are the identifiers of the nodes and in the second one the identifiers of the parents of those nodes. It is also possible to add further columns to provide additional information.

Once obtained the trees we have analyzed their structure by performing a traversal that visits all the nodes. In the specific, taxonomic trees are $n$-ary trees in which leaves are usually labeled with species (or with sequences representing them) and internal nodes can be labeled with name of higher ranks (see Figure 4.1). However, from the performed analysis, we have discovered that 3 (Greengenes, SILVA, LTP) of the 5 considered taxonomy are binary. This is probably due to a "binarization process" in which every internal node is "divided" into sub-nodes with two children each. In the specific, for each internal node $i$ with $n$ children, its leftmost child (assuming that children are ordered) is maintained as it is and the remaining $n-1$ children are moved as descendants of a created right child. At this point, this latter node (which is the right child of $i$) has the $n-1$ children of $i$ as its children. Then this procedure is repeated ($n-1$ times) until each node has two children (see Figure 4.4 for an example). For this reason, the "new" produced tree has more internal nodes than the original one, but more important the

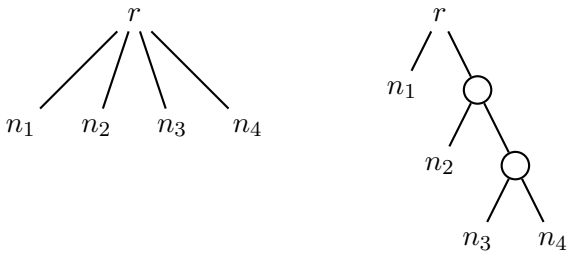



FIGURE 4.4: Example of "binarization process". On the left there is an example of tree in which a node $r$ has 4 children (named $n_1$, $n_2$, $n_3$ and $n_4$). On the right there is the same tree after the binarization, in which the circled node are the ones added in the process.

depth of the nodes (as also the depth of the tree) changes drastically. In addition to this, it is not easy to "de-binarize" (perform the reverse process) in order to obtain the original tree because it is not always possible to identify the added nodes and the others. In fact, since in a taxonomic tree the internal nodes could not be labeled, they can't be distinguished from the new ones that must be contracted to recover the original tree.

The results of the analysis that we have performed on the considered taxonomy trees, are reported in Table 4.2. In particular, the maximum depth of the tree is calculated starting from the root (which has depth = 1) and looking for the farthest leaf. As expected, $n$-ary taxonomies have a maximum depth that is some order of magnitude smaller than the one of binary trees. Also LTP, that is two order of magnitude smaller (in the number of leaves and nodes) than NCBI, has a maximum depth that is one order of magnitude greater than this latter taxonomy. An additional confirmation of the fact that Greengenes, SILVA and LTP taxonomies are (perfect) binary, is given by the property that the total number of nodes $|V|$ is twice the number of the leaves $|L|$, minus one: $|V| = 2 \cdot |L| - 1$. This is true for all the three previously mentioned taxonomies. For the "real" $n$-ary trees the maximum depth is higher than the expected (7 rank levels) but this is justified by the fact that NCBI adds more subdivisions and new ranks (for a total of 28 valid ranks), and also the possibility of having (internal) nodes with "no rank". In the case of RDP there are 2 more ranks, with respect to KPCOFGS, which are *subclass* and *suborder*. This leads to a maximum depth of 10: the 7 KPCOFGS ranks + 2 other ranks (subclass and suborder) + the root.

### 4.2.3 Inferring Trees from Lineages

Due to the highlighted problems, we have decided to build the taxonomic trees by inferring them from the *lineages*. In the specific, a lineage is the "complete classification" of an organism, involving all the taxonomic ranks that, in the case of the tree, correspond to the path from the root to that organism. We have done this operation for the

| Database | Tree Type | N. Leaves | Total N. Nodes | Max. Depth |
|---|---|---|---|---|
| NCBI | $n$-ary | 831,261 | 934,329 | 42 |
| RDP | $n$-ary | 1,073,220 | 1,075,623 | 10 |
| Greengenes | binary | 408,135 | 816,269 | 2,479 |
| SILVA | binary | 739,633 | 1,479,265 | 2,795 |
| LTP | binary | 9,279 | 18,557 | 130 |

TABLE 4.2: Statistics on the taxonomic trees. In the specific, the second column reports the type of the tree which can be $n$-ary or binary; the third and fourth columns report the number of leaves and total nodes, respectively; the last column reports the maximum depth of the tree.

three main used databases in metagenomic analysis: NCBI, RDP and Greengenes. For the NCBI taxonomy, we have used the previously mentioned dump file which reports the information about the "node - parent" relationship, allowing us to reconstruct the correct tree. In addition to this, we have used the third column of the file, containing the taxonomic rank, to correctly label the nodes of the reconstructed tree. For the remaining two databases, RDP and Greengenes, we have used a *GenBank* and a FASTA file, respectively. In the former file there is a record for each of the available leaves of the tree, reporting all the information about such species (see `http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord` for details about the format). Among all the fields, in the one named "`ORGANISM`" the taxonomic lineage of such organism is detailed. Each of the lineages in the GenBank file of RDP is a sequence of names (one for each rank of the taxonomy), separated by semicolons, of fixed length. In agreement with the $n$-ary tree in Newick format, the lineages of RDP have length 10 (including the root), reflecting the depth of such a tree. The following example shows the lineage of an organism present in the GenBank file of RDP:

```
ORGANISM  Acidimicrobium sp. Y0018
          Root; Bacteria; Actinobacteria; Actinobacteria;
          Acidimicrobidae; Acidimicrobiales; Acidimicrobineae;
          Acidimicrobiaceae; Acidimicrobium.
```

The first name present in the `ORGANISM` field is the *species* name (which is the leaf name of the tree) and then there is the lineage from the *Root* of the *Genus* level (which is the one before species). In the specific, each position in the lineage correspond to a specific rank in the taxonomy, following the KPCOFGS ranks plus the two introduced subranks (subclass and suborder):

| Depth | N. Nodes (N. Leaves) |
|---|---|
| 1 | 1 (0) |
| 2 | 3 (0) |
| 3 | 295 (252) |
| 4 | 134,128 (134,016) |
| 5 | 102,529 (102,334) |
| 6 | 197,936 (197,559) |
| 7 | 117,188 (115,446) |
| 8 | 1,540,555 (1,540,464) |
| 9 | 22,784 (22,484) |
| 10 | 207,909 (207,909) |
| **Total:** | 2,323,328 (2,320,464) |

TABLE 4.3: Statistics on the reconstructed RDP taxonomy. In the specific, for each depth of the tree, the number of nodes (resp. leaves) is reported.

| | | |
|---|---|---|
| 1 | Root: | `Root` |
| 2 | Kingdom: | `Bacteria` |
| 3 | Phylum: | `Actinobacteria` |
| 4 | Class: | `Actinobacteria` |
| 5 | Subclass: | `Acidimicrobidae` |
| 6 | Order: | `Acidimicrobiales` |
| 7 | Suborder: | `Acidimicrobineae` |
| 8 | Family: | `Acidimicrobiaceae` |
| 9 | Genus: | `Acidimicrobium` |
| 10 | Species | `Acidimicrobium sp. Y0018` |

It must be noticed that there are also cases in which the lineage is truncated at a certain level, meaning that the precise classification for such a species is not available (and so the leaves are at a different depth in the tree). In this way the taxonomic tree can be reconstructed as an $n$-ary tree. In Table 4.3 the number of nodes (and leaves) for each depth of the reconstructed tree of the last version of the RDP taxonomy are reported. As it is possible to notice, the root has 3 children, Bacteria, Archaea and the unclassified organisms (this last with 252 leaves) and there are a total of 2,320,464 leaves (see description of the RDP database in 4.2.1).

A similar process is done for the Greengenes taxonomy. In this case the file containing the information about the lineages is in FASTA format. This latter is a well know format, and is commonly used to represent sequences of nucleotides in which each record is composed of two lines: the header (that starts with `>`) and the sequence. In Greengenes the first field of such format, i.e. the first line of each entry, is used to describe the lineage of an organisms (with its associated sequence of nucleotides in the second line).

The following is an example of FASTA entry of the Greengenes database (which is split on multiple lines to ease the reading):

```
>37 M36507.1 Methanococcus vannielii k__Archaea; p__Euryarchaeota;
             c__Methanococci; o__Methanococcales; f__Methanococcaceae;
             g__Methanococcus; s__Methanococcus vannielii; otu_144
```

More specifically, after the initial symbol there is a numeric identifier associated to the organism followed by its specific name and by its taxonomic classification (which respects the KPCOFGS ranks). Moreover, each taxonomic rank is identified by a prefix added to the corresponding rank name, which is composed of the first letter of the rank (i.e. `k` for Kingdom, `p` for Phylum, and so on) followed by two "underscore" characters. The lineage is then terminated by the *otu* identifier and, when creating the tree, the root node must be added. In the previous example the classification of the organism is:

| | | |
|---|---|---|
| 1 | Kingdom: | `Archaea` |
| 2 | Phylum: | `Euryarchaeota` |
| 3 | Class: | `Methanococci` |
| 4 | Order: | `Methanococcales` |
| 5 | Family: | `Methanococcaceae` |
| 6 | Genus: | `Methanococcus` |
| 7 | Species: | `Methanococcus vannielii` |

In addition to the procedure introduced before for inferring the taxonomic tree from their lineages, we have decided to perform a contraction of such trees in order to have only the KPCOFGS ranks. This *KPCOFGS contraction* is done with a process that is similar to the inverse of the previously described "binarization". It is possible to define the contracted taxonomic tree, given a subset of its ranks, in the following way:

**Definition 4.1.** Let $T$ be a taxonomic tree in which every node $x$ is labeled with its taxonomic rank, i.e. $rank(x) \in Ranks$, and let $Ranks' \subseteq Ranks$ be a set of valid ranks. The *contracted* tree $T'$, derived from $T$ w.r.t. $Ranks'$, is the tree such that $L(T') = L(T)$ and the two trees $T'$ and $T$ have the same root $r$. Moreover, for each node $x \in I(T) \setminus r$, if $rank(x) \in Ranks'$, then $x \in I(T') \setminus r$, and for each edge $(x, y) \in T'$, $y$ is a proper descendant of $x$ in $T$.

Starting from Definition 4.1, we have formulated the general problem as follows.

*Problem* 2. **Taxonomic Tree Contraction**
**Input**: a taxonomic tree $T$ in which every node $x$ is labeled with its taxonomic rank

$rank(x) \in Ranks$ and a set $Ranks' \subseteq Ranks$ of valid ranks.
**Output**: the *contracted* tree $T'$, derived from $T$ w.r.t. $Ranks'$.

This problem can be solved by performing a *postorder traversal* of the tree $T$, which guarantees that, when visiting a node, all its children are already visited. During the visit, if a node has a label that is not among the valid ranks, it is contracted by assigning all its children to the parent node. Otherwise the node is maintained as it is. A special condition holds for both the root node and the leaves, which are both always kept in the contracted tree $T'$. More specifically, let $x$ be a node of $T$, $parent(x)$ is its parent node and $children(x)$ is the set of its children. Algorithm 4 propose a recursive solution to Problem 2. For ease of reading, we have adopted a simple data structure for the tree representation, which is slightly different form the one used in the improved version of TANGO. We remand the reader to Section 4.4.1 for a more detailed explanation of the adopted data structures.

**Algorithm 4**: Tree-Contraction($r$,$Ranks'$)

**Data**: a rooted tree with root $r$ in which nodes are labeled with labels in $Ranks$ and a set $Ranks' \subseteq Ranks$ of valid ranks.

```
1  if r is a leaf then
2      return;
3  foreach x ∈ children(r) do
4      Tree-Contraction(x,Ranks');
   // Root is a special case.
5  if r is the Root then
6      return;
   // Check if the node r has a valid rank.
7  if rank(r) ∉ Ranks' then
       // Node r must be contracted.
8      foreach x ∈ children(r) do
9          parent(x) = parent(r);
10     Delete r from children(parent(r));
11 return;
```

The correctness of Algorithm 4 is guaranteed by the use of a postorder traversal of the tree. In particular, the base case is verified by the leaves and also the root that are always preserved. Then, for each node $x$ the pre-conditions are that all its children are already contracted and this is guaranteed by the properties of the postorder traversal. The post-conditions are that if $x$ has a valid rank the node is maintained, otherwise all its children are moved as children of the parent of $x$. These conditions are obviously verified by the presence of the conditional statement in the algorithm.

