POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree Course in Computer Engineering

Master Degree Thesis

A modern reimplementation of an alignment pipeline for the analysis and quantification of small non-coding RNA and isoforms using C++ and Python

Supervisor
Gianvito Urgese

Candidate
Marco Capettini

March/April 2020

Abstract

During the last years computer science has taken on an increasingly central role in the processes underlying the production and analysis of biological data. The continuous development of new cutting-edge machines such as NGS has made it possible to make great progress in the field of genetic sequence analysis. For this reason, and also due to the enormous amount of data produced daily with these procedures, many algorithms and tools developed for the analysis need to be optimized for exploiting the enhanced features of new computing systems.

With this thesis work I propose a modern reimplementation of an alignment tool called isomiR-SEA which was developed with a precise objective in mind: overcoming some of the limitations of today's general-purpose alignment algorithms, that usually lack accuracy and completeness in the results. The first version of the tool was designed to detect and quantify small non-coding RNA sequences (microRNAs) and their variants isomiRs. Such sequences are provided to the program as simple text substrings composed of combinations of A, C, T and G characters, which represent very short segments of RNA made up of about 20-22 nucleotides. These small sequences play a critical role in gene expression because of their regulatory functions on the production of proteins. It is in fact widely proven that they are fundamental in several cellular processes and, as a consequence, in the onset and progression of many diseases such as immune disorders and cancer.

isomiR-SEA algorithm was developed in C++14, written in a non-optimized way and not completely tested. So, in order to make it usable by the bioinformatics community, there was a strong need for software re-engineerization and bug-correction. For this reason I decided to reimplement the software by conforming to the modern C++17 programming standard and to SeqAn3, the today's latest version of the library for the analysis of biological sequences which replaces SeqAn2, used in the old version of isomiR-SEA.

Besides fixing bugs, I have implemented several new features such as the serialization of the input reference databases, in order to save time in consecutive executions, and the possibility of providing only a single file as input to the program representing the union of several smaller ones, allowing to obtain with a single execution the same results that before would have required many more execution cycles.

This, together with a revised data printing mechanism which originally wasted a large amount of resources saving temporary structures in memory, has allowed to switch from a first prototype of the tool to a working version tested in an environment very close to the intended one, providing a product that is currently usable by an end bioinformatician user. The new version of isomiR-SEA achieved a significant increase in performance by gaining both in terms of execution times (up to ~60%) and drastically decreasing max RAM consumption (by ~75%).

Finally, I dealt with the post analysis of the output data, porting into Python3 scripts what were previously implemented using Knime, a software useful to create and productionize data science using intuitive environment. Although Knime is very convenient for prototyping thanks to its intuitive and model-oriented graphical interface, it flaws in terms of efficiency and performance when compared to Python.

Contents

List of Figures

Chapter 1

Introduction

1.1 Bioinformatics: what is it?

Bioinformatics is a discipline that deals with developing new techniques, algorithms, and software tools for analyzing biological data. It was officially defined for the first time in 1970 as a "study of informatic processes in biotic systems" [1] and today it's characterized by mathematical, statistical and computational methods for molecular biology to study and understand the biochemical and biophysical processes that underlie life, and to solve the problems deriving from the management and analysis of biological data.

Producing these kind of data is essential in biomedicine to fully understand the internal mechanism of diseases, but it is also useful to enable much more accurate diagnoses and to be more precise in identifying targets both at the microscopic and macroscopic level. However, due to this, data are often heterogeneous and their quantity is continuously and exponentially growing. The only way to be able to deal with this immense amount of information is the constant and continuous development of new methods of analysis based on computer science's principles, so we are not only talking about algorithms per se, but also about all those tricks and computational techniques that concern selection of data structures suitable for the context, and advanced level programming.

Such data structures can be seen as containers of encoded biological information, and in the digital world they can have different shapes and features depending on the application field for which they are used. For example, they can be simple Excel tables, more complex SQL databases, graphic representations such as nodes in a network or vertices of a graph. On the other hand if you want to see them from a low level point of view they can also be seen as variables and constructs within the code of a program.

This allows us to relate directly to them, making navigation and interrogation easier and allowing us to reconstruct the internal logics of the phenomena of interest. Finally is extremely important because the exploration and observation of data can

bring out characteristics that otherwise would have been difficult to notice.

So the figure of the bioinformatician refers to the one who guides the generation of data step by step, transforms them so that they can be a clearer and more defined source of information and finally extrapolates knowledge from them.

1.2 Background

All the information provided in this section is useful to guarantee the reader a comprehensive theoretical compendium in order to understand the subsequent steps, to better frame the work done and to provide a broad picture of this area of study. In particular, we explain the procedure for generating the processed data, continuing with a focus on specific algorithms for the field of sequence alignment, and an in-depth analysis regarding the fundamental biological theme: small non-coding RNA and isoforms. Finally, we list the main file formats used.

1.2.1 DNA/RNA sequencing

The purpose of sequencing is to discover the sequence of nitrogenous bases (adenine, cytosine, guanine and thymine/uracil) that alternate within a molecule.

To do this, the DNA is first divided into many small fragments and each of these is analyzed and translated into nucleotide sequences using special sequencing machines. On the other hand, to sequence RNA, it is necessary to first reverse transcribe it into complementary DNA (cDNA) [2]. This is because DNA molecules are more stable, they allow the amplification process which, as we will see, uses DNA polymerases, and above all a more mature DNA sequencing technology can be exploited.

Anyway, all these procedures are not free from errors in the sense that the identified sequences can reflect more or less precisely the real reference sequence. So often for this reason, together with the sequenced fragments which are called reads and are in the form of string of characters, it is reported their nucleotide by nucleotide quality (see 1.2.4).

Depending on how they are generated, there are two types of reads [3]:

- **single-end reads**: the fragments are sequenced from one end to the other, completely, thus producing single reads of variable length;
- **paired-end reads**: the fragments are sequenced starting from both the ends for a certain number of bases, producing two different segments for each read, called mates. The two mates have a mutual distance in terms of nucleotides and knowing it can be very useful for the alignment phases. These types of sequences are useful for specific applications like the recognition of gene fusions.

When working with small non-coding RNA reads instead it is essential, before proceeding with sequencing, to filter the genomic material according to the length of each sequences. This must be done because usually these sequences are of limited length, and it is therefore possible to discard all those outside this range, considerably reducing the amount of data to be analyzed. Such filtering involves the use of coloured markers for each sequence of suitable length during a gel electrophoresis process. In this process the sequences of interest are inserted into a fluid to which a charge is applied, allowing them to be positioned at different levels depending on their length, so that those that are too long or short can be discarded, being able to proceed with the small RNA sequencing [4].

Sequencing has always been a long and expensive process, for this reason over the years techniques have refined and the set of those most used today takes the name of **Next Generation Sequencing** (**NGS**). When we talk about NGS technologies we refer to methods for sequencing whole genomes, transcriptomes, or even small non-coding RNA sequences. NGS machines are defined parallel machines because, after the fragmentation into sub-sequences, copies of the same are created (amplification process) which are then analyzed in parallel. This is very useful to decrease the number of errors, which can be both in the amplification and in the sequencing phase, and to have sequences more consistent with reality (better quality).

The **amplification phase** is a technique that is called PCR [5] and involves three phases (*Figure 1.1*):

- **Denaturation**: the double helix is divided into two separate strands, each complementary to the other, which will be copied;
- **Annealing**: *primers* are attached to the ends of each fragment, they are very short sequences that act as anchors, that is, as a starting point for the copy (i.e. for the binding of the subsequent bases);
- **Elongation**: a different nucleotide at a time is added manually to the solution, if it is complementary to that one present on the reference sequence at the point where the production of the copy-fragment arrived, then it will bind to the sequence that is being formed, otherwise not. By adding these bases step by step the DNA polymerase synthesizes a new DNA strand complementary to the DNA template.

The process is done for both strands.