In our case, the set $Ranks'$ consists of exactly the KPCOFGS ranks, meaning that the contracted taxonomic trees have all the same rank levels (and also the maximum depth).

A last but fundamental step performed in this "pre-processing" of the taxonomies is the *mapping* of the nodes. As anticipated before, we have decided to do that only for the leaves, in fact, by reconstructing the tree from the lineages, each entry (i.e. each lineage) corresponds to a leaf (this is also why we keep all the leaves in the contraction procedure of the tree). The idea is to map these entries in order to create a mapping file that associates the leaves of a taxonomy to the nodes of another one. As an example, we have mapped RDP and Greengenes databases to NCBI (i.e. RDP $\Rightarrow$ NCBI and Greengenes $\Rightarrow$ NCBI). For the first case (RDP to NCBI) it is possible to extract such information from the GenBank file, while for the second case Greengenes provides an additional file. Although this operation can be easily accomplished by simply reading two files, it must be noticed that there could be a problem. In fact, there is the possibility that one entry (that is a leaf in the first taxonomy) is mapped to an internal node of another taxonomy, causing some problems in the read assignment preformed by TANGO software.

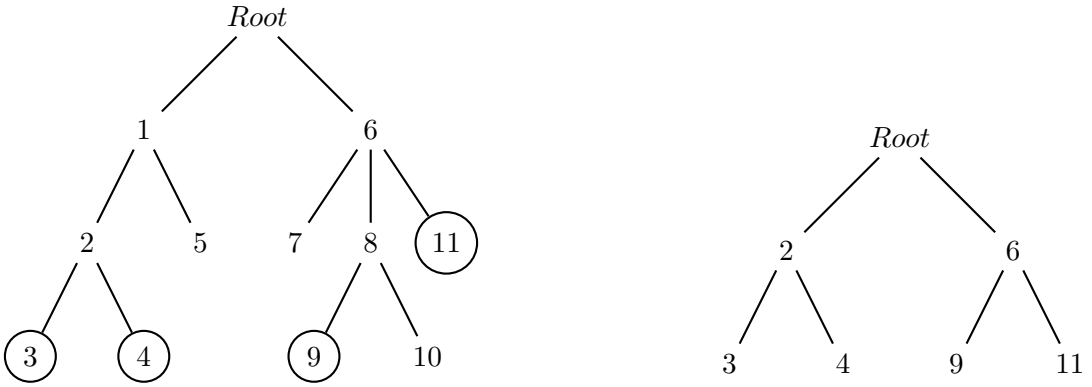
## 4.3 Minimum Penalty Score Calculation

The main problem in the taxonomic assignment in TANGO regards the calculation of the *penalty score* ($PS$). Recalling Section 4.1.2, this function is computed for each node in the subtree induced by the set of matches of a read. For each read $R_i \in R$ with its associated set of matches $M_i$ (i.e. the subset of leaves the match the read $R_i$), the induced subtree $T_i$ of $T$ is the tree rooted at the LCA of $M_i$. Then for each node $j \in T_i$, its penalty score is calculated (based on the chosen parameter $q$) and the node $j$ which has the lowest value is chosen as representative of $M_i$. In other words, the read $R_i$ is assigned to node $j$.

The improvement of this procedure is based on two fundamental observations: (i) the penalty score value can be the same for adjacent nodes and (ii) we want to select a node that is as far as possible from the root. In the specific, the penalty score value changes only if the sets $FN$, $FP$ and $TP$ change, and we want to find a node for which this value is minimum. In this way, it is possible to define the following problem.

*Problem* 3. **Minimum Penalty Score Calculation**
**Instance**: a rooted tree $T$, a set $S \subseteq L(T)$ of leaves and the parameter $q$.
**Solution**: a node $n$ in $T$.
**Measure**: the value of the *penalty score* function, $PS$.
**Goal**: minimization.

Let us recall the Definition 4.1 of the penalty score ($PS$) function. In particular, once the parameter $q$ is fixed, given a node $x \in N(T)$ we want to obtain its penalty score,




(a) Original tree $T$. (b) Skeleton tree $T_S^{LCA}$.

FIGURE 4.5: An example of rooted tree $T$ is shown in 4.5(a), where the circled leaves are the ones in the set $S$, i.e. $S = \{3, 4, 9, 11\}$. In 4.5(b) the skeleton tree obtained from Definition 4.2, starting from the tree $T$ and the subset of leaves $S$.

expressed as:

$$PS(x) = q \cdot \frac{|FN_x|}{|TP_x|} + (1 - q) \cdot \frac{|FP_x|}{|TP_x|} \tag{4.2}$$

where the sets $TP_x$, $FN_x$ and $FP_x$ are respectively the set of true positives, false negatives and false positives, with respect to the node $x$ of $T$.

Instead of computing such a function for all the nodes $n \in N(T)$ to find the minimum value, it is possible to consider only the nodes present in the LCA skeleton tree. In the following we show the correctness of the penalty score calculation by using such a tree and we also describe an efficient way to do it.

### 4.3.1 An Asymptotically Optimal Algorithm

To solve the Problem 3 in a correct and efficient way, we have developed an algorithm based on the *LCA skeleton tree* computation. In the specific, in this section we prove that the procedure involving such a tree to find the minimum value of the $PS$ function is correct and we also describe an optimal procedure to compute it. The LCA skeleton tree was defined in [61] as follows.

**Definition 4.2.** Given a rooted tree $T$ and a subset of its leaves $S \subseteq L(T)$, the *LCA skeleton tree* $T_S^{LCA}$ has node set $N(T_S^{LCA}) \subseteq N(T)$ and edge set $E(T_S^{LCA})$. In particular, a node $n \in N(T_S^{LCA})$ if and only if there exist $x, y \in S$ such that $n = LCA(x, y)$ and an edge $(x, y) \in E(T_S^{LCA})$ if and only if there are no other nodes in $N(T_S^{LCA})$, except for $x$ and $y$, on the unique path from $x$ to $y$ in $T$ (see Figure 4.5 for an example).

Using Definition 4.2, we start by showing the correctness in the use of the LCA skeleton tree for the calculation of a node that has the minimum value of $PS$. In fact, by

computing the $PS$ function only in the nodes of the LCA skeleton tree, it is guaranteed that the minimum value (and consequently a node) will be found. The following lemma states this fact.

**Lemma 4.3.** *Let $T$ be a rooted tree and let $S \subseteq L(T)$ be a set of leaves, and let also $T_S^{LCA}$ be the LCA skeleton tree. Given a node $x \in T_S^{LCA}$, then the same value of the penalty score $PS(x)$ can be obtained in both the trees, $T$ and $T_S^{LCA}$.*

*Proof.* In order to proof this lemma, it is necessary to add an additional information on the tree $T$ (and consequently on the LCA skeleton tree $T_S^{LCA}$). Let us suppose, without loss of generality, that for each node $x \in T$, $nDesc(x)$ is its set of descendant leaves (obtained with a traversal of $T$, which is independent from $S$). This set is kept also for the nodes of $T_S^{LCA}$. Let us recall from Definition 4.2 that $N(T_S^{LCA}) \subseteq N(T)$ and also that the set of descendant leaves of a node $x \in N(T_S^{LCA})$ correspond to $TP_x$. The same set for the node $x \in T$ can be obtained by $TP_x = \{y \in nDesc(x) : y \in S\}$.

At this point, by having for each node $x \in T_S^{LCA}$ the same set $nDesc(x)$ and the same set $TP_x$ in both the trees, the same penalty score value $PS(x)$ can be obtained in both the trees. In particular, the sets $FN_x$ and $FP_x$ can be calculated in the following way:

$$FN_x = S \setminus TP_x \qquad \text{and} \qquad FP_x = nDesc(x) \setminus TP_x$$

As it is possible to notice, the two sets depend on $TP_x$ and $nDesc(x)$ that are the same in both the trees (and also $S$ that is a fixed input parameter). This means that also $FN_x$ and $FP_x$ are the same in the two trees, and consequently the value of $PS(x)$. □

**Lemma 4.4.** *Let $T$ be a rooted tree and let $S \subseteq L(T)$ be a set of leaves. A node $x \in N(T)$ has the minimum penalty score value $PS(x)$ if and only if $x \in N(T_S^{LCA})$ and it has the same minimum penalty score value.*

*Proof.* ($\Rightarrow$) Let us suppose that $x \notin N(T_S^{LCA})$. It is always possible to find a node $y \in N(T_S^{LCA})$ such that $x$ is a proper ancestor of $y$. In fact, the only case in which there are no proper descendant of $x$ that are in $T_S^{LCA}$, is when $x$ has no descendant leaves in $S$. Anyway, in such a case $TP_x = \varnothing$ and so $PS(x) = \infty$. This lead to an absurd. Let us also suppose, without loss of generality, that there are no other nodes $z \in N(T_S^{LCA})$ on the path from $x$ to $y$ in $T$.

From Definition 4.2, $x$ and $y$ have the same set $TP$, because such a set changes only among nodes in $T_S^{LCA}$. It follows that $TP_x = TP_y$. For the same reason, also the set $FN$ must be the same, i.e. $FN_x = FN_y$. In the specific, both $TP$ and $FN$ involve descendant leaves in the set $S$, so these sets can only vary in nodes the are LCA of nodes

in such a set (i.e. nodes of $T_S^{LCA}$). On the other hand, the set $FP$ strictly increase when climbing the tree $T$ (until reaching the next node in $T_S^{LCA}$) by adding leaves that are not in $S$. This means that $FP_x > FP_y$, so $PS(x) > PS(y)$. Recall from Definition 4.2 that $N(T_S^{LCA}) \subseteq N(T)$, so $y \in N(T_S^{LCA})$. This lead to an absurd because from hypothesis $x$ has the minimum penalty score value, so $x \in N(T_S^{LCA})$. In addition, from Lemma 4.3, the penalty score value of $x$ (which is the minimum in $T$) is the same also in $T_S^{LCA}$.

($\Leftarrow$) Let $x \in N(T_S^{LCA})$ be a node with the minimum penalty score value $PS(x)$ in $T_S^{LCA}$. From Definition 4.2, $N(T_S^{LCA}) \subseteq N(T)$, so $x \in N(T)$. Now we have to proof that the value of $PS(x)$ is minimum also in $T$.

Let us suppose that there exists a node $y \in N(T) \setminus N(T_S^{LCA}$ such that $PS(y) < PS(x)$. This means that there exists a node in $T$, but not in $T_S^{LCA}$, that has the minimum penalty score value. As shown in the previous case, it is always possible to find a node $z \in N(T_S^{LCA})$ such that $y$ is a proper ancestor of $z$ and also $PS(z) < PS(y)$. This lead to a contradiction because $PS(z) < PS(y) < PS(x)$ in which $z, x \in N(T_S^{LCA})$, but from hypothesis $x$ has the minimum penalty score value in $T_S^{LCA}$. This means that $x$ has the minimum value of $PS(x)$ also in $T$. □

It has been shown in [62] that, starting from a rooted tree $T$ and a subset of its leaves $S$, the LCA skeleton tree $T_S^{LCA}$ can be computed in $O(|S|)$ time, after having done a pre-processing of $T$ (that is independent of $S$) in $O(|T|)$ time. Moreover, each step of the construction of the skeleton tree can be used to make some calculations. For this reason, we have decided to use the skeleton tree construction, or better its traversal, to compute the penalty score function, in order to find the minimum value. In fact, for this latter function, the tree (the input of the LCA skeleton tree) correspond to the subtree $T_i$ induced by the read $R_i$ (i.e. the subtree of $T$ that is rooted at the LCA of $M_i$) and the subset of leaves correspond to the match set, i.e. $S = M_i$.

We have already proved in Lemma 4.4 that given a tree $T$ and a set $S \subseteq L(T)$ of leaves, to find a node that has the minimum value of the penalty score function can be equivalently done in the LCA skeleton tree $T_S^{LCA}$. In fact, it is guarantee that a node $x$ that has the minimum value $PS(x)$ in a tree, has also the minimum value of such a function in the other tree (correctness property). Now we show how to perform this calculation in an optimal way, by using the LCA skeleton tree.

**Lemma 4.5.** *Problem 3 can be solved in $O(|S|)$ time.*

*Proof.* In the proof of Lemma 4.3 we said that it is necessary to store, for each node $x \in N(T)$, its number of descendant leaves $|nDesc(x)|$. This can be done during the pre-processing of $T$, in $O(|T|)$ time. In fact, by preforming a *postorder traversal* of $T$,

for each node $x \in N(T)$, its number of descendant leaves is the sum of the numbers of descendant leaves of all its children, i.e.

$$|nDesc(x)| = \sum_{y \in children(x)} |nDesc(y)|$$

For the leaves, this value is 1, i.e. if $x$ *is leaf*, then $|nDesc(x)| = 1$.

The algorithm for the construction of the LCA skeleton tree, given a rooted tree and a subset of its leaves, described in [62] requires $O(|S|)$ time. Such a procedure calculates the tree by adding nodes following a *postorder traversal* order of the LCA skeleton tree. For this reason we start from the LCA skeleton tree $T_S^{LCA}$, and we perform a postorder traversal of it, that require $O(|S|)$ time (in fact the leaves are the nodes of the set $S$). We now prove that, when visiting a node, it is possible to calculate the penalty score function in constant time.

The first step is to show how to calculate the number $|TP|$ for each node in $T_S^{LCA}$. In particular, this set corresponds to the set of descendant leaves of each node. Since we are doing a postorder traversal, it is guaranteed that when visiting a node, its children are already visited. In this way, starting from the leaves that have $|TP| = 1$, for each node $x \in N(T_S^{LCA})$:

$$|TP_x| = \sum_{y \in children(x)} |TP_y|$$

At this point for each node $x \in N(T_S^{LCA})$ we have $|nDesc(x)|$ (obtained in the pre-processing of $T$) and $|TP_x|$. As anticipated before, the two values $|FN_x|$ and $|FP_x|$ can be obtained as:

$$|FN_x| = |S| - |TP_x| \qquad \text{and} \qquad |FP_x| = |nDesc(x)| - |TP_x|$$

Both these operations require constant time, so the calculation of the penalty score function, which is:

$$PS(x) = q \cdot \frac{|FN_x|}{|TP_x|} + (1 - q) \cdot \frac{|FP_x|}{|TP_x|}$$

can be done in constant time. This means that the overall procedure for the penalty score calculation in $T_S^{LCA}$ requires $O(|S|)$ time. □

As a final remark, we recall the necessity of a pre-processing of the tree $T$ in the LCA skeleton tree construction. In the specific, a first pre-process of $T$ that requires $O(|T|)$ time is done, so that it is possible to obtain the LCA of any pair of nodes of $T$ in $O(1)$ time (see [63, 64] for details). Moreover, we perform and additional postorder traversal of $T$, that takes $O(|T|)$ time, in order to calculate the depth (necessary for the efficient

construction of the LCA skeleton tree) and to compute the set of descendant leaves of each node.

## 4.4 TANGO Improvement

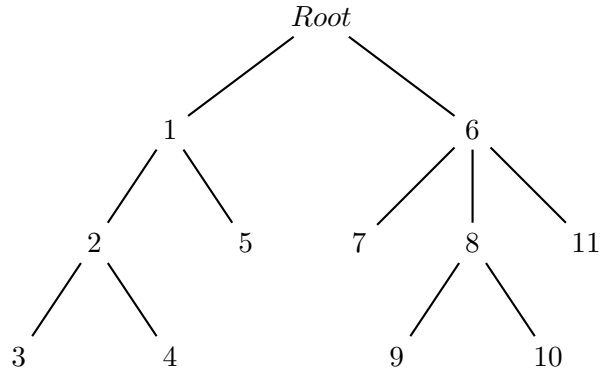
We start now describing the main improvements in the new version of the TANGO software. As anticipated in the previous sections, the objectives of this work are to realize a version of the software that performs the taxonomic assignment in a much faster way and also offers more flexibility in the choice of the taxonomy. To achieve the first goal, we have changed both the data structure and the algorithms used for the calculation of the penalty score. Moreover, the calculation of this last function is done by implementing the optimal algorithm described in the previous section. Indeed, to accomplish the second task, we have developed a procedure to contract the input taxonomies and to provide a support to them. In the rest of this section these steps are explained in detail.

### 4.4.1 New Data Structures

The first change in the TANGO program regards the data structures used to represent the taxonomic trees. More specifically, the chosen language of the program is *perl*[2], and in particular we adopted the *bio-perl* project [65] which is a collection of perl modules that are particularly suited for *bioinformatics* purposes. One of the main advantages in using this programming language is that it has the support of *regular expression*, for the processing of strings. Its regular expression support is the most versatile in existence and it is integrated into the language. In addition to this, the perl language can be used to realize stand-alone programs but it also offers the possibility of being integrated into other applications. Moreover, in bio-perl are present modules that provide an implementation of the tree data structure that can be used to represent $n$-ary trees. Although this data structure works well for general trees (also for taxonomic trees), it has some "limitations" for our purposes. In particular, one of the main issues is related to the size of the considered taxonomies: by dealing with entire taxonomic trees such as NCBI or RDP (that can have up to $\sim 10^6$ nodes), the memory occupation increases causing also a significant decrease of the (time) performance.

For this reason, for our objective, we have decided to use a simple representation of the tree based on the *parent, first-child, next-sibling* data structure, with no additional information (except for the node identifier and its taxonomic rank). This widely adopted

[2] `http://www.perl.org/`




| **Node** | *Root* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *parent* | *Root* | *Root* | 1 | 2 | 2 | 1 | *Root* | 6 | 6 | 8 | 8 | 6 |
| *first-child* | 1 | 2 | 3 | *null* | *null* | *null* | 7 | *null* | 9 | *null* | *null* | *null* |
| *next-sibling* | *null* | 6 | 5 | 4 | *null* | *null* | *null* | 8 | 11 | 10 | *null* | *null* |

FIGURE 4.6: Example of tree and the corresponding data structure obtain by taking the nodes as ordered from left to right (according to the picture). In particular, for each node we use three pointers: *parent*, $first\text{-}child$ and *next-sibling*. Notice the following special cases: the *Root* node (which has itself as parent), the leaves (which do not have a first child) and the last siblings (which do not have the next sibling).

tree representation is particularly suited for trees which do not have a fixed number of children. For each node, the information on its parent, its first child (if exists) and its next sibling (if exists) are stored. Taxonomic trees are not ordered trees, but the adoption of this data structure imposes an order among the children of any node. The resulting order is, however, immaterial with respect to our final goals. More specifically, given a rooted tree $T$, each node $x \in N(T)$ (where $x$ is the identifier of such node) contains the pointer to its parent node (denoted by $parent(x)$), the pointer to its first child node (denoted by $first\text{-}child(x)$), the pointer to its next sibling (denoted by $next\text{-}sibling(x)$) and a value associated to its taxonomic rank (denoted by $rank(x)$). The root $r$ of $T$ is a special case, in fact it has no parent; for this reason we decided to assign itself as parent, i.e. $parent(r) = r$. This property has allowed us to define the condition for the correct identification of the root in the taxonomic tree: the root $r$ is the (only) node $x \in N(T)$ such that $parent(x) = x$. The other special case involves the leaves of the tree, because they (by definition) do not have children. This means that, in particular, they do not have any "first child" and so the corresponding value is undefined, i.e. *null*. This means that if a node $x$ is a leaf, then $first\text{-}child(x) = null$. As in the previous case, this has allowed us to define the condition for the correct identification of such nodes: the leaves are the (only) nodes $x \in N(T)$ such that $first\text{-}child(x) = null$. A similar condition holds for the "last siblings" because they do not have any next sibling. This means that the corresponding value is *null*, i.e. $next\text{-}sibling(x) = null$ (see Figure 4.6 for an example).

The only other information used in the adopted data structure is the (taxonomic) rank

of the nodes. This is done with a bijective function $val$, that assigns to each rank in the set $Ranks$ of ranks, an integer value, i.e. $val : Ranks \to \mathbb{N}$. In particular, starting with the root which has value 1, higher taxonomic ranks will have lower values, while lower taxonomic ranks will have higher values. As stated before, let $Ranks$ be the set of ranks of a considered taxonomy, then it is possible to order such a set according to the taxonomic rank, from the highest to the lowest. Then for each rank $r_i \in Ranks$, $val(r_i) = i$, where $1 \leqslant i \leqslant |Ranks|$ is the position in the ordered set of ranks. Without loss of generality, it is possible to insert the root in the set $Ranks$, as the element with the highest taxonomic rank $r_1$, so that $val(r_1) = 1$. Recalling from Section 4.2.3, we have considered contracted trees in which nodes have only labels in $Ranks' \subseteq Ranks$. As a result, given a node $x$ in a contracted tree $T'$ (with ranks in $Ranks'$), it could happen that there exists a node $y$, child of $x$, which has a non-consecutive $val$ value, with respect to the one of $x$, i.e. $val(y) \neq val(x)+1$. As an example, consider the NCBI taxonomy which has 29 taxonomic ranks, meaning that the associated values of such ranks are from 1 to 29. The rank "Phylum" has $val = 6$, instead "Class" has $val = 9$. When contracting such taxonomy to the KPCOFGS valid ranks, the two previous ranks will not have consecutive values (although they could be consecutive in a lineage).

For each node $x$ of the taxonomic tree, we have stored the value of the function $val$, calculated on the taxonomic rank of the node $x$, in the structure $rank(x)$. This means that $rank(x) = val(r_i)$, where $r_i \in Ranks$ is the taxonomic rank of the node $x \in N(T)$. The property that is conserved by the contraction of a tree $T$ with rank set $Ranks$ to a tree $T'$ with rank set $Ranks' \subseteq Ranks$, is the following: each path $\langle x_1, \ldots, x_n \rangle$ from the root $r'(= x_1)$ to a leaf $x_n$ (lineage) has strictly increasing ranks, i.e. $rank(x_i) < rank(x_{i+1})$ for $1 \leqslant i < n$ (meaning that the taxonomic rank of node $x_i$ is higher than the one of the node $x_{i+1}$). In general, in each lineage by definition, the rank values increase while descending from the root to the leaves.

We have implemented these data structures by using perl *hash tables* (for details see perl documentation at `http://www.perl.org/docs.html`). More specifically, to represent the *parent*, *first-child*, *next-sibling* and *rank* structures we have used the hash tables, one for each of them. In this latter tables node identifiers are the keys of corresponding the entries. The use of hash tables guarantee us a fast access to each entry, so all the operations on the trees take advantage of this fact. Finally, one of the main advantages in using this data structure is that it is the simplest one which allow us to perform all the required operations on trees, in an efficient way.

### 4.4.2 Optimal Algorithm Implementation

The second contribution of this work is the realization of the new procedure for the penalty score calculation used in the taxonomic assignment. Let us just recall the algorithm showed in Section 4.3.1. We have proved that looking for the minimum value of the penalty score function can be equivalently done in both the trees (taxonomy and skeleton). Moreover, the calculation of such a function can be performed while constructing the skeleton tree, also maintaining an optimal time. We have implemented this procedure, that is based on the skeleton tree, and more specifically, instead of constructing it from the taxonomy, it is traversed (in postorder). In fact, as anticipated before, the building process is equivalent to a postorder traversal of the skeleton tree, so we have realized a procedure that visits only the nodes of this latter tree in the taxonomy, simulating the traversal.

In the algorithmic procedure, the only information needed for the computation of the penalty score function (done in constant time) are the number of descendant leaves in the taxonomy (that correspond to the $|nDesc|$ value) and in the skeleton tree (that correspond to the $|TP|$ value). We have obtained the first of them with a postorder traversal of the taxonomy, and we have stored this information in an additional hash table (with respect to the ones used to represent the tree, as described in Section 4.4.1). Such a table has the node identifiers as keys and the respective $|nDesc|$ as values. Indeed, we have incorporate the calculation of the second information (i.e. the $|TP|$ values) in the process for the read assignments, described in Algorithm 5.