However, primers are complementary sequences with respect to the small segment of DNA template to which they bind, but not yet knowing the nucleotide sequence of this DNA molecule, how do you choose the right primers? For this reason there is the **preparation phase**, prior to amplification. Here we attach to the reference strand (the one from which the copy fragments are produced) two adapters: one at the beginning of the strand which will be used for the link of the

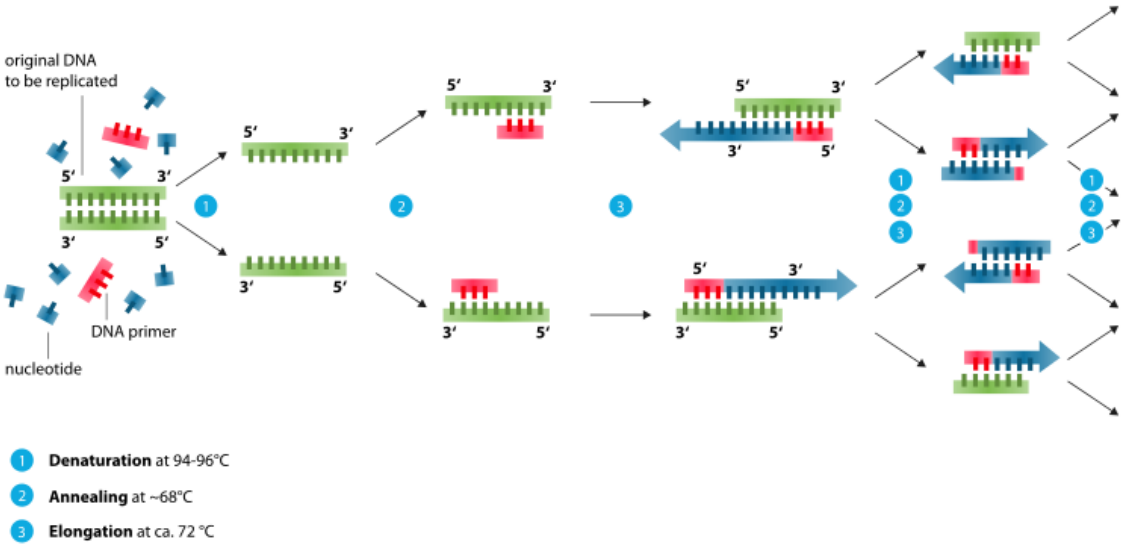


Figure 1.1: **Polymerase Chain Reaction** [6].

primer for amplification, and one at the end of the strand which will be used for the link of the primer for sequencing. These adapters are small sequences exactly complementary to the primer sequences that we want to attach.

Finally there is the **sequencing phase**. To date there are numerous and different sequencing techniques, some of them are based on fluorescence such as *Illumina Sequencing* [7], some on pH variation as *Ion Semiconductor Sequencing* [8], other on the production of light as *Pyrosequencing* [9]. The latter is a procedure very similar to the just described PCR, with the difference that with this technique, each time a group of identical nucleotides is inserted in the solution, if one or more of these manage to bind in the correct position of the strand that is forming an amount of Pyrophosphate (PPi) will be released, and this amount is proportional to the number of nucleotides that have bound. This PPi will then be fundamental in the chemical reaction which will produce a more or less intense light captured by a camera. Each time, depending on the intensity of the light produced, it is possible to understand how many identical nucleotides have linked consecutively. Then with the action of Apyrase the solution is "cleaned", i.e. the excess unbound nucleotides are eliminated and the solution is ready to receive the next group of nucleotides. In this way, step by step, all the light intensities captured are reported on a graph called Pyrogram, from which it is possible to trace the actual nucleotide string which will then be reported on text files, e.g. fasta, fastq, ... (*Figure 1.2*).

At this point one of the possible steps that follow sequencing is the reconstruction of the entire original genome/transcriptome through the assembly or alignment of the reads [3].

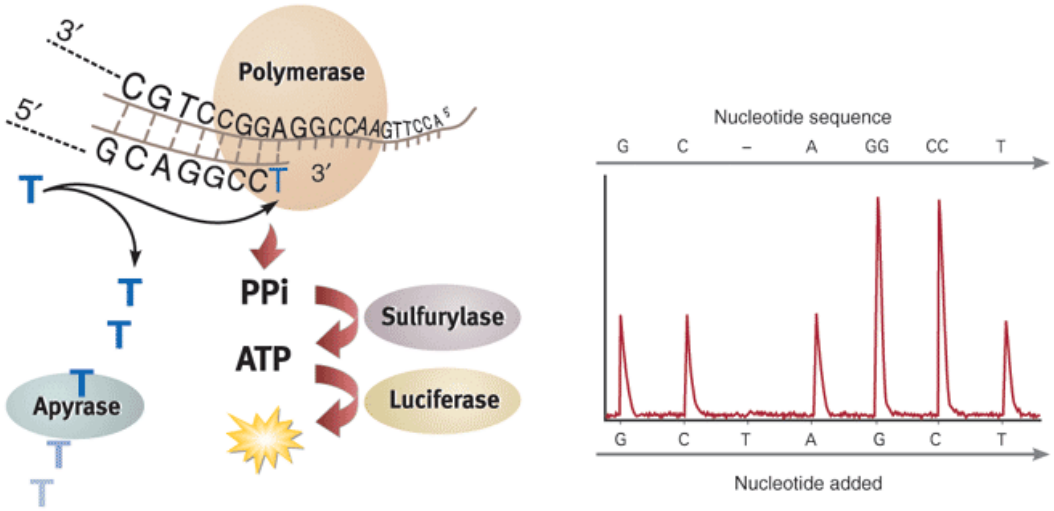


Figure 1.2: **Pyrosequencing**. On the left the strand being synthesised along the reference sequence, on the right the pyrogram: peaks means binding of a nucleotide, higher peaks means binding of several identical nucleotides consecutively [10].

- **Assembly**: it's an iterative and time consuming process in which we do not know what to expect, like when we do a puzzle without peeking at how the final picture should come. The reads are assembled with each other by overlapping them onto similar parts they have in common, for example the series of nucleotides at the end of a sequence can be identical to the series of nucleotides at the beginning of another sequence. Two or more assembled reads form a new longer sequence which is called *contig*. Then the contigs will be assembled to form longer sequences called *scaffolds*. Finally the latter are joined to form the final assembly. This procedure is done each time we face for the first time a genome never sequenced, because the "big picture" is not yet available and therefore the sequences must be assembled trying to discover the genome and use it as a reference.

- **Alignment**: it's a simpler process because you already know what the final result will be. Usually for alignment we use DNA as a reference even if we have RNA data (reads). In particular if we sequence DNA then we use the DNA/genome as a reference, if we sequence RNA we can use the DNA/genome as a reference or even the transcriptome (mRNA). The transcriptome is the set of all the transcripts that we can have, the transcripts are all the mRNAs that are provided by a cell. A single gene can produce several transcripts and therefore several mRNAs (isoforms). In any case when we try to align the reads that come from a certain subject using a DNA/transcriptome as a reference, this cannot be exactly identical to the assembly of our reads, because the DNA/transcriptome of each living being is different (even

if from the same organism). So the reference is used as a *guide* in reconstruction. These differences can be many and evident, but reads usually come from samples of sick creatures, cancer cells and so on, and understanding what the differences and similarities are compared to a reference genome is extremely important.

1.2.2 Sequence alignment algorithms

Usually the next step after producing data with NGS technologies is the analysis of the extracted sequences, this is done by aligning them on a reference database of well-known sequences. Sequence alignment is an extremely important bioinformatic process that allows you to compare two or more segments of DNA, RNA or even amino acids with the aim of identifying similar or even identical regions so as to be able to infer functionalities.

To find such correspondences, it would be unthinkable to look at and compare the tertiary structure of the atoms of the molecules in exam, also taking in account the enormous amount of data produced disproportionately with the swiftly evolving sequencing technologies. So what you usually do is to compare their respective strings of characters. When you find an important "similarity" then you can carry on with more in-depth post-analysis.

During the alignment phase you must consider four fundamental biological events that can occur by comparing the sequence under examination, referred to as the *query*-sequence, and the *reference* sequence (sometimes called *subject*-sequence): Conservation, Substitution, Insertion and Deletion. *Conservation* occurs when the two letters in question are identical and this event is defined **Match**, while *Substitution* occurs when these letters are different, and this is called **Mismatch**. Finally there is *Insertion* or *Deletion* when in one of the two aligned sequences a letter is aligned with a **Gap** (*Figure 1.3*).