As it is possible to notice, lines from 2 to 7 are the same as the first 6 lines (from 1 to 6) of Algorithm 3. Indeed, we want to detail the new adopted procedure for the non-trivial case (i.e. the one where the match set $|M_i| > 1$). Before this, the number of descendant leaves in the taxonomy is obtained with a postorder traversal (line 1) and these values are stored in the $nDescVal$ hash table. The first performed step is the calculation of the skeleton tree $T^{LCA}_{M_i}$ (line 9) and then a "postorder traversal" of it in which, for each node of such a tree, its number of descendant leaves is calculated (line 10), is performed. These values are stored in the $TPVal$ hash table, having node identifiers as keys and $|TP|$ as values. It must be noticed that this table contains only the nodes of the skeleton tree, i.e. the ones for which the penalty score function must be computed (let us recall that $N(T^{LCA}_{M_i}) \subseteq N(T)$). At this point, for each element $k$ in the $TPVal$ table, the penalty score value is calculated, starting from $nDescVal(k)$, $TPVal(k)$ and $q$ (lines 13 and 14). Then, if the new penalty score value $PS(k)$ is lower than the current minimum, it becomes the new minimum value, and a new list of "best" nodes is created by inserting the only element $k$. Otherwise, if the new penalty score value is the same as the current minimum, the element $k$ is simply appended to the latter list (lines 15–19). In this way,

**Algorithm 5**: Improved-TANGO

**Data**: a reference taxonomy $T$, a set $R$ of reads with associated the set of matches $M_i$ for each $R_i \in R$ and the parameter $k \in [0, 1]$.

```
1  nDescVal ← postorder traversal of T;
2  Assignments ← [ ];
3  foreach R_i ∈ R do
4      if |M_i| = 0 then
           // No assignment.
5          Assignments[i] ← ∅;
6      else if |M_i| = 1 then
           // The only leaf in M_i.
7          Assignments[i] ← (0, {x ∈ M_i});
8      else
9          T_{M_i}^{LCA} ← compute LCA skeleton tree;
10         TPVal ← postorder traversal of skeleton tree T_{M_i}^{LCA};
11         minPS ← ∞;
12         bestNodeSet ← ∅;
           // Penalty score calculation for the nodes of the skeleton tree.
13         foreach k in keys of TPVal do
14             PS(k) ← penalty score of k with values nDescVal(k), TPVal(k) and q;
15             if PS(k) < minPS then
16                 minPS ← PS(k);
17                 bestNodeSet ← {k};
18             else if PS(k) = minPS then
19                 append(bestNodeSet, k);
20     Assignments[i] ← (minPS, bestNodeSet);
21 return Assignments;
```

at the end of the procedure, the minimum value of the penalty score (for the read $R_i$) and also the *bestNodeSet* set of nodes having this value, are inserted into the list of the assignments, which is then returned.

By using the data structure introduced in Section 4.4.1, it is easy to see that such procedure is optimal. In fact, the pre-processing that traverses the taxonomy takes $O(|T|)$ time. Moreover, the loop at line 3 is executed $|R|$ times, and at each iteration there is the traversal of the skeleton tree that requires $O(|M_i|)$ time, and the same applies to the loop at line 13. All the other operations in the inner loop can be done in constant time, i.e. $O(1)$. This means that the overall time for executing the lines 4–20 is $O(|M_i|)$, where $M_i$ depends on the specific $R_i \in R$. The total required time for executing the algorithm 5 is $O(|T| + M)$, where

$$M = \sum_{i=1}^{|R|} |M_i|$$

This time can also be split to distinguish between the $O(|T|)$ time, needed to pre-process the taxonomy (that is independent from the set $R$ of reads), and the time required for the assignment of the reads, that is $O(M)$. This computational time is equivalent to the one showed in Section 4.3.1. Moreover, we consider contracted taxonomies as input of TANGO. Recalling Section 4.2.3, the contraction process performed with Algorithm 4 mimics a postorder traversal of the tree (that can be performed in $O(T)$), in which every visited node can be contracted. This latter operation requires time that is proportional to the number of children of the node being contracted.

Here we have showed how to obtain, for each read $R_i \in R$, all the nodes in $T_{M_i}^{LCA}$ with an optimal penalty score value. Moreover, by Lemma 4.4, these are all the nodes in the tree $T$, having the minimum value of the penalty score. As anticipated before, since we want to report one representative, we select the one with the lowest taxonomic rank (in order to be more specific). If there is more than one node, with both the same value and rank, we can arbitrary choose among them. This is done in a "post-processing" of the results of the assignments.

### 4.4.3 Input Mapping

As a final step, we have also developed a procedure to convert the input of TANGO to allow the assignment on a different taxonomy, with respect to the one used for the alignment of the reads. Let us recall from Section 4.1.2 that, in addition to the reference taxonomy $T$, the input to the TANGO software is a set of reads $R$ and, for each $R_i \in R$ its match set $M_i$. This latter is the set of nodes (usually leaves) of the reference taxonomy $T$ (represented as a tree), for which the read $R_i$ has a valid match. It is easy to see that this can be seen as the result of an alignment of $R_i$ in the reference taxonomy $T$ (obviously allowing multiple valid hits). One of the simplest way to represent such information is the following:

```
Read1    Node1  Node5  Node8  Node13
Read2    Node2  Node6
Read3    Node9
Read4    Node7  Node10 Node13
...
```

in which each row identify the alignment of a single read $R_i \in R$. In the specific, for each row there is the read identifier, followed by a space-separated list of nodes that are the ones for which the read has a valid match.

The conversion of this input into another taxonomy can be easily done by using the mapping among taxonomic trees, introduced at the end of Section 4.2.3. This latter process, which is done during the pre-processing step of the taxonomies, consists of creating a mapping file, in which there are the correspondences among nodes of two taxonomic trees. This mapping from taxonomy $T_1$ to taxonomy $T_2$, indicated as $T_1 \Rightarrow T_2$, assigns to each node $x_1 \in T_1$ that has a genomic sequence, the corresponding node $x_2 \in T_2$. For this reason, usually only the leaves are mapped into another taxonomy because these are the only nodes having a genome. In fact, the internal nodes, which correspond to higher taxonomic ranks, do not usually have a sequence of nucleotides. This also means that the read alignments are usually referred to leaves. In addition to this, when constructing a tree starting from the lineages, the only nodes having a sequence are the leaves. For this reason the list of nodes in the input file, representing the set of matches, is composed of leaves of the reference taxonomy.

The problem arises when we want to convert such a input into another taxonomy. The simplest way to do this, is to use the mapping among taxonomic trees (built in the pre-processing step) and to substitute each node in the list of matches, with the corresponding node in the other taxonomy. As anticipated before, the mapping does not guarantee that a leaf node is mapped to another leaf (in the other taxonomic tree). This can cause some problem in the calculation of the penalty score because the match set is not well identified (and consequently the set of true positives). To overcome this limitation, we have decided to ignore such mappings and to not report them in the conversion. This is equivalent to the situation for which the mapping of a node is not available, or simply not present, in the other taxonomy.

As a final remark, we want to underline the choice, in the *tree contraction* procedure (see Section 4.2.3), to keep the leaves also if they do not have a valid rank. In fact, as explained before, the matches of reads usually refer to such nodes, so we want to keep them also in the contracted tree.

## 4.5 Conclusions and Future Work

In this work we have presented an improvement of the taxonomic assignment procedure performed by the TANGO software. This is one of the main tasks in metagenomics and it is the base of analysis aimed, for example, to identify the composition specific microbial communities. The advent of NGS techniques has made possible such studies, that were non achievable before. In fact, such methods are widely used for sequencing samples, allowing us to obtain a set of reads coming from the organisms present in the environment. One of the most used approaches to distinguish and identify the organisms

present in a sample, starting from a set of short reads coming from it, is to assign them into a reference taxonomy. To do this, the sequences are firstly aligned to such taxonomy and then a representative node is chosen for each read. As underlined, this method has a big limitation: the assignment depends on the chosen taxonomy. To overcome this fact, the first part of the presented work was aimed to analyze the available taxonomies. We have discovered that when considering the tree representation, they have some problems, especially in the structure, making the comparison difficult. For this reason we have decided to construct the tree starting from the lineages, which helped us to solve some inconsistencies among trees. More specifically, we have constructed different taxonomies in order to make the read assignment process more flexible. Moreover, we have developed a method to contract the taxonomic trees with respect to a subset of taxonomic ranks, in order to reduce the analyzed trees to the same set of ranks. This procedure can help the comparison of the taxonomic assignments of the reads. We have also provided the support to different taxonomies, allowing the user to choose among them for the read assignments.

The second part of this work is focused on the realization of a new version of the TANGO software, with the precise objective of improving its performances. The core of this work is the realization of a new algorithms for the calculation of the minimum penalty score value, used to select the representative node, in the taxonomic assignment. In particular, we have showed how it is possible to use the skeleton tree for the calculation process, and we have proved its optimality. This means that, instead of computing the penalty score in each node in the subtree induced by the match set, this function can be calculated while traversing the skeleton tree for its construction. We have showed that, looking for the minimum value of this function in the skeleton tree, instead of in the taxonomic tree, is equivalent. This guarantees that the minimum value is found. Moreover, we showed that the computations in each node of the skeleton tree can be done in constant time, so that the overall time required, remains the same to the one required for the construction of the skeleton tree.

We have implemented these features in the new version of the TANGO program, with the additional scope of making the assignment process faster than the actual one. To accomplish this task, we have used new data structures to represent the taxonomic trees. We have decided to use the hash table of the perl language to speed up the interrogations and the basic operations on the trees.

Although TANGO provides an optimal assignment with respect to the penalty score value, there could exist more nodes in the tree, with such a (optimal) value. This means that they can be equally chosen as representative of the read, and actually, this choice is done by selecting the node with the lower taxonomic rank. If there are more nodes

with the same lower taxonomic rank too, we chose one of these nodes randomly (in order to have only one representative node for read). For this reason, we need to develop a method that allows to weigh the reads in the tree, to identify the most relevant ones. This can be done for example, by assigning probabilistic values, based on the taxonomic assignment of all the reads. This can be seen as a global view of the assignment that guide the choice on relevant nodes.

Another problem that is closely related to the previous one, is the choice of the parameter $q$ used for the calculation of the penalty score function. In fact, such a value influences the assignment by preferring lower or higher taxonomic ranks (see Section 4.1.2). We want to realize a procedure to automatize this choice, maybe based of external information. So doing, the entire process of taxonomic assignment in TANGO could become transparent to the user.

In addition to the two previously introduced future developments, that regard the assignment process, there are other possible improvement of the TANGO software. One of these, is to provide the support for the NGS aligning programs. More specifically, since the input is the set of valid matches of a read (which is the result of an alignment of such a read in the taxonomy), we want to provide the support for the most widely used programs. Some examples are BLAST[38], which is still used also for NGS data or Bowtie[28] that is one of the most used software, due to its speed.

On the other hand, a further future work is about the produced output. Up to now, we only give, for every read in the input set, its optimal taxonomic assignment, i.e. a node in the taxonomic tree that best represents such a read. In some cases, this kind of output could be too specific, because sometimes it is necessary to have a complete overview on all the assignments. To do this, we want to produce visual output that gives a "compact view" of these assignments in order to have, for example, for each taxonomic rank, the partition of the reads. More specifically, how the reads are subdivided among the nodes of the taxonomic tree. Recalling the previously introduced taxonomies, we have Bacteria and Archaea as *kingdoms*. A possible interrogation of the assignment process could be to know how the reads are split between the two previous kingdoms, or alternatively to know how many reads are assigned directly to these specific taxonomic ranks. All these information can be summarized in tabular format or graphics, such as pie-charts or histograms.

An additional future work, regards the process of mapping between taxonomic trees. As underlined, there could be some problems during this procedure, especially when the mapping (that usually starts from a leaf node) ends in an internal node of the tree. We want to realize a procedure to manage this case. A possible direction is the propagation of the mapping on all the leaf descendants of the internal nodes to which the mapping is

directed. Alternatively, the mapping could be partitioned among the descendant nodes by assigning a weight to them.

Finally, we have the objective of extending the support to other available taxonomies, offering to the user the possibility to choose among them porting the results, or better to perform the assignment on all of them and to compare the results, highlighting differences an similarities. All these future works can result in a very powerful tool for the taxonomic assignment of read in metagenomics, and considering the so fast development of NGS techniques, this task is going to be a central one in bioinformatics.

# Appendix A

# Additional Data

TABLE A.1: Details of the first experiment of Section 3.5 on 112 genes. For each gene, the data concerning the predicted splicing graph (denoted as $G_R$) and the correct isoform graph (denoted as $G_S$) are reported. For $G_S$, its total number of nodes and arcs are shown. For the nodes of $G_R$, the total number, the number of correct ones, Sn and PPV values are reported. The same values are shown also for the arcs of $G_R$. Finally, the sum of total (and correct) nodes and arcs are reported. The average and median values of Sn and PPV are calculated. Two average values are present, denoted as **Global** and **Single**. The first one is obtained starting from the **Total** values, while the second one is the average of all the values present in the table.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| ARHGAP4 | 34 | 47 | 32 | 31 | 0.91 | 0.97 | 45 | 41 | 0.87 | 0.91 |
| ATP11A | 20 | 23 | 20 | 17 | 0.85 | 0.85 | 21 | 14 | 0.61 | 0.67 |
| ATP6AP1 | 9 | 12 | 9 | 8 | 0.89 | 0.89 | 11 | 10 | 0.83 | 0.91 |
| AVPR2 | 7 | 8 | 7 | 7 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| BPIL2 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| BRCC3 | 11 | 15 | 9 | 9 | 0.82 | 1.00 | 11 | 10 | 0.67 | 0.91 |
| C20orf173 | 5 | 6 | 5 | 5 | 1.00 | 1.00 | 6 | 6 | 1.00 | 1.00 |
| C22orf24 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| C22orf28 | 11 | 13 | 11 | 11 | 1.00 | 1.00 | 12 | 12 | 0.92 | 1.00 |
| C22orf30 | 12 | 15 | 12 | 10 | 0.83 | 0.83 | 15 | 11 | 0.73 | 0.73 |
| C6orf150 | 7 | 7 | 7 | 7 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| C9orf106 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| CEP250 | 22 | 29 | 18 | 15 | 0.68 | 0.83 | 25 | 13 | 0.45 | 0.52 |
| CGN | 11 | 12 | 11 | 10 | 0.91 | 0.91 | 11 | 10 | 0.83 | 0.91 |
| CPNE1 | 33 | 54 | 23 | 19 | 0.58 | 0.83 | 29 | 21 | 0.39 | 0.72 |
| CRAT | 13 | 18 | 12 | 12 | 0.92 | 1.00 | 16 | 14 | 0.78 | 0.88 |
| CTAG1A | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| CTAG1B | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |

(continue)

TABLE A.1: Details of the first experiment of Section 3.5.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| CTAG2 | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| DDX43 | 5 | 4 | 5 | 5 | 1.00 | 1.00 | 4 | 4 | 1.00 | 1.00 |
| DEPDC5 | 34 | 42 | 37 | 31 | 0.91 | 0.84 | 44 | 36 | 0.86 | 0.82 |
| DKC1 | 19 | 25 | 17 | 16 | 0.84 | 0.94 | 19 | 16 | 0.64 | 0.84 |
| DNASE1L1 | 10 | 14 | 10 | 10 | 1.00 | 1.00 | 14 | 14 | 1.00 | 1.00 |
| DOLPP1 | 9 | 11 | 9 | 9 | 1.00 | 1.00 | 11 | 11 | 1.00 | 1.00 |
| DRG1 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| EEF1A1 | 25 | 36 | 23 | 22 | 0.88 | 0.96 | 32 | 28 | 0.78 | 0.88 |
| EIF4ENIF1 | 19 | 23 | 17 | 17 | 0.90 | 1.00 | 20 | 20 | 0.87 | 1.00 |
| EMD | 14 | 18 | 12 | 11 | 0.79 | 0.92 | 12 | 8 | 0.44 | 0.67 |
| ENPP1 | 5 | 5 | 8 | 4 | 0.80 | 0.50 | 9 | 4 | 0.80 | 0.44 |
| ERGIC3 | 33 | 46 | 30 | 26 | 0.79 | 0.87 | 41 | 26 | 0.56 | 0.63 |
| F10 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| F7 | 12 | 16 | 10 | 10 | 0.83 | 1.00 | 12 | 8 | 0.50 | 0.67 |
| F8 | 8 | 8 | 8 | 8 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| F8A1 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| FAM3A | 21 | 31 | 17 | 15 | 0.71 | 0.88 | 18 | 14 | 0.45 | 0.78 |
| FAM50A | 12 | 13 | 12 | 12 | 1.00 | 1.00 | 13 | 13 | 1.00 | 1.00 |
| FAM73B | 25 | 35 | 24 | 20 | 0.80 | 0.83 | 29 | 20 | 0.57 | 0.69 |
| FAM83C | 5 | 6 | 6 | 5 | 1.00 | 0.83 | 6 | 6 | 1.00 | 1.00 |
| FBXO7 | 14 | 19 | 13 | 12 | 0.86 | 0.92 | 17 | 16 | 0.84 | 0.94 |
| FER1L4 | 42 | 59 | 44 | 41 | 0.98 | 0.93 | 61 | 57 | 0.97 | 0.93 |
| FLNA | 40 | 57 | 38 | 34 | 0.85 | 0.90 | 53 | 42 | 0.74 | 0.79 |
| FOXP4 | 11 | 14 | 10 | 10 | 0.91 | 1.00 | 12 | 12 | 0.86 | 1.00 |
| FRS3 | 8 | 8 | 8 | 8 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| FUNDC2 | 10 | 11 | 10 | 10 | 1.00 | 1.00 | 12 | 11 | 1.00 | 0.92 |
| G6PD | 19 | 24 | 16 | 13 | 0.68 | 0.81 | 18 | 12 | 0.50 | 0.67 |
| GAB3 | 10 | 12 | 9 | 9 | 0.90 | 1.00 | 10 | 10 | 0.83 | 1.00 |
| GDF5 | 3 | 2 | 3 | 3 | 1.00 | 1.00 | 2 | 2 | 1.00 | 1.00 |
| H2AFB1 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| HCFC1 | 7 | 9 | 6 | 5 | 0.71 | 0.83 | 7 | 6 | 0.67 | 0.86 |
| IER5L | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| IKBKG | 19 | 25 | 19 | 17 | 0.90 | 0.90 | 31 | 15 | 0.60 | 0.48 |
| IRAK1 | 30 | 46 | 23 | 22 | 0.73 | 0.96 | 29 | 23 | 0.50 | 0.79 |
| KATNAL1 | 7 | 8 | 10 | 6 | 0.86 | 0.60 | 9 | 6 | 0.75 | 0.67 |
| KCNQ5 | 10 | 11 | 10 | 7 | 0.70 | 0.70 | 9 | 3 | 0.27 | 0.33 |
| L1CAM | 25 | 31 | 22 | 21 | 0.84 | 0.95 | 27 | 22 | 0.71 | 0.81 |
| LACE1 | 6 | 7 | 5 | 4 | 0.67 | 0.80 | 4 | 3 | 0.43 | 0.75 |

(continue)

TABLE A.1: Details of the first experiment of Section 3.5.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| LAGE3 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| MCF2L | 48 | 58 | 46 | 44 | 0.92 | 0.96 | 56 | 49 | 0.84 | 0.88 |
| MDFI | 11 | 14 | 11 | 11 | 1.00 | 1.00 | 14 | 14 | 1.00 | 1.00 |
| MECP2 | 16 | 21 | 15 | 13 | 0.81 | 0.87 | 19 | 8 | 0.38 | 0.42 |
| MMP24 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| MOXD1 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| MPP1 | 23 | 33 | 22 | 22 | 0.96 | 1.00 | 32 | 32 | 0.97 | 1.00 |
| MTCP1 | 7 | 10 | 6 | 5 | 0.71 | 0.83 | 8 | 3 | 0.30 | 0.38 |
| MTO1 | 15 | 21 | 14 | 13 | 0.87 | 0.93 | 19 | 18 | 0.86 | 0.95 |
| NCR2 | 5 | 6 | 5 | 5 | 1.00 | 1.00 | 6 | 6 | 1.00 | 1.00 |
| NFS1 | 18 | 24 | 17 | 16 | 0.89 | 0.94 | 22 | 21 | 0.88 | 0.95 |
| NR2E1 | 6 | 5 | 6 | 6 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| NUP188 | 21 | 23 | 19 | 17 | 0.81 | 0.90 | 20 | 16 | 0.70 | 0.80 |
| OPN1LW | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| OPN1MW | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| OSTM1 | 12 | 15 | 11 | 9 | 0.75 | 0.82 | 12 | 8 | 0.53 | 0.67 |
| PCDH15 | 19 | 27 | 18 | 18 | 0.95 | 1.00 | 23 | 23 | 0.85 | 1.00 |
| PGC | 7 | 8 | 7 | 7 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| PHYHD1 | 13 | 19 | 12 | 11 | 0.85 | 0.92 | 18 | 14 | 0.74 | 0.78 |
| PIP5K1A | 19 | 23 | 17 | 15 | 0.79 | 0.88 | 19 | 13 | 0.56 | 0.68 |
| PISD | 23 | 32 | 18 | 15 | 0.65 | 0.83 | 19 | 15 | 0.47 | 0.79 |
| PLXNA3 | 16 | 17 | 14 | 13 | 0.81 | 0.93 | 15 | 12 | 0.71 | 0.80 |
| POGZ | 22 | 28 | 22 | 21 | 0.95 | 0.95 | 27 | 27 | 0.96 | 1.00 |
| PPP2R4 | 25 | 32 | 25 | 25 | 1.00 | 1.00 | 32 | 32 | 1.00 | 1.00 |
| PSMB4 | 12 | 17 | 11 | 11 | 0.92 | 1.00 | 15 | 14 | 0.82 | 0.93 |
| PSMD4 | 23 | 31 | 21 | 19 | 0.83 | 0.91 | 30 | 20 | 0.65 | 0.67 |
| RBM12 | 5 | 7 | 4 | 4 | 0.80 | 1.00 | 4 | 4 | 0.57 | 1.00 |
| RBM39 | 41 | 58 | 35 | 33 | 0.81 | 0.94 | 41 | 36 | 0.62 | 0.88 |
| RENBP | 16 | 23 | 15 | 15 | 0.94 | 1.00 | 20 | 20 | 0.87 | 1.00 |
| RFPL2 | 3 | 2 | 3 | 3 | 1.00 | 1.00 | 2 | 2 | 1.00 | 1.00 |
| RFPL3 | 3 | 2 | 1 | 0 | 0.00 | 0.00 | 0 | 0 | 0.00 | |
| RFPL3S | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| RFX5 | 25 | 36 | 18 | 14 | 0.56 | 0.78 | 21 | 12 | 0.33 | 0.57 |
| RP11-374F3.4 | 15 | 17 | 15 | 15 | 1.00 | 1.00 | 17 | 17 | 1.00 | 1.00 |
| RPL10 | 16 | 25 | 13 | 12 | 0.75 | 0.92 | 13 | 13 | 0.52 | 1.00 |
| SEC63 | 2 | 0 | 2 | 2 | 1.00 | 1.00 | 0 | 0 | | |
| SELENBP1 | 22 | 33 | 21 | 21 | 0.95 | 1.00 | 26 | 26 | 0.79 | 1.00 |
| SFI1 | 43 | 54 | 42 | 42 | 0.98 | 1.00 | 54 | 50 | 0.93 | 0.93 |

(continue)

TABLE A.1: Details of the first experiment of Section 3.5.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| SH3GLB2 | 18 | 26 | 16 | 14 | 0.78 | 0.88 | 21 | 16 | 0.61 | 0.76 |
| SLC10A3 | 7 | 9 | 7 | 7 | 1.00 | 1.00 | 9 | 9 | 1.00 | 1.00 |
| SLC5A1 | 5 | 4 | 3 | 2 | 0.40 | 0.67 | 2 | 1 | 0.25 | 0.50 |
| SLC5A4 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| SNX27 | 6 | 6 | 5 | 5 | 0.83 | 1.00 | 5 | 5 | 0.83 | 1.00 |
| SNX3 | 5 | 7 | 5 | 5 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| SPAG4 | 14 | 19 | 13 | 12 | 0.86 | 0.92 | 17 | 13 | 0.68 | 0.77 |
| STAG2 | 26 | 37 | 31 | 22 | 0.85 | 0.71 | 58 | 21 | 0.57 | 0.36 |
| SYN3 | 19 | 19 | 19 | 19 | 1.00 | 1.00 | 19 | 19 | 1.00 | 1.00 |
| TAZ | 30 | 48 | 25 | 23 | 0.77 | 0.92 | 28 | 24 | 0.50 | 0.86 |
| TFEB | 24 | 29 | 20 | 17 | 0.71 | 0.85 | 22 | 8 | 0.28 | 0.36 |
| TIMP3 | 5 | 6 | 3 | 3 | 0.60 | 1.00 | 2 | 1 | 0.17 | 0.50 |
| TKTL1 | 10 | 12 | 9 | 8 | 0.80 | 0.89 | 10 | 9 | 0.75 | 0.90 |
| TUFT1 | 12 | 14 | 12 | 12 | 1.00 | 1.00 | 14 | 14 | 1.00 | 1.00 |
| USP49 | 6 | 7 | 6 | 6 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| VPS72 | 6 | 7 | 4 | 3 | 0.50 | 0.75 | 3 | 2 | 0.29 | 0.67 |
| YWHAH | 6 | 7 | 6 | 6 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| ZNF687 | 10 | 13 | 10 | 10 | 1.00 | 1.00 | 13 | 13 | 1.00 | 1.00 |
| **Total** | 1521 | 1966 | 1415 | 1303 | | | 1719 | 1415 | | |
| **Global Avg.** | | | | | 0.86 | 0.92 | | | 0.72 | 0.82 |
| **Single Avg.** | | | | | 0.88 | 0.93 | | | 0.77 | 0.86 |
| **Median** | | | | | 0.91 | 1.00 | | | 0.83 | 0.93 |