(Qry)	A C D E F G	A C D E F G	A C D E F G	A C -- E F G
(Sbj)	A C D E F G	A C L E F G	A C -- E F G	A C D E F G
Biological event	**Conservation**	**Substitution**	**Insertion**	**Deletion**
Alignment represent	**Match**	**Mismatch**	**Gap**	**Gap**

Figure 1.3: **Biological events and their representation in alignment** [11].

Anyway, the position of matches, mismatches and gaps detected during the alignment phase depends on the *scoring scheme/gap models* used and on the adopted *alignment algorithm*. The scoring scheme is the set of rules used to define the yardstick of matches and mismatches, and there exist different types of them such as

those based on the Levenshtein [12] or Hamming distance [13] or those who use Susbtitutional Matrices like PAM [14] or BLOSUM [15]. The gap model instead helps to decide how to consider insertion and deletion events, examples are: Linear, Affine, Convex [16] or even Dynamic Gap Model (an optimized version of the Affine Gap designed by Urgese and called Dynamic Gap Selector [17]).

However, a fundamental role in the alignment mechanism is covered by the chosen algorithm: over the years many have been developed, each one with its own features, below there is a list of the most noteworthy ones.

Needleman-Wunsch algorithm

It was developed and published in the early 1970s [18] and is usually considered to be one of the first application of dynamic programming to sequence analysis. Dynamic programming is an algorithm-design technique that is based on the idea of dividing the main problem into many small sub-problems and then proceeding to solve each of them by finding optimal solutions which are then used to build the solution to the main proposed problem. For this reason it is often referred to as the *optimal matching algorithm*. Furthermore, this type of algorithm is able to generate global alignments, i.e. the sequences are aligned to each other according to their entire length (or inserting gaps if they have different lengths), this allows alignments with the maximum number of elements aligned between two sequences.

As shown in *Figure 1.4*, to produce the alignment we start by building a score matrix. In particular, the two strings are inserted one letter per cell in the grid starting from the third column (horizontally) and the third row (vertically). At this point the scores for match, mismatch and indel (i.e. insertion / deletion) must be established, and the second column and row are filled starting from zero and adding each time the value of the indel. Then we proceed to fill the table according to the system of equations reported in the central part of the figure: the value of each cell is calculated starting from the maximum value among its three adjacent cells, also considering the specific biological event with respect to the current indices. Each time a new cell is calculated, the link to the cell with the maximum value is registered with an arrow. Once the matrix is complete, starting from the cell on the bottom right back to the cell on the top left we mark the path following the arrows, and the alignment is calculated so:

- diagonal arrow represents a match or mismatch;
- horizontal or vertical arrow represents an indel (horizontal arrows means a gap to the letter of the row, vertical arrows means a gap to the letter of the column;
- multiple arrows indicate that different alignments are possible.

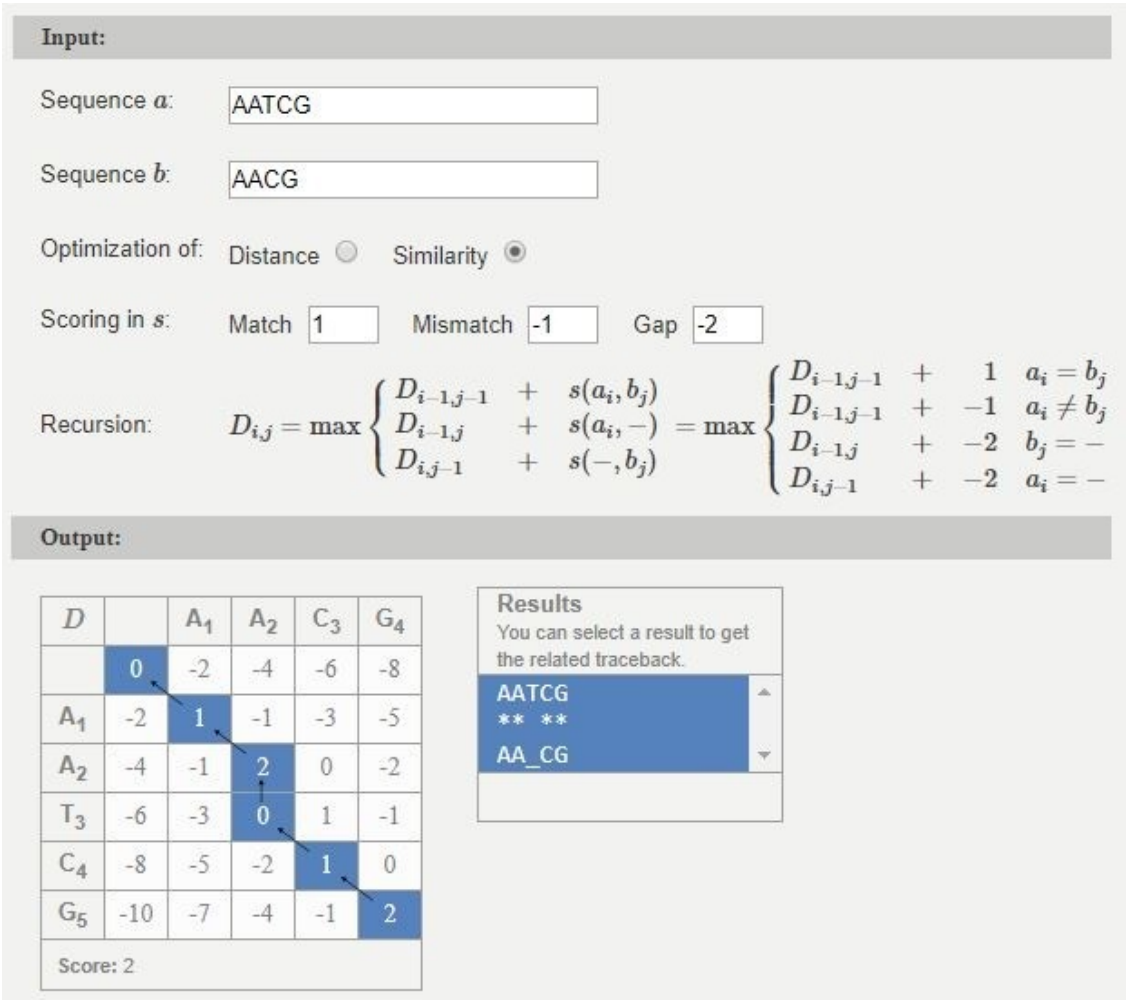


Figure 1.4: **Needleman-Wunsch algorithm and matrix of scores:** input section shows the sequences under considerations together with the scores associated with each biological event and the mathematical recursion used to calculate each cell in the grid; output section shows the resulting matrix highlighting the path of the alignment [19].

Smith-Waterman algorithm

As the former, is an application of dynamic programming and guarantees to find optimal alignments, but there is an important difference: the produced alignments are local. With a local alignment you encourage localized similarities in finite regions, therefore not extended to the whole sequence. This algorithm was developed in 1981 [20] and is usually referred to as a variant of the Needleman-Wunsch, in fact the process is very similar but in this case the negative cells of the scoring matrix are set to zero (see *Figure 1.5*). This allows the local alignments to be (positive

and) visible. Also the traceback procedure is different: we start from the cell with the higher score proceeding until we meet a cell with zero score. The traceback phase can be repeated starting from another cell with a score equal to or less than the maximum, to determine the other possible local alignments.