TABLE A.2: Details of the second experiment of Section 3.5 on 112 genes. For each gene, the data concerning the predicted splicing graph (denoted as $G_R$) and the correct isoform graph (denoted as $G_S$) are reported. For $G_S$, its total number of nodes and arcs are shown. For the nodes of $G_R$, the total number, the number of correct ones, Sn and PPV values are reported. The same values are shown also for the arcs of $G_R$. Finally, the sum of total (and correct) nodes and arcs are reported. The average and median values of Sn and PPV are calculated. Two average values are present, denoted as **Global** and **Single**. The first one is obtained starting from the **Total** values, while the second one is the average of all the values present in the table.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| ARHGAP4 | 34 | 47 | 33 | 30 | 0.88 | 0.91 | 45 | 40 | 0.85 | 0.89 |
| ATP11A | 20 | 23 | 21 | 19 | 0.95 | 0.91 | 23 | 20 | 0.87 | 0.87 |
| ATP6AP1 | 9 | 12 | 9 | 8 | 0.89 | 0.89 | 11 | 10 | 0.83 | 0.91 |
| AVPR2 | 7 | 8 | 7 | 7 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| BPIL2 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| BRCC3 | 11 | 15 | 9 | 8 | 0.73 | 0.89 | 11 | 9 | 0.60 | 0.82 |
| C20orf173 | 5 | 6 | 5 | 5 | 1.00 | 1.00 | 6 | 6 | 1.00 | 1.00 |
| C22orf24 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| C22orf28 | 11 | 13 | 11 | 11 | 1.00 | 1.00 | 12 | 12 | 0.92 | 1.00 |
| C22orf30 | 12 | 15 | 12 | 10 | 0.83 | 0.83 | 15 | 11 | 0.73 | 0.73 |
| C6orf150 | 7 | 7 | 7 | 7 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| C9orf106 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| CEP250 | 22 | 29 | 19 | 16 | 0.73 | 0.84 | 28 | 15 | 0.52 | 0.54 |
| CGN | 11 | 12 | 13 | 9 | 0.82 | 0.69 | 14 | 8 | 0.67 | 0.57 |
| CPNE1 | 33 | 54 | 25 | 24 | 0.73 | 0.96 | 35 | 31 | 0.57 | 0.89 |
| CRAT | 13 | 18 | 12 | 12 | 0.92 | 1.00 | 16 | 14 | 0.78 | 0.88 |
| CTAG1A | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| CTAG1B | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| CTAG2 | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| DDX43 | 5 | 4 | 5 | 5 | 1.00 | 1.00 | 4 | 4 | 1.00 | 1.00 |
| DEPDC5 | 34 | 42 | 36 | 31 | 0.91 | 0.86 | 41 | 36 | 0.86 | 0.88 |
| DKC1 | 19 | 25 | 17 | 16 | 0.84 | 0.94 | 19 | 16 | 0.64 | 0.84 |
| DNASE1L1 | 10 | 14 | 10 | 10 | 1.00 | 1.00 | 14 | 14 | 1.00 | 1.00 |
| DOLPP1 | 9 | 11 | 9 | 9 | 1.00 | 1.00 | 11 | 11 | 1.00 | 1.00 |
| DRG1 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| EEF1A1 | 25 | 36 | 23 | 22 | 0.88 | 0.96 | 32 | 28 | 0.78 | 0.88 |
| EIF4ENIF1 | 19 | 23 | 18 | 16 | 0.84 | 0.89 | 21 | 19 | 0.83 | 0.91 |
| EMD | 14 | 18 | 13 | 12 | 0.86 | 0.92 | 14 | 11 | 0.61 | 0.79 |
| ENPP1 | 5 | 5 | 8 | 4 | 0.80 | 0.50 | 9 | 4 | 0.80 | 0.44 |
| ERGIC3 | 33 | 46 | 33 | 25 | 0.76 | 0.76 | 45 | 26 | 0.56 | 0.58 |
| F10 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |

(continue)

TABLE A.2: Details of the second experiment of Section 3.5.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| F7 | 12 | 16 | 10 | 10 | 0.83 | 1.00 | 12 | 8 | 0.50 | 0.67 |
| F8 | 8 | 8 | 8 | 8 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| F8A1 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| FAM3A | 21 | 31 | 17 | 15 | 0.71 | 0.88 | 18 | 14 | 0.45 | 0.78 |
| FAM50A | 12 | 13 | 12 | 11 | 0.92 | 0.92 | 12 | 11 | 0.85 | 0.92 |
| FAM73B | 25 | 35 | 24 | 19 | 0.76 | 0.79 | 29 | 19 | 0.54 | 0.66 |
| FAM83C | 5 | 6 | 7 | 4 | 0.80 | 0.57 | 6 | 4 | 0.67 | 0.67 |
| FBXO7 | 14 | 19 | 16 | 13 | 0.93 | 0.81 | 21 | 17 | 0.90 | 0.81 |
| FER1L4 | 42 | 59 | 44 | 41 | 0.98 | 0.93 | 61 | 57 | 0.97 | 0.93 |
| FLNA | 40 | 57 | 40 | 36 | 0.90 | 0.90 | 59 | 47 | 0.82 | 0.80 |
| FOXP4 | 11 | 14 | 10 | 9 | 0.82 | 0.90 | 12 | 10 | 0.71 | 0.83 |
| FRS3 | 8 | 8 | 8 | 8 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| FUNDC2 | 10 | 11 | 10 | 10 | 1.00 | 1.00 | 12 | 11 | 1.00 | 0.92 |
| G6PD | 19 | 24 | 16 | 13 | 0.68 | 0.81 | 18 | 12 | 0.50 | 0.67 |
| GAB3 | 10 | 12 | 9 | 9 | 0.90 | 1.00 | 10 | 10 | 0.83 | 1.00 |
| GDF5 | 3 | 2 | 3 | 3 | 1.00 | 1.00 | 2 | 2 | 1.00 | 1.00 |
| H2AFB1 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| HCFC1 | 7 | 9 | 7 | 6 | 0.86 | 0.86 | 9 | 7 | 0.78 | 0.78 |
| IER5L | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| IKBKG | 19 | 25 | 22 | 17 | 0.90 | 0.77 | 38 | 21 | 0.84 | 0.55 |
| IRAK1 | 30 | 46 | 23 | 22 | 0.73 | 0.96 | 30 | 23 | 0.50 | 0.77 |
| KATNAL1 | 7 | 8 | 17 | 6 | 0.86 | 0.35 | 38 | 6 | 0.75 | 0.16 |
| KCNQ5 | 10 | 11 | 12 | 7 | 0.70 | 0.58 | 12 | 3 | 0.27 | 0.25 |
| L1CAM | 25 | 31 | 22 | 21 | 0.84 | 0.95 | 27 | 22 | 0.71 | 0.81 |
| LACE1 | 6 | 7 | 5 | 4 | 0.67 | 0.80 | 4 | 3 | 0.43 | 0.75 |
| LAGE3 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| MCF2L | 48 | 58 | 46 | 46 | 0.96 | 1.00 | 56 | 52 | 0.90 | 0.93 |
| MDFI | 11 | 14 | 10 | 10 | 0.91 | 1.00 | 12 | 12 | 0.86 | 1.00 |
| MECP2 | 16 | 21 | 15 | 13 | 0.81 | 0.87 | 19 | 8 | 0.38 | 0.42 |
| MMP24 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| MOXD1 | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| MPP1 | 23 | 33 | 22 | 22 | 0.96 | 1.00 | 32 | 32 | 0.97 | 1.00 |
| MTCP1 | 7 | 10 | 7 | 7 | 1.00 | 1.00 | 10 | 10 | 1.00 | 1.00 |
| MTO1 | 15 | 21 | 15 | 15 | 1.00 | 1.00 | 21 | 21 | 1.00 | 1.00 |
| NCR2 | 5 | 6 | 5 | 5 | 1.00 | 1.00 | 6 | 6 | 1.00 | 1.00 |
| NFS1 | 18 | 24 | 17 | 17 | 0.94 | 1.00 | 22 | 22 | 0.92 | 1.00 |
| NR2E1 | 6 | 5 | 9 | 5 | 0.83 | 0.56 | 9 | 3 | 0.60 | 0.33 |
| NUP188 | 21 | 23 | 19 | 17 | 0.81 | 0.90 | 20 | 16 | 0.70 | 0.80 |

(continue)

TABLE A.2: Details of the second experiment of Section 3.5.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| OPN1LW | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| OPN1MW | 3 | 3 | 3 | 3 | 1.00 | 1.00 | 3 | 3 | 1.00 | 1.00 |
| OSTM1 | 12 | 15 | 10 | 9 | 0.75 | 0.90 | 12 | 8 | 0.53 | 0.67 |
| PCDH15 | 19 | 27 | 18 | 18 | 0.95 | 1.00 | 23 | 23 | 0.85 | 1.00 |
| PGC | 7 | 8 | 7 | 7 | 1.00 | 1.00 | 8 | 8 | 1.00 | 1.00 |
| PHYHD1 | 13 | 19 | 12 | 11 | 0.85 | 0.92 | 18 | 14 | 0.74 | 0.78 |
| PIP5K1A | 19 | 23 | 17 | 15 | 0.79 | 0.88 | 19 | 13 | 0.56 | 0.68 |
| PISD | 23 | 32 | 19 | 17 | 0.74 | 0.90 | 22 | 18 | 0.56 | 0.82 |
| PLXNA3 | 16 | 17 | 14 | 13 | 0.81 | 0.93 | 15 | 12 | 0.71 | 0.80 |
| POGZ | 22 | 28 | 22 | 21 | 0.95 | 0.95 | 27 | 27 | 0.96 | 1.00 |
| PPP2R4 | 25 | 32 | 25 | 25 | 1.00 | 1.00 | 32 | 32 | 1.00 | 1.00 |
| PSMB4 | 12 | 17 | 11 | 11 | 0.92 | 1.00 | 15 | 14 | 0.82 | 0.93 |
| PSMD4 | 23 | 31 | 23 | 22 | 0.96 | 0.96 | 36 | 28 | 0.90 | 0.78 |
| RBM12 | 5 | 7 | 4 | 4 | 0.80 | 1.00 | 4 | 4 | 0.57 | 1.00 |
| RBM39 | 41 | 58 | 35 | 33 | 0.81 | 0.94 | 41 | 36 | 0.62 | 0.88 |
| RENBP | 16 | 23 | 15 | 14 | 0.88 | 0.93 | 20 | 18 | 0.78 | 0.90 |
| RFPL2 | 3 | 2 | 3 | 3 | 1.00 | 1.00 | 2 | 2 | 1.00 | 1.00 |
| RFPL3 | 3 | 2 | 1 | 0 | 0.00 | 0.00 | 0 | 0 | 0.00 | |
| RFPL3S | 5 | 5 | 5 | 5 | 1.00 | 1.00 | 5 | 5 | 1.00 | 1.00 |
| RFX5 | 25 | 36 | 19 | 16 | 0.64 | 0.84 | 22 | 15 | 0.42 | 0.68 |
| RP11-374F3.4 | 15 | 17 | 15 | 15 | 1.00 | 1.00 | 17 | 17 | 1.00 | 1.00 |
| RPL10 | 16 | 25 | 13 | 12 | 0.75 | 0.92 | 13 | 13 | 0.52 | 1.00 |
| SEC63 | 2 | 0 | 2 | 2 | 1.00 | 1.00 | 0 | 0 | | |
| SELENBP1 | 22 | 33 | 21 | 20 | 0.91 | 0.95 | 26 | 24 | 0.73 | 0.92 |
| SFI1 | 43 | 54 | 42 | 42 | 0.98 | 1.00 | 54 | 50 | 0.93 | 0.93 |
| SH3GLB2 | 18 | 26 | 17 | 14 | 0.78 | 0.82 | 24 | 17 | 0.65 | 0.71 |
| SLC10A3 | 7 | 9 | 7 | 7 | 1.00 | 1.00 | 9 | 9 | 1.00 | 1.00 |
| SLC5A1 | 5 | 4 | 3 | 2 | 0.40 | 0.67 | 2 | 1 | 0.25 | 0.50 |
| SLC5A4 | 1 | 0 | 1 | 1 | 1.00 | 1.00 | 0 | 0 | | |
| SNX27 | 6 | 6 | 5 | 5 | 0.83 | 1.00 | 5 | 5 | 0.83 | 1.00 |
| SNX3 | 5 | 7 | 5 | 5 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| SPAG4 | 14 | 19 | 14 | 13 | 0.93 | 0.93 | 18 | 15 | 0.79 | 0.83 |
| STAG2 | 26 | 37 | 32 | 23 | 0.89 | 0.72 | 64 | 24 | 0.65 | 0.38 |
| SYN3 | 19 | 19 | 19 | 19 | 1.00 | 1.00 | 19 | 19 | 1.00 | 1.00 |
| TAZ | 30 | 48 | 26 | 22 | 0.73 | 0.85 | 29 | 23 | 0.48 | 0.79 |
| TFEB | 24 | 29 | 21 | 19 | 0.79 | 0.91 | 24 | 20 | 0.69 | 0.83 |
| TIMP3 | 5 | 6 | 3 | 3 | 0.60 | 1.00 | 2 | 1 | 0.17 | 0.50 |
| TKTL1 | 10 | 12 | 10 | 10 | 1.00 | 1.00 | 12 | 12 | 1.00 | 1.00 |

(continue)

TABLE A.2: Details of the second experiment of Section 3.5.

| | $G_S$ | | $G_R$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nodes | Arcs | Nodes | | | | Arcs | | | |
| Gene | Tot. | Tot. | Tot. | Correct | Sn | PPV | Tot. | Correct | Sn | PPV |
| TUFT1 | 12 | 14 | 12 | 12 | 1.00 | 1.00 | 14 | 14 | 1.00 | 1.00 |
| USP49 | 6 | 7 | 6 | 6 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| VPS72 | 6 | 7 | 4 | 3 | 0.50 | 0.75 | 3 | 2 | 0.29 | 0.67 |
| YWHAH | 6 | 7 | 6 | 6 | 1.00 | 1.00 | 7 | 7 | 1.00 | 1.00 |
| ZNF687 | 10 | 13 | 10 | 10 | 1.00 | 1.00 | 13 | 13 | 1.00 | 1.00 |
| **Total** | 1521 | 1966 | 1458 | 1322 | | | 1819 | 1477 | | |
| **Global Avg.** | | | | | 0.87 | 0.91 | | | 0.75 | 0.81 |
| **Single Avg.** | | | | | 0.89 | 0.92 | | | 0.78 | 0.85 |
| **Median** | | | | | 0.93 | 1.00 | | | 0.84 | 0.91 |

# Bibliography


[1] M. Gabut, P. Samavarchi-Tehrani, X. Wang, V. Slobodeniuc, D. O'Hanlon, H.-K. Sung, M. Alvarez, S. Talukder, Q. Pan, E. Mazzoni, S. Nedelec, H. Wichterle, K. Woltjen, T. Hughes, P. W. Zandstra, A. Nagy, J. Wrana, and B. Blencowe, *Cell*. Cell Press, Sep 2011, vol. 147, no. 1, ch. An Alternative Splicing Switch Regulates Embryonic Stem Cell Pluripotency and Reprogramming, pp. 132–146. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0092867411009494

[2] S. Beretta, P. Bonizzoni, G. Della Vedova, and R. Rizzi, "Reconstructing Isoform Graphs from RNA-Seq data," in *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Oct. 2012, in press.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[4] H. Pearson, "Genetics: What is a gene?" *Nature*, vol. 441, no. 7092, pp. 398–401, May 2006. [Online]. Available: http://dx.doi.org/10.1038/441398a

[5] M. B. Gerstein, C. Bruce, J. S. Rozowsky, D. Zheng, J. Du, J. O. Korbel, O. Emanuelsson, Z. D. Zhang, S. Weissman, and M. Snyder, "What is a gene, post-ENCODE? History and updated definition," *Genome Research*, vol. 17, no. 6, pp. 669–681, 2007. [Online]. Available: http://genome.cshlp.org/content/17/6/669.abstract

[6] D. Brett, H. Pospisil, J. Valcarcel, J. Reich, and P. Bork, "Alternative splicing and genome complexity," *Nat Genet*, vol. 30, no. 1, pp. 29–30, Jan 2002. [Online]. Available: http://dx.doi.org/10.1038/ng803

[7] J. D. Watson and F. H. C. Crick, "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid," *Nature*, vol. 171, no. 4356, pp. 737–738, Apr 1953. [Online]. Available: http://dx.doi.org/10.1038/171737a0

[8] F. Sanger, S. Nicklen, and A. R. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proceedings of the National Academy of Sciences*, vol. 74, no. 12, pp.

5463–5467, 1977. [Online]. Available: http://www.pnas.org/content/74/12/5463.abstract

[9] M. L. Metzker, "Sequencing technologies - the next generation." *Nature reviews. Genetics*, vol. 11, no. 1, pp. 31–46, 2010. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/19997069

[10] D. R. Smith, A. R. Quinlan, H. E. Peckham, K. Makowsky, W. Tao, B. Woolf, L. Shen, W. F. Donahue, N. Tusneem, M. P. Stromberg, D. A. Stewart, L. Zhang, S. S. Ranade, J. B. Warner, C. C. Lee, B. E. Coleman, Z. Zhang, S. F. McLaughlin, J. A. Malek, J. M. Sorenson, A. P. Blanchard, J. Chapman, D. Hillman, F. Chen, D. S. Rokhsar, K. J. McKernan, T. W. Jeffries, G. T. Marth, and P. M. Richardson, "Rapid whole-genome mutational profiling using next-generation sequencing technologies," *Genome Research*, vol. 18, no. 10, pp. 1638–1642, 2008. [Online]. Available: http://genome.cshlp.org/content/18/10/1638.abstract

[11] R. D. Morin, M. D. O'Connor, M. Griffith, F. Kuchenbauer, A. Delaney, A.-L. Prabhu, Y. Zhao, H. McDonald, T. Zeng, M. Hirst, C. J. Eaves, and M. A. Marra, "Application of massively parallel sequencing to microRNA profiling and discovery in human embryonic stem cells," *Genome Research*, vol. 18, no. 4, pp. 610–621, 2008. [Online]. Available: http://genome.cshlp.org/content/18/4/610.abstract

[12] D. S. Johnson, A. Mortazavi, R. M. Myers, and B. Wold, "Genome-Wide Mapping of in Vivo Protein-DNA Interactions," *Science*, vol. 316, no. 5830, pp. 1497–1502, 2007. [Online]. Available: http://www.sciencemag.org/content/316/5830/1497.abstract

[13] M. Bainbridge, R. Warren, M. Hirst, T. Romanuik, T. Zeng, A. Go, A. Delaney, M. Griffith, M. Hickenbotham, V. Magrini, E. Mardis, M. Sadar, A. Siddiqui, M. Marra, and S. Jones, "Analysis of the prostate cancer cell line LNCaP transcriptome using a sequencing-by-synthesis approach," *BMC Genomics*, vol. 7, no. 1, p. 246, 2006. [Online]. Available: http://www.biomedcentral.com/1471-2164/7/246

[14] K. V. Voelkerding, S. A. Dames, and J. D. Durtschi, "Next-Generation Sequencing: From Basic Research to Diagnostics," *Clinical Chemistry*, vol. 55, no. 4, pp. 641–658, 2009. [Online]. Available: http://www.clinchem.org/content/55/4/641.abstract

[15] T. C. Glenn, "Field guide to next-generation DNA sequencers," *Molecular Ecology Resources*, vol. 11, no. 5, pp. 759–769, 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1755-0998.2011.03024.x

[16] Z. Wang, M. B. Gerstein, and M. Snyder, "RNA-Seq: a revolutionary tool for transcriptomics." *Nature reviews. Genetics*, vol. 10, no. 1, pp. 57–63, Jan. 2009. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/19015660

[17] M. Nicolae, S. Mangul, I. I. Mandoiu, and A. Zelikovsky, "Estimation of alternative splicing isoform frequencies from RNA-Seq data." *Algorithms for molecular biology : AMB*, vol. 6, no. 1, p. 9, apr 2011. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/21504602

[18] J. Feng, W. Li, and T. Jiang, "Inference of isoforms from short sequence reads," *Journal of Computational Biology*, vol. 18, no. 3, pp. 305–321, 2011. [Online]. Available: http://www.liebertonline.com/doi/abs/10.1089/cmb.2010.0243

[19] S. Heber, M. A. Alekseyev, S.-H. Sze, H. Tang, and P. A. Pevzner, "Splicing graphs and EST assembly problem," in *Proc. 10th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB) (Suppl. of Bioinformatics)*, vol. 18, 2002, pp. 181–188.

[20] V. Lacroix, M. Sammeth, R. Guigó, and A. Bergeron, "Exact transcriptome reconstruction from short sequence reads," in *Proc. 8th Int. Workshop on Algorithms in Bioinformatics (WABI)*, ser. LNCS, K. A. Crandall and J. Lagergren, Eds., vol. 5251. Springer, 2008, pp. 50–63. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87361-7_5

[21] C. Trapnell, B. A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. J. van Baren, S. L. Salzberg, B. J. Wold, and L. Pachter, "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation," *Nature Biotechnology*, vol. 28, no. 5, pp. 516–520, may 2010. [Online]. Available: http://www.nature.com/doifinder/10.1038/nbt.1621

[22] M. Guttman, G. Manuel, J. Levin, J. Donaghey, J. Robinson, X. Adiconis, L. Fan, M. Koziol, A. Gnirke, C. Nusbaum, J. Rinn, E. Lander, and A. Regev, "From Ab initio reconstruction of cell type–specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs," *Nature Biotechnology*, vol. 28, pp. 503–510, 2010.

[23] G. Robertson, J. Schein, R. Chiu, R. Corbett, M. Field, S. Jackman, K. Mungall, S. Lee, H. Okada, J. Qian, M. Griffith, A. Raymond, N. Thiessen, T. Cezard, Y. Butterfield, R. Newsome, S. Chan, R. She, R. Varhol, B. Kamoh, A. Prabhu, A. Tam, Y. Zhao, R. Moore, and M. e. a. Hirst, "De novo assembly and analysis of RNA-seq data," *Nature Methods*, vol. 7, no. 5, pp. 909–912, 2010.

[24] D. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research*, vol. 18, pp. 821–829, 2008.

[25] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. a. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, Z. Chen, E. Mauceli, N. Hacohen, A. Gnirke, N. Rhind, F. di Palma, B. W. Birren, C. Nusbaum, K. Lindblad-Toh, N. Friedman, and A. Regev, "Full-length transcriptome assembly from RNA-Seq data without a reference genome," *Nature Biotechnology*, vol. 29, no. May, may 2011. [Online]. Available: http://www.nature.com/doifinder/10.1038/nbt.1883

[26] M. Garber, M. G. Grabherr, M. Guttman, and C. Trapnell, "Computational methods for transcriptome annotation and quantification using RNA-seq." *Nature methods*, vol. 8, no. 6, pp. 469–77, jun 2011. [Online]. Available: http://dx.doi.org/10.1038/nmeth.1613

[27] J. a. Martin and Z. Wang, "Next-generation transcriptome assembly," *Nature Reviews Genetics*, vol. 12, no. 10, pp. 671–682, Sep. 2011. [Online]. Available: http://www.nature.com/doifinder/10.1038/nrg3068

[28] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome." *Genome biology*, vol. 10, no. 3, p. R25, Jan. 2009. [Online]. Available: http://genomebiology.com/2009/10/3/R25

[29] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/25/14/1754.abstract

[30] K. Wang, D. Singh, Z. Zeng, S. Coleman, Y. Huang, G. Savich, X. He, P. Mieczkowski, S. Grimm, C. Perou, J. MacLeod, D. Chiang, J. Prins, and J. Liu, "MapSplice: accurate mapping of RNA-seq reads for splice junction discovery," *Nucleic Acids Research*, vol. 38, no. 18, p. e178, 2010.