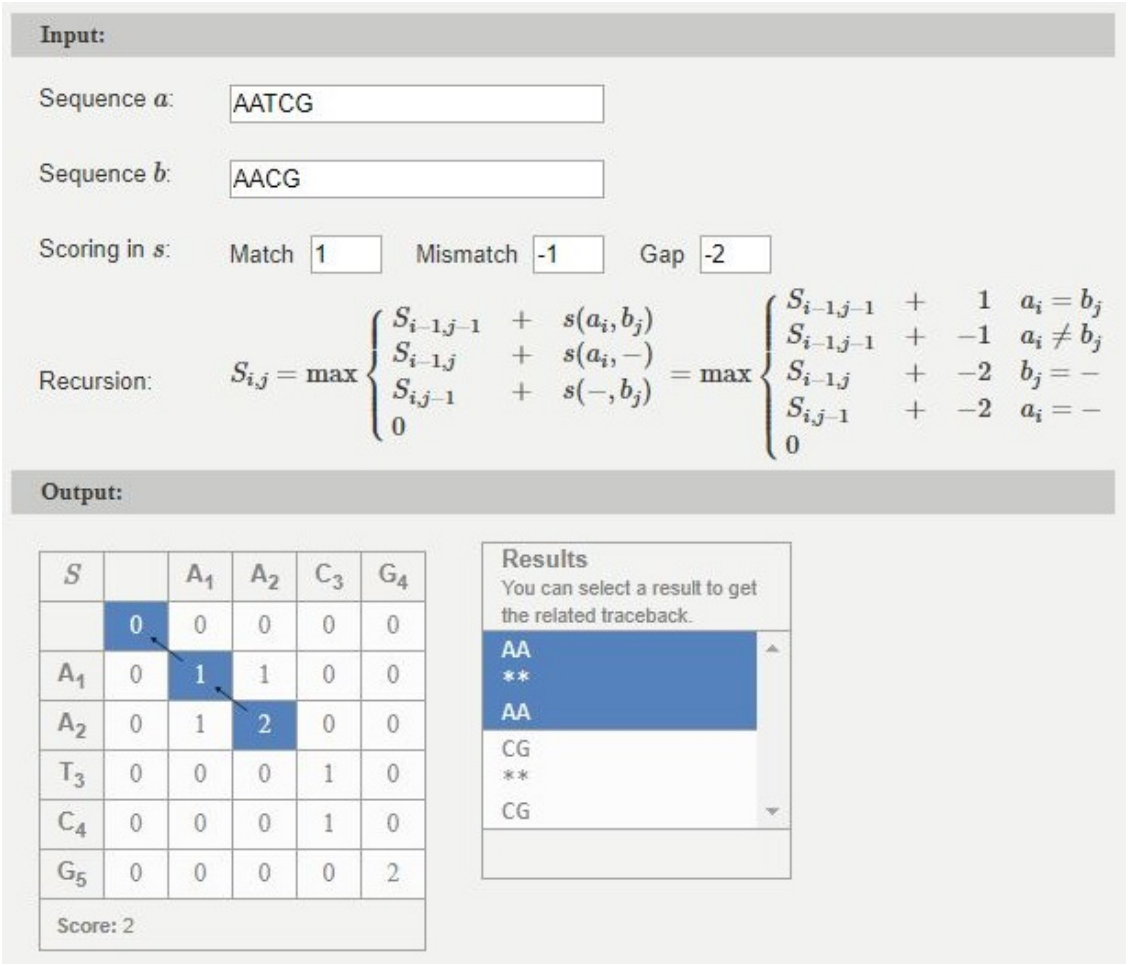


Figure 1.5: **Smith-Waterman algorithm and matrix of scores:** input section shows the sequences under considerations together with the scores associated with each biological event and the mathematical recursion used to calculate each cell in the grid; output section shows the resulting matrix highlighting the path of the alignment [19].

Seed&extension algorithms

Since usually sequences are many and very long, it is very expensive to use the algorithms just described, which have a quadratic complexity in time and space. For this reason, computationally more efficient alternatives were developed between

the 80s and 90s that adopt filtering strategies. In other words, instead of exploring the entire space of possibilities, we start from a small match to discriminate and work on a smaller amount of data, after which we extend the alignment to be able to make more specific and dynamic analyses.

The common feature of these algorithms is that they use a heuristic approach, that is employ a practical method that is not guaranteed to produce optimal solutions, but that it is extremely faster. Basic Local Alignment Search Tool (BLAST) [21] is probably the most prominent tool of this kind. It is a program that allows you to compare biological sequences such as those of amino acids, proteins or DNA/RNA nucleotides by first locating short and local matches between sequences (this is called *seeding*). The algorithm consists of several steps [22], which can be summarized as follows:

- **List of w-letter words** (*Figure 1.6* A): starting from the query-sequence, strings of length w are extracted one after the other following the letters from left to right. For each of these words neighborhood words are generated using specific scoring matrix such as BLOSUM [15] and among these only those that have a score greater than a certain threshold are selected.
- **Scan the database for exact matches** (*Figure 1.6* B): here the program begins to look for the selected words within each single reference-sequence.
- **Extend the exact matches** (*Figure 1.6* C): every time an exact match is found, the query word is used as an anchor for the alignment, which is extended in both directions until the score begins to decrease. Each alignment is called Maximal Segment Pair (MSP).
- **Evaluate the MSPs**: at this point, a value that establishes its statistical significance is assigned to each MSP, this is useful to filter the results.

Index based algorithms

Although seed-based alignment has significantly reduced search times compared to the first algorithms based on dynamic programming, it is still a slow process. For this reason, starting from the 90s, we began to look for other approaches like genome indexing systems. Creating an index is as useful as it is in any other application, just think about the index of a book: if you want to know on which page a certain word appears it is much more efficient to search for it on the index or glossary rather than browsing the book page after page. The same is true for alignments, where creating an index of the reference database allows you to search the query sequence on a much smaller space, saving both time and memory.

Usually these indexing systems are preceded by completely invertible transforms in order to have more efficiency in the research phase. Transforms like that of Burrows-Wheeler (BWT) [23] are in fact useful because they permute the order of

the text or the sequence to have a single character repeated multiple times in a row (ex: ^BANANA| becomes BNN^AA|A), and this allows a greater space reduction during the compression phase. However, the huge advantage that you have over a simple alphabetical ordering (which allows you to compress the data easily too) is that in this way you preserve the possibility of tracing back to the original data (reversability).

Bowtie [24] is a famous sequence alignment software that best exemplifies the process just described: it uses a permanent and reusable index of the genome (for the human genome the index is about 2.2 GB) to allow extremely fast read alignments. In particular it's based on the FM-Index which is an index combining the BWT with a few small auxilliary data structures in order to allow compression of the reference text and fast lookup for queries. However there are also other tools that use the Burrow-Wheeler transform to create an index of the genome, for example BWA [25] which is a bit slower than Bowtie but, unlike it, allows gapped alignment.

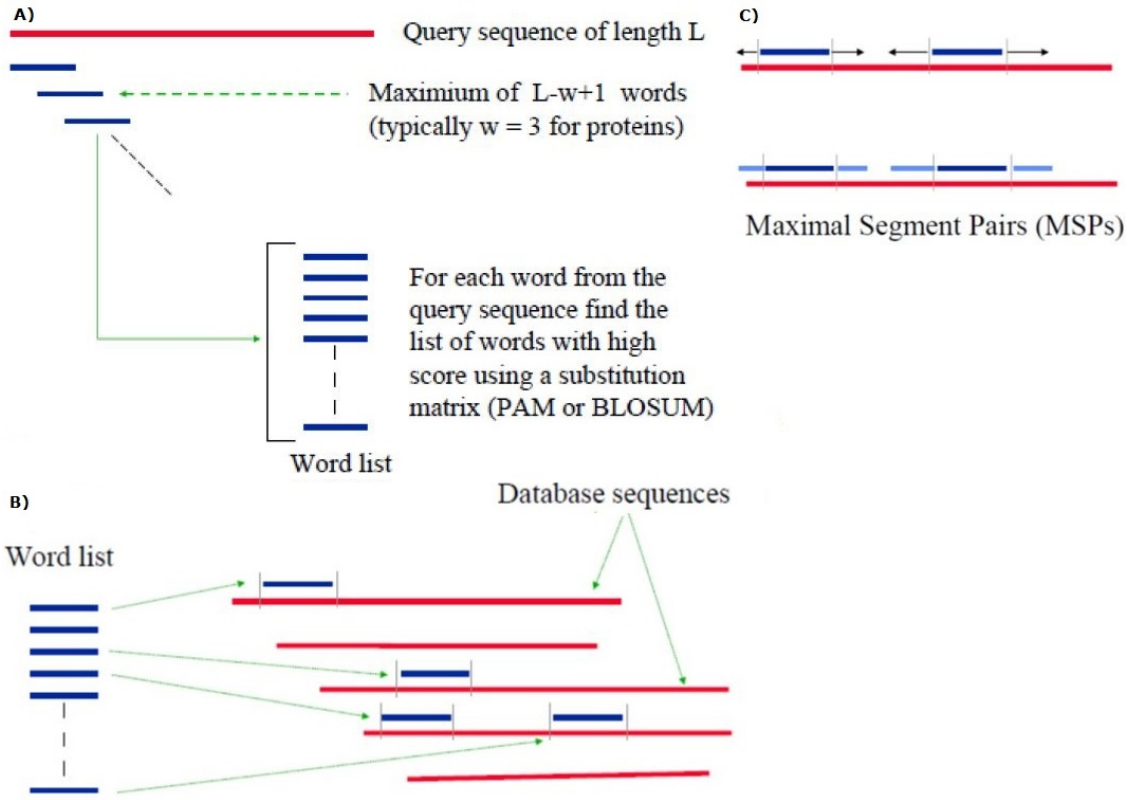


Figure 1.6: **BLAST: step of the algorithm** A) Words extraction. B) Matches of words of the list to the database sequences. C) Extension of the alignment in both directions to find high score segments. [26].