[31] A. Fai, H. Jiang, L. Lin, Y. Xing, and W. Wong, "Detection of splice junctions from paired-end RNA-seq data by SpliceMap," *Nucleic Acids Research*, 2010.

[32] C. Trapnell, L. Pachter, and S. L. Salzberg, "TopHat: discovering splice junctions with RNA-Seq," *Bioinformatics*, vol. 25, no. 9, pp. 1105–1111, 2009. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp120

[33] T. D. Wu and S. Nacu, "Fast and SNP-tolerant detection of complex variants and splicing in short reads." *Bioinformatics (Oxford, England)*, vol. 26, no. 7, pp. 873–81, Apr. 2010.

[34] F. De Bona, S. Ossowski, K. Schneeberger, and G. Ratsch, "Optimal spliced alignments of short sequence reads," *BMC Bioinformatics*, vol. 9, no. Suppl 10,

p. O7, 2008. [Online]. Available: http://www.biomedcentral.com/1471-2105/9/S10/O7

[35] A. Mortazavi, B. A. Williams, K. McCue, L. Schaeffer, and B. Wold, "Mapping and quantifying mammalian transcriptomes by RNA-Seq." *Nature methods*, vol. 5, no. 7, pp. 621–8, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1038/nmeth.1226

[36] C. W. Sugnet, W. J. Kent, M. Ares, and D. Haussler, "Transcriptome and genome conservation of alternative splicing events in humans and mice," *Pac Symp Biocomput*, pp. 66–77, 2004, [PubMed:14992493].

[37] R. Guigó, P. Flicek, J. Abril, A. Reymond, J. Lagarde, F. Denoeud, S. Antonarakis, M. Ashburner, V. B. Bajic, E. Birney, R. Castelo, E. Eyras, C. Ucla, T. R. Gingeras, J. Harrow, T. Hubbard, S. E. Lewis, and M. G. Reese, "EGASP: the human ENCODE Genome Annotation Assessment Project." *Genome biology*, vol. 7, no. Suppl 1, pp. S2.1–31, jan 2006. [Online]. Available: http://dx.doi.org/10.1186/gb-2006-7-s1-s2

[38] S. Altschul, W. Gish, W. Miller, E. Myers, and J. Lipman, "Basic local alignment search tool," *J. of Molecular Biology*, vol. 215, pp. 403–410, 1990.

[39] Y. Pirola, R. Rizzi, E. Picardi, G. Pesole, G. Della Vedova, and P. Bonizzoni, "PIntron: a fast method for detecting the gene structure due to alternative splicing via maximal pairings of a pattern and a text," *BMC Bioinformatics*, vol. 13, no. Suppl 5, p. S2, 2012. [Online]. Available: http://www.biomedcentral.com/1471-2105/13/S5/S2

[40] J. Clemente, J. Jansson, and G. Valiente, "Flexible taxonomic assignment of ambiguous sequencing reads," *BMC Bioinformatics*, vol. 12, no. 1, p. 8, 2011. [Online]. Available: http://www.biomedcentral.com/1471-2105/12/8

[41] H. N. Poinar, C. Schwarz, J. Qi, B. Shapiro, R. D. E. MacPhee, B. Buigues, A. Tikhonov, D. H. Huson, L. P. Tomsho, A. Auch, M. Rampp, W. Miller, and S. C. Schuster, "Metagenomics to Paleogenomics: Large-Scale Sequencing of Mammoth DNA," *Science*, vol. 311, no. 5759, pp. 392–394, 2006. [Online]. Available: http://www.sciencemag.org/content/311/5759/392.abstract

[42] Z. Liu, T. Z. DeSantis, G. L. Andersen, and R. Knight, "Accurate taxonomy assignments from 16S rRNA sequences produced by highly parallel pyrosequencers," *Nucleic Acids Research*, vol. 36, no. 18, p. e120, 2008. [Online]. Available: http://nar.oxfordjournals.org/content/36/18/e120.abstract

[43] T. Z. DeSantis, P. Hugenholtz, K. Keller, E. L. Brodie, N. Larsen, Y. M. Piceno, R. Phan, and G. L. Andersen, "NAST: a multiple sequence alignment

server for comparative analysis of 16S rRNA genes," *Nucleic Acids Research*, vol. 34, no. suppl 2, pp. W394–W399, 1 July 2006. [Online]. Available: http://nar.oxfordjournals.org/content/34/suppl_2/W394.abstract

[44] P. J. Turnbaugh, M. Hamady, T. Yatsunenko, B. L. Cantarel, A. Duncan, R. E. Ley, M. L. Sogin, W. J. Jones, B. A. Roe, J. P. Affourtit, M. Egholm, B. Henrissat, A. C. Heath, R. Knight, and J. I. Gordon, "A core gut microbiome in obese and lean twins," *Nature*, vol. 457, no. 7228, pp. 480–484, Jan 2009. [Online]. Available: http://dx.doi.org/10.1038/nature07540

[45] P. Ribeca and G. Valiente, "Computational challenges of sequence classification in microbiomic data," *Briefings in Bioinformatics*, vol. 12, no. 6, pp. 614–625, 2011. [Online]. Available: http://bib.oxfordjournals.org/content/12/6/614.abstract

[46] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/26/5/589.abstract

[47] W. Kent, "BLAT-The BLAST-Like Alignment Tool," *Genome Research*, vol. 12, pp. 656–664, 2002.

[48] P. Ribeca, "Short-Read Mapping," in *Bioinformatics for High Throughput Sequencing*, N. Rodríguez-Ezpeleta, M. Hackenberg, and A. M. Aransay, Eds. Springer New York, 2012, pp. 107–125, 10.1007/978-1-4614-0782-9_7. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-0782-9_7

[49] J. G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Pena, J. K. Goodrich, J. I. Gordon, G. A. Huttley, S. T. Kelley, D. Knights, J. E. Koenig, R. E. Ley, C. A. Lozupone, D. McDonald, B. D. Muegge, M. Pirrung, J. Reeder, J. R. Sevinsky, P. J. Turnbaugh, W. A. Walters, J. Widmann, T. Yatsunenko, J. Zaneveld, and R. Knight, "QIIME allows analysis of high-throughput community sequencing data," *Nat Meth*, vol. 7, no. 5, pp. 335–336, May 2010. [Online]. Available: http://dx.doi.org/10.1038/nmeth.f.303

[50] P. D. Schloss, S. L. Westcott, T. Ryabin, J. R. Hall, M. Hartmann, E. B. Hollister, R. A. Lesniewski, B. B. Oakley, D. H. Parks, C. J. Robinson, J. W. Sahl, B. Stres, G. G. Thallinger, D. J. Van Horn, and C. F. Weber, "Introducing mothur: Open-Source, Platform-Independent, Community-Supported Software for Describing and Comparing Microbial Communities," *Applied and Environmental Microbiology*, vol. 75, no. 23, pp. 7537–7541, December 1, 2009. [Online]. Available: http://aem.asm.org/content/75/23/7537.abstract

[51] D. H. Huson, A. F. Auch, J. Qi, and S. C. Schuster, "MEGAN analysis of metagenomic data," *Genome Research*, vol. 17, no. 3, pp. 377–386, 2007. [Online]. Available: http://genome.cshlp.org/content/17/3/377.abstract

[52] S. Federhen, "The NCBI taxonomy database," *Nucleic Acids Research*, vol. 40, no. D1, pp. D136–D143, 2012. [Online]. Available: http://nar.oxfordjournals.org/content/40/D1/D136.abstract

[53] D. Alonso-Alemany, J. Clemente, J. Jansson, and G. Valiente, "Taxonomic Assignment in Metagenomics with TANGO," *EMBnet.journal*, vol. 17, no. 2, 2011. [Online]. Available: http://journal.embnet.org/index.php/embnetjournal/article/view/237

[54] M. Santamaria, B. Fosso, A. Consiglio, G. De Caro, G. Grillo, F. Licciulli, S. Liuni, M. Marzano, D. Alonso-Alemany, G. Valiente, and G. Pesole, "Reference databases for taxonomic assignment in metagenomics," *Briefings in Bioinformatics*, 2012. [Online]. Available: http://bib.oxfordjournals.org/content/early/2012/07/10/bib.bbs036.abstract

[55] J. R. Cole, Q. Wang, E. Cardenas, J. Fish, B. Chai, R. J. Farris, A. S. Kulam-Syed-Mohideen, D. M. McGarrell, T. Marsh, G. M. Garrity, and J. M. Tiedje, "The Ribosomal Database Project: improved alignments and new tools for rRNA analysis," *Nucleic Acids Research*, vol. 37, no. suppl 1, pp. D141–D145, 2009. [Online]. Available: http://nar.oxfordjournals.org/content/37/suppl_1/D141.abstract

[56] D. McDonald, M. N. Price, J. Goodrich, E. P. Nawrocki, T. Z. DeSantis, A. Probst, G. L. Andersen, R. Knight, and P. Hugenholtz, "An improved greengenes taxonomy with explicit ranks for ecological and evolutionary analyses of bacteria and archaea," *ISME J*, vol. 6, no. 3, pp. 610–618, Mar 2012. [Online]. Available: http://dx.doi.org/10.1038/ismej.2011.139

[57] E. Pruesse, C. Quast, K. Knittel, B. M. Fuchs, W. Ludwig, J. Peplies, and F. O. Glöckner, "SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB," *Nucleic Acids Research*, vol. 35, no. 21, pp. 7188–7196, 2007. [Online]. Available: http://nar.oxfordjournals.org/content/35/21/7188.abstract

[58] P. Yarza, M. Richter, J. Peplies, J. Euzeby, R. Amann, K.-H. Schleifer, W. Ludwig, F. O. Glöckner, and R. Rosselló-Móra, "The All-Species Living Tree project: A 16S rRNA-based phylogenetic tree of all sequenced type strains," *Systematic and Applied Microbiology*, vol. 31, no. 4, pp. 241 – 250, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S072320200800060X

[59] P. Yarza, W. Ludwig, J. Euzéby, R. Amann, K.-H. Schleifer, F. O. Glöckner, and R. Rosselló-Móra, "Update of the All-Species Living Tree Project based on 16S and 23S rRNA sequence analyses," *Systematic and Applied Microbiology*, vol. 33, no. 6, pp. 291 – 299, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0723202010001025

[60] R. Munoz, P. Yarza, W. Ludwig, J. Euzéby, R. Amann, K.-H. Schleifer, F. O. Glöckner, and R. Rosselló-Móra, "Release LTPs104 of the All-Species Living Tree," *Systematic and Applied Microbiology*, vol. 34, no. 3, pp. 169 – 170, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0723202011000749

[61] J. Fischer and D. H. Huson, "New common ancestor problems in trees and directed acyclic graphs," *Inf. Process. Lett.*, vol. 110, no. 8-9, pp. 331–335, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ipl.2010.02.014

[62] D. Alonso-Alemany and G. Valiente, "LCA Skeleton Tree in Linear Time," 2012, submitted.

[63] D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM J. Comput.*, vol. 13, no. 2, pp. 338–355, May 1984. [Online]. Available: http://dx.doi.org/10.1137/0213024

[64] M. A. Bender and M. Farach-Colton, "The LCA Problem Revisited," in *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, ser. LATIN '00. London, UK, UK: Springer-Verlag, 2000, pp. 88–94. [Online]. Available: http://dl.acm.org/citation.cfm?id=646388.690192

[65] J. E. Stajich, D. Block, K. Boulez, S. E. Brenner, S. A. Chervitz, C. Dagdigian, G. Fuellen, J. G. Gilbert, I. Korf, H. Lapp, H. Lehväslaiho, C. Matsalla, C. J. Mungall, B. I. Osborne, M. R. Pocock, P. Schattner, M. Senger, L. D. Stein, E. Stupka, M. D. Wilkinson, and E. Birney, "The Bioperl Toolkit: Perl Modules for the Life Sciences," *Genome Research*, vol. 12, no. 10, pp. 1611–1618, 2002. [Online]. Available: http://genome.cshlp.org/content/12/10/1611.abstract