1.2.3 Small non-coding RNA and isoforms

Thanks to the advent of NGS technologies, today we know that more than 90% of the human genome is transcribed. Although trascription is the first important step in gene expression, only about 2% of the genetic material is made up of genes that encode for proteins [27][28], this means that a huge portion of non-coding transcriptome remains. Initially these sequences were considered almost as garbage, so much that they were even called "junk", but recently it is becoming clear that this transcriptome has a fundamental role in the physiological cellular development and in the onset of many diseases both in animals and plants by triggering or inhibiting certain cell functions. The fact that there are several cases of RNA molecules that play a fundamental role in this sense has certainly prompted researchers and scientists to investigate the topic and, depending on the length of the transcript, today two main classes of non-coding RNA can be identified: long and small. In particular among the latter there are very short sequences that play a critical role in regulating gene expression: they are called microRNAs (miRNAs).

Biogenesis of microRNA and its isoforms

miRNAs are short segments of non-coding RNA made-up of about 20-22 nucleotides. Their genesis involves several stages of development [28] that begin with the transcription of sequences of up to a hundred nucleotides named *hairpins*, for their peculiar shape. Usually these hairpins are called primary transcripts (pri-miRNA) and they are cleaved before being exported to the cytoplasm (pre-miRNA). At this point, these small agglomerations are again cleaved into strands which are identified as the mature microRNA sequences (see *Figure 1.7*).

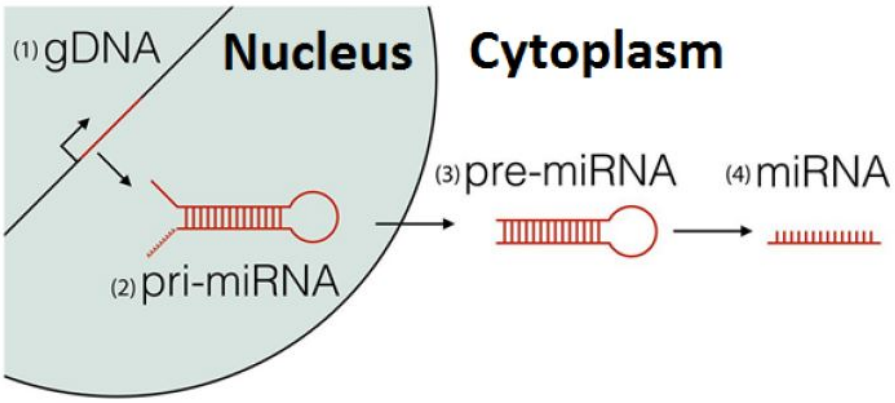


Figure 1.7: **Main steps in miRNA biogenesis** [28].

Initially it was thought that for each pre-miRNA molecule only two well-known kind of strand could be produced, in particular one from the 5' arm (5p strand) and the other from the 3' arm (3p strand). However, also thanks to the new sequencing

technologies introduced in recent times, we now know that there can be many and distinct variants of a single miRNA that can differ both in terms of length (quantity of nucleotides) and composition: these are usally called isomiRs. This isoforms are slight mutations due to chemical processes taking place on site during the two cleavage steps from pri-miRNA to pre-miRNA and from pre-miRNA to mature miRNA. To date we know that there exist three main types of isomiR called 5' isomiR, 3' isomiR and single-nucleotide polymorphism (SNP). In particular the first two are mutations resulting from insertion or deletion respectively in the 5' end and in the 3' end of the mature sequence while SNPs are isoforms where the sequence presents a mismatch with respect to mature one.

miRNA / isomiRs regulatory functions and interaction sites

Both miRNAs and isomiRs are fundamental in the post transcriptional phases where they participate in the downregolation of protein expression. This is possible because these small sequences interact with specific target mRNA through base paring, which means that they bind to specific portion of the messenger RNA forming bounds between subsequent nucleotides. These small non-coding segments have this capability of binding to specific positions of an RNA molecule thanks to the conservation (or not) of some interesting nucleotide subsquences, such small groups of bases identify the so-called interaction sites (see *Figure 1.8*) among which we find:

- **Seed site**: from nucleotide 2 to 7, it is certainly the most important and critical region because an uncorrupted seed sequence can act as an anchor to ensure the miRNA-mRNA complex stability [29], and therefore for this reason it is extremely useful for miRNA target recognition. Moreover, it is a region that is preserved within the various isoforms, so it's a precious starting point in the isomiRs recognition process which is the heart of the tool used for this thesis: isomiR-SEA.

- **Offset site**: nucleotide 8, not rarely the pairing of this nucleotide is a pre-requisite for miRNA-mRNA interactions.

- **Supplementary site**: from nucleotide 13 to 16, but can often be even larger, from nucleotide 12 to 20 approximately and in this case is called compensatory site.

- **Central site**: from nucleotide 4 to 16 approximately, this set of nucleotides was found to be of considerable importance in case seed pairing is not perfect, compensating for a mismatch, or simply as a supplement to that pairing [30].

Having understood that miRNAs are a relevant component in the repression of targets mRNA, it is not difficult to think about the fact that they could play an

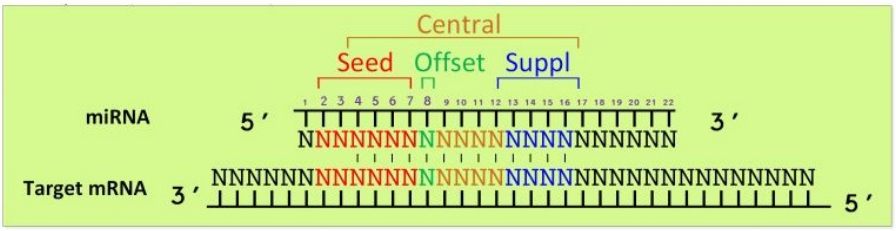


Figure 1.8: **Main steps in miRNA biogenesis** [28].

important role in human physiology and in the development of diseases. In fact, as Li and colleagues say [31], "microRNA are predicted to control the activity of approximately 30% of all protein-coding genes in mammals" and their studies confirm that this regulatory function affects the onset and progression of pathologies like cardiovascular disease, immune disorders or even cancer where they can potentially act as biomarkers [32] because of their involvement in various carcinogenetic processes (as oncogenes or tumor suppressors).

1.2.4 File formats

This section proposes and explains some of the most commonly used file formats in bioinformatics, with the aim of clarifying the different possible representations and guiding the user so that he can recognize and understand them.

FASTA

FASTA is a text-based file format used to represent in a simple way one or more nucleotides or amino acids sequences of nucleic acids and proteins. For each sequence there are two main fields: the description and the sequence itself. The description is a single-line that begins with the symbol > (greater than) and gives basic information about the sequence, for instance a unique name and number. This line is generally uniformly formatted, dictated by the sequence source database or by the generation software. Following line(s) contain strings of characters representing the molecule nucleotide after nucleotide, using the appropriate encoding alphabet (for DNA or RNA). Each sequence ends when another line starts with a > symbol. See *Figure 1.9a* for more details.

FASTQ

FASTQ format is very similar to FASTA with the addition of information regarding the quality with which the sequences were produced by sequencing machines. In FASTQ files each entry is associated with 4 lines:

- Line 1 begins with a @ character and like the FASTA header is a sequence identifier/description.
- Line 2 is the sequence itself.
- Line 3 is the + character optionally followed by additional description.
- Line 4 encodes the quality value for each base of the sequence so it must contain the same number of symbols as line 2. These symbols are ASCII characters from ! which represents the lowest quality up to ~. See *Figure 1.9b* for more details.

```
>Tca-Bantam_3p
UGAGAUCAUUGUGAAAGCUGAUU
>Dno-Mir-340_5p
UUAUAAAGCAAUGAGACUGAUU
>Rno-Mir-504-v1_5p
AGACCCUGGUCUGCACUCUGUCU
>Cfa-Mir-504-v2_5p
GACCCUGGUCUGCACUCUAUC
```

(a) FASTA format

```
@BIOSEQZIP|ID:84|CN:1
AAAAAAACATGGGGCACTTCTTT
+
:114A1DDFHBA3AEFHGGEGHI
@BIOSEQZIP|ID:85|CN:1
AAAAAAACATGGTGCACTTC
+
+114=ADDHHHHDHIIHIII
```

(b) FASTQ format

Figure 1.9: **FASTA and FASTQ format: an example**.

TAG and TAGQ

TAG and TAGQ file formats are customized output files of the tool BioSeqZip [33] which is the software used in this thesis work to perform exact collapsing of NGS datasets of redundant sequences. They are tabular file with a column based structure. In particular TAG files have two columns: a unique sequence called tag and its reads-count, i.e. the number of sequences identical to this one in the non-collapsed input file. TAGQ files have an additional column which represents the tag quality, obtained as the average quality of the collapsed sequences. See *Figure 1.10* for more details.

GFF

GFF stands for General Feature Format and the current stable version is GFF3 which refers to a file format with a tabular structure consisting of a maximum of 9 columns separated by tabs. This format was initially designed to have an easy way to represent genomic features, editable with any type of text editor. Here the detail of each column (see also *Figure 1.11*):

- Col 1: ID of the genomic location of the feature.
- Col 2: a keyword identifying the procedure that generated the feature.
- Col 3: type of the feature.
- Col 4 and 5: start and end genomic coordinates of the feature.
- Col 6: score of the feature.
- Col 7: strand of the feature (+ positive, - negative).
- Col 8: phase of the feature.
- Col 9: list of attributes in tag=value format.

```
AAAAAACCTCCCCCTTTTCGTGG 1
CGGGCCTGGTTAGTAC        3
GGGGGATTAGCTCAAA        7
TCCCCCGGATCTGCCCCTC     1
TCCCCCGGATCTGCCCCTCCAGT 2
TCTCTCGGCTCCTCGCGGCTCT  51
TGTGCAATTCCATGCAAAACTGA 1
TGTGCATTTCTCTCCCTTCTAGA 2
```

(a) TAG file

```
AAAAAAAAAGACACCCCCCACA  @@@DDDDDHFD?DHIIIIIA;B  1
AACGCGGACGTGGAAGAA      CCCFFFFFHHHFFGHJHJ      1
ACTATACAATCTACTCCCTCA   ACCDBECCC@DCA@?0?A<=>   20
CAAAGTTCGGTAGTGCACTGA   JJJJIHHF:HHHHFFFEFCCB   3
CTTTTTGCGATCTGGGCTTGC   AAAEEEEEG;FHHHIIHHIHF   2
GGAGACTCACAAGTTCCTGC    IIIHHHIGGGHHEEDED@AA    14
TACTTGGTTATCTAGCTGTTTGA 4+14ADDBFFFHHFGGIIIIHCA 1
TCTTGTTATCTAGCTGTATG    AA?EDDDDGCADFFGE?DHH    2
```

(b) TAGQ file

Figure 1.10: **TAG and TAGQ format: an example**.

```
1    .    pre_miRNA 1104842   1104894   .    +    .    ID=Cin-Mir-7_pre;Alias=MI0007155
1    .    miRNA     1104842   1104865   .    +    .    ID=Cin-Mir-7_5p;Alias=MIMAT0006091
1    .    miRNA     1104873   1104894   .    +    .    ID=Cin-Mir-7_3p*;Alias=MIMAT0015251
9    .    pre_miRNA 23617075  23617135  .    -    .    ID=Tgu-Mir-551-P2_pre
9    .    miRNA     23617075  23617097  .    -    .    ID=Tgu-Mir-551-P2_5p*
9    .    miRNA     23617115  23617135  .    -    .    ID=Tgu-Mir-551-P2_3p
chr1 .    pre_miRNA 178677763 178677822 .    +    .    ID=Hsa-Mir-4424_pre;Alias=MI0016763
chr1 .    miRNA     178677763 178677784 .    +    .    ID=Hsa-Mir-4424_5p*;Alias=MIMAT0018939
chr1 .    miRNA     178677801 178677822 .    +    .    ID=Hsa-Mir-4424_3p
chr1 .    pre_miRNA 183212430 183212493 .    +    .    ID=Ocu-Mir-670_pre;Alias=MI0039459
chr1 .    miRNA     183212430 183212453 .    +    .    ID=Ocu-Mir-670_5p*;Alias=MIMAT0048553
chr1 .    miRNA     183212469 183212493 .    +    .    ID=Ocu-Mir-670_3p;Alias=MIMAT0048554
```

Figure 1.11: **GFF format: an example** (a dot indicates lack of information).

SAM/BAM

SAM files are tab delimited text format composed of header lines (starting with a @) and alignment lines, so it's a format used to represent mapping of reads to reference sequence. Each record is an alignment and has 11 mandatory fields (see *Figure 1.12*). BAM format, instead, is the binary version of SAM files, so they contains exactly the same informations. These files can be viewed and analyzed using several software tools such as SAMTools, a command line and open source software.

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,254}`	Query template NAME
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME
4	POS	Int	$[0,2^{31}-1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next read
8	PNEXT	Int	$[0,2^{31}-1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31}+1,2^{31}-1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

Figure 1.12: **SAM format: brief description of each column** [34].

However, there are also many other popular formats such as Comma-Separated Values (CSV), Browser Extensible Data (BED), eXtensible Markup Language (XML) or simpler SQL databases.

1.3 Pipeline

Detecting and quantifying miRNAs and isomiRs from sequencing data is therefore considered the central step in the characterization of a biological sample, and for this reason the heart of all the study done for this thesis is a tool called isomiR-SEA. The name stands for isomiRNA Seed Extension Aligner and it is an alignment tool for the analysis and quantification of small non-coding RNA and its isoforms developed by Urgese in 2016 [28]. Here I propose a complete reimplementation of this tool, designed in such a way as to conform to the modern C++17 programming language and the latest versions of open source libraries for sequence analysis (see *Section 1.4*). Furthermore, in the remodelling process I introduced many important changes and added support to some features essential to reduce computational costs and to be able to provide a more streamlined and performing software (see *Sections 2.3 and 2.4*). Afterwards I have completely revised the post-analysis phase of the data produced by this tool, through which it is possible to obtain exhaustive graphical reports (see *Section 2.6*). However this process required the deepening

of different components of a much larger pipeline (see *Figure 1.13*), within which isomiR-SEA and post-analysis are the final two steps.

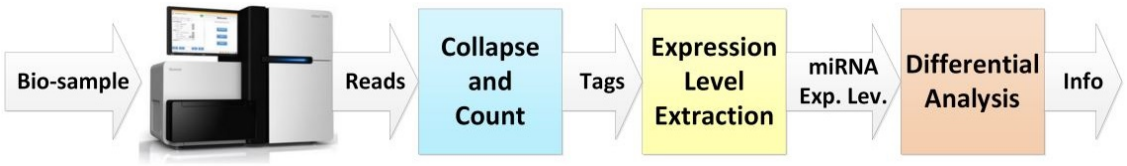


Figure 1.13: **Analysis work-flow of miRNA expression level extraction**. Main steps: biological material extraction and sequencing, collapsing of the reads, alignment, miRNA expression level extraction, output normalization and differential analysis [28].

First of all, all the material extracted from the cell and purified is treated with the electrophoresis protocol, a laboratory procedure that involves the use of a fluid, inside which the molecules of interest are scattered, and an underlying electric charge: applying this charge, the sequences move within the gel based on their size allowing their separation. Since miRNAs are of limited and roughly constant length, sequences with a size that falls within this range are identified with markers (fluorescent material) making their isolation possible. Adapters, the "glues" we have talked about that act as anchor for the sequencing chip, are then attached to these sequences of interest, so we can proceed with the actual Small RNA-seq sequencing process [4].

At this point there is the filtering phase: the text files (ex: FASTQ) produced by the sequencing undergo a quality check to verify which reads have sufficient length and quality to be able to be aligned. After this, adapters must be removed, and this can be done using ad hoc tools like Flexbar [35], Cutadapt [36] and so on.

The pre-processing phase of the reads ends with their collapsing. For this purpose Urgese, Parisi and colleagues in 2020 developed BioSeqZip [33] to perform exact collapsing of II-Generation sequencing datasets: each read is associated with the number of times it appears within the sample(s) and its relative average quality.

Last but not least, we find isomiR-SEA which takes these collapsed file as inputs together with well known pre-existing or manually supplied miRNA datasets in order to perform alignments and produce very detailed tabular output files. These files will then be processed with Python3 scripts to extract information necessary to produce graphical reports showing the spectrum of isoforms in the form of a stacked bar chart (see *Figure 1.14, 1.15, 1.16, 1.17*) or even useful for any machine learning analysis. More specifically, in *Figure 1.14 and 1.15* we can observe, respectively, the percentages of miRNAs/isomiRs and the conserved interaction sites detected in a sample, while in *Figure 1.16 and 1.17* you can see the same kind of analysis expressed in counts (total number of reads) instead of percentages.

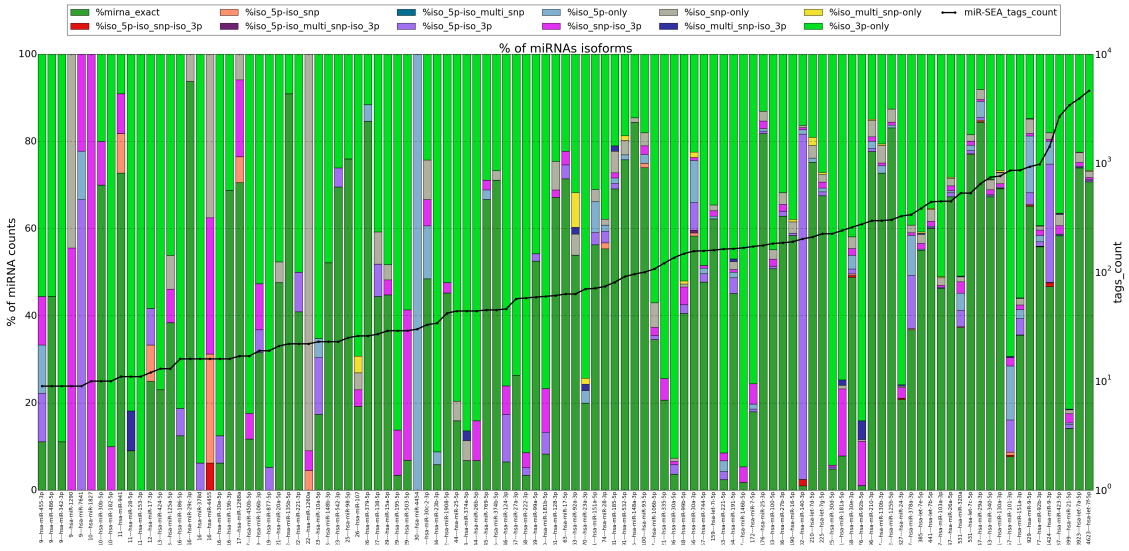


Figure 1.14: **Example of isomiRs percentages detected in a sample**. On the left y-axis are reported the percentages of reads accounting for exact miRNAs or isomiRs while on the x-axis are reported all the analyzed miRNAs in a list. The black line denotes the absolute number of miRNAs mapped reads and its logarithmic scale is on the right y-axis [28].

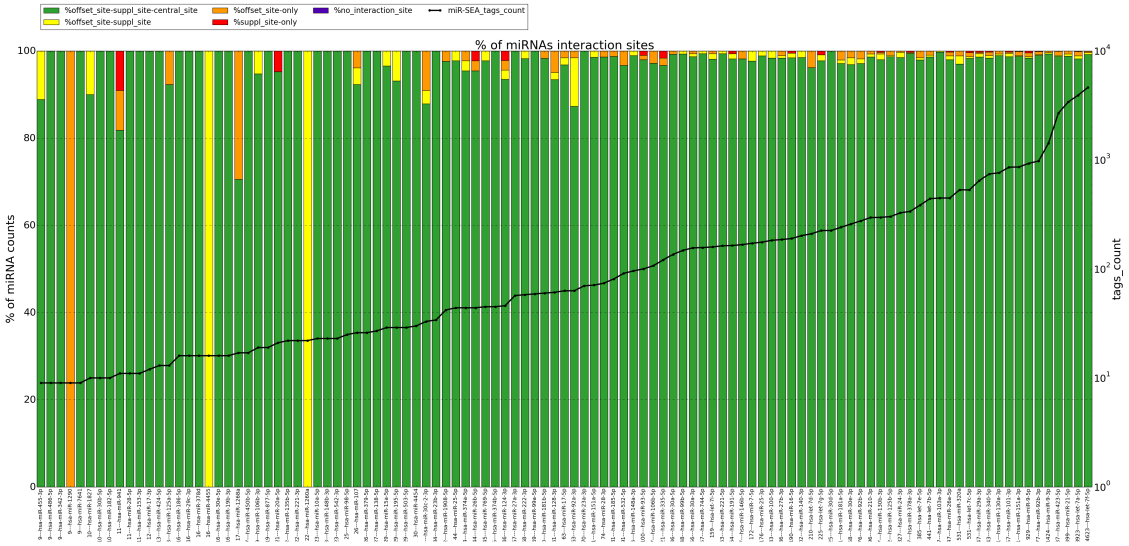


Figure 1.15: **Example of conserved interaction sites percentages detected in a sample**. On the left y-axis are reported the percentages of the conserved interaction sites while on the x-axis are reported all the analyzed miRNAs in a list. The black line denotes the absolute number of miRNAs mapped reads and its logarithmic scale is on the right y-axis [28].

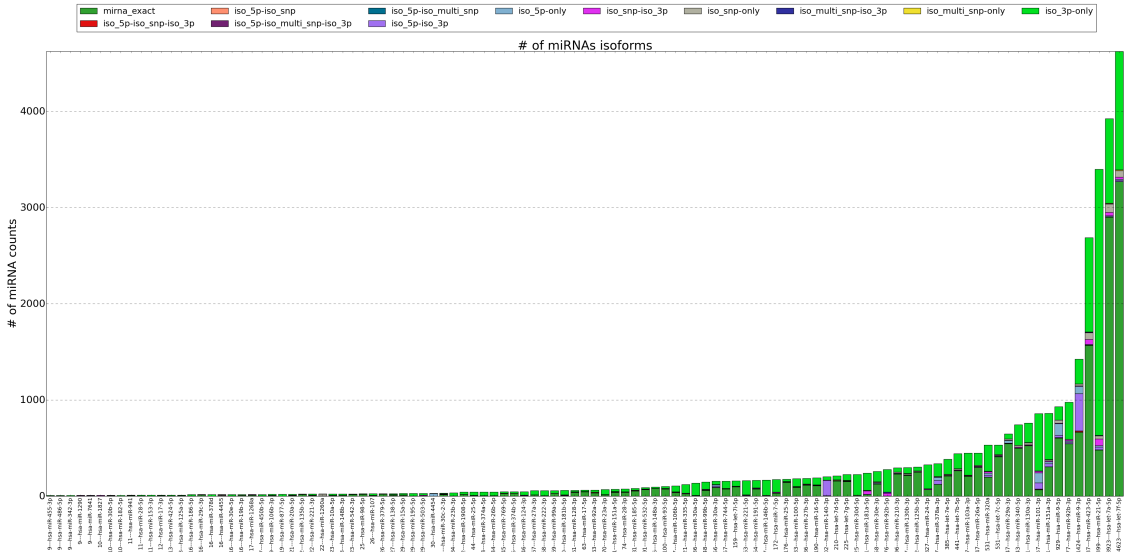


Figure 1.16: **Example of isomiRs count detected in a sample**. On the left y-axis are reported the total number of reads accounting for exact miRNAs or isomiRs while on the x-axis are reported all the analyzed miRNAs in a list.

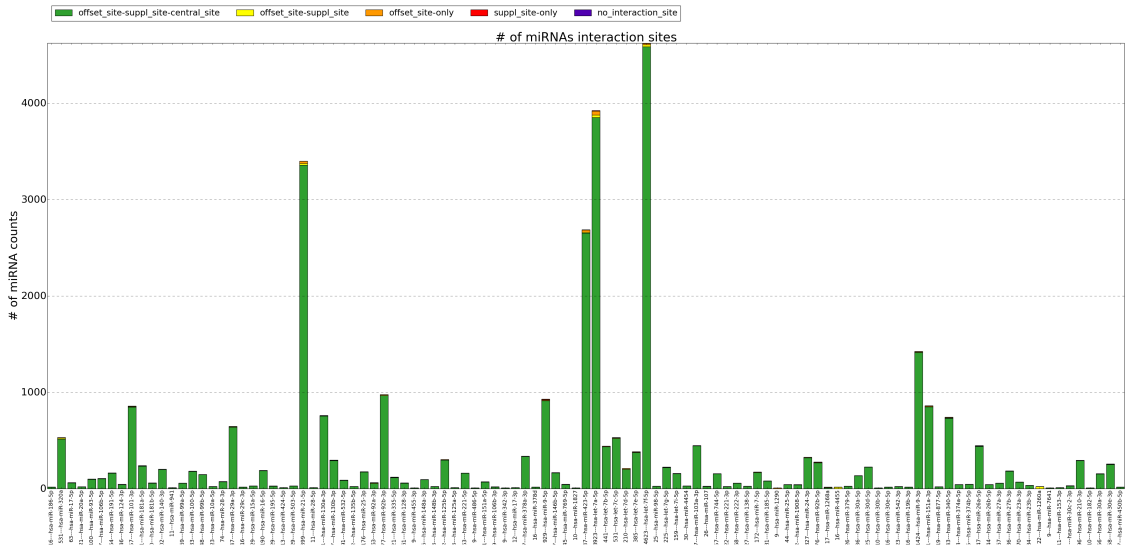


Figure 1.17: . On the left y-axis are reported the total number of the conserved interaction sites while on the x-axis are reported all the analyzed miRNAs in a list.

1.4 Libraries and standards

The design and implementation of proper algorithms are the key activities that can be carried out by a bioinformatician to obtain insights from the data. These algorithms can be developed by writing scripts in common scripting languages,

however, programming languages such as C and C++ are usually preferred because they allow optimized implementations and parallel executions, which are essential when analyzing large amounts of data with complex techniques.

Therefore, in the reimplementation of isomiR-SEA tool I decided to adopt the C++17 standard [37] both to provide a software that complies with a modern and stable release of the standard and to be ready to more easily accommodate the next improvements and changes that will be made with C++20 and from which, however, some important features have already been introduced within this work. In particular C++17 introduced many changes, here are some of the most important:

- Class Template Argument Deduction (CTAD): constructor deduction guides to avoid specifying all template arguments, which are now deduced by the compiler.
- Inline variables: they eliminate the main obstacle to packaging C++ code as header-only libraries.
- Guaranteed copy elision.
- Lambda functions improvements.
- New library utility features such as std::variant which is a type-safe union that can hold a value of one of its alternative types, std::any which is a type-safe container for single values of any type, and std::optional which manages an optional contained value, i.e. a value that may or may not be present.
- Possibility to run in parallel a bunch of std::algorithms, under request.
- Introduction of other features such as: fold-expressions, structured bindings, initializers for if and switch statements, UTF-8 character literal, std::unique_ptr instead of std::auto_ptr and many more.

However, as far as the field of sequence analysis is concerned, C++ and other commonly used programming languages do not provide default packages or libraries capable of meeting the needs of a bioinformatician. For this reason, several external and open source libraries have been developed over the years to specifically support the analysis of biological data. In this work we refer to SeqAn3 [38], the latest version of the well known SeqAn template library for the analysis of biological sequences. It provides many generic algorithms and data structures for sequence representation and transformation, managing input and output of common file formats, text-indexing, text search and sequence alignment. This is possible because the library provides a series of APIs predisposed to the analysis of biological data that are based on data structures finely designed for this purpose, also taking advantage of some features of the modern C++20 language. In addition to this, the

library is very easy to use thanks to a modular structure which provides logical units separated into modules and submodules, to be included in your project when and if necessary. Here are some of the main supported features:

- **Alphabets and data structures**: SeqAn implements specific and optimised alphabets, which are set of symbols used to represent a biological text, for managing sequences of RNA, DNA, protein, quality strings and gap annotation, together with a series of functions useful to, for instance, retrieve the char representation of a symbol and vice versa. Furthermore these data structure are perfectly compatible with STL containers.

- **Input and Output**: provides a set of APIs to assist the user in reading, writing and managing a multitude of file formats usually used in this field.

- **Algorithms**: the library owns a core implementation based on Dynamic Programming which can be extended with a variety of possible configurations in order to compute many desired alignment variants, even choosing the scoring and/or gap scheme.

- **Concepts**: this is a feature of the upcoming C++20 that SeqAn decided to port into its library. Concepts can be seen as an extension to templates and they are useful in all those situations where you want to use generic constructs or algorithms that can work with any type of data as long as they offer a minimal interface (e.g. a T objects that must be equality comparable). In particular you can now *constrain* templates which means that you can make requirements of a template argument explicit (see *Figure 1.18*).

- **Ranges**: another C++20 feature introduced by SeqAn. Ranges were introduced to make C++ code more expressive, avoiding to explicitly use iterators when handling STL containers and collections, which often confuse the programmer leading him to make many mistakes. They are at a level of abstraction above iterators in the sense that they are implemented in terms of iterators but you don't have to worry about them. Moreover, specific kind of range called **Views** are extremely useful when you want to transform ranges via some algorithm or operations. Views in fact are *lazy evaluated* which means that whatever transformation they apply, they do so at the moment you request an element, not when the view is created, allowing you to combine multiple views in a chain (see *Figure 1.19*).

```cpp
// Here we define the concept EqualityComparable
template<typename T>
concept EqualityComparable = requires(T a, T b) {
    { a == b } -> std::boolean;
    { a != b } -> std::boolean;
};

// Here we constrain a function template on the concept just declared
void f(const EqualityComparable auto&);
```

Figure 1.18: **Concepts: example of usage**. Here we have the definition of a new concept named *EqualityComparable* which is satisfied by a and b lvalues of generic type T. In particular the results of $a==b$ and $a!=b$ must be convertible to the *boolean* type. Finally we declare a function constrained on the concept just described.

```cpp
// Declare a vector
std::vector<int> numbers = { 1, 2, 3, 4, 5 };

// Apply range transformation
auto newNumbers = numbers | ranges::view::filter([](int n){ return n % 2 == 0; })
                          | ranges::view::transform([](int n) { return n * 2; });
// Output: 4 8
```

Figure 1.19: **Ranges and views: example of usage**. In the upper part we declare a range (in particular a *std::vector*) while in the lower part we apply some range transformations on it using views: the range is first filtered by selecting only even numbers and then is transformed multiplying all its elements by two.

Chapter 2

Methods

2.1 isomiR-SEA algorithm and flowchart

isomiR-SEA algorithm is divided in three main phases as shown in *Figure 2.1*: the preprocessing step, the alignment procedure and the generation of output files.

Preprocessing step (*Figure 2.1.A*) In this phase parameters are set in order to select the desired isomiR-SEA configuration and the input files are loaded into specific data structures in memory. These parameters and files are processed in order to extract the miRNA sequences and their seeds (as explained in *Section 2.3*), which are the algorithm starting point. The complete list of parameters and the various possible input file combinations will be discussed in *Section 2.5*.

Alignment procedure (*Figure 2.1.B*) The first step of the algorithm involves searching for each miRNA seed within the different tags. Every time a seed is found in a tag, the actual alignment begins: a so-called "ungapped extension" is performed between the tag concerned and each miRNA that owns the seed. This kind of extension is called so because it extends the alignment in both 5' and 3' directions until a *gap* or *mismatch* is found. The ungapped extension is then performed a second time allowing for the presence of a second mismatch.

At this point a first check is made to verify that the size of the alignment does not exceed a certain default or user-selected threshold, so as to discard uninteresting alignments. Then the extension procedure is repeated, but this time only in the 3' direction and for an arbitrary number of times chosen accordingly to the number of mismatches that the user has chosen to allow within a single alignment.

Once here, we compute the alignment score and we evaluate, in the mapped tag, isomiRs and interaction sites. If the produced alignment has a score higher than the pre-set threshold, we try to extend it also to the miRNA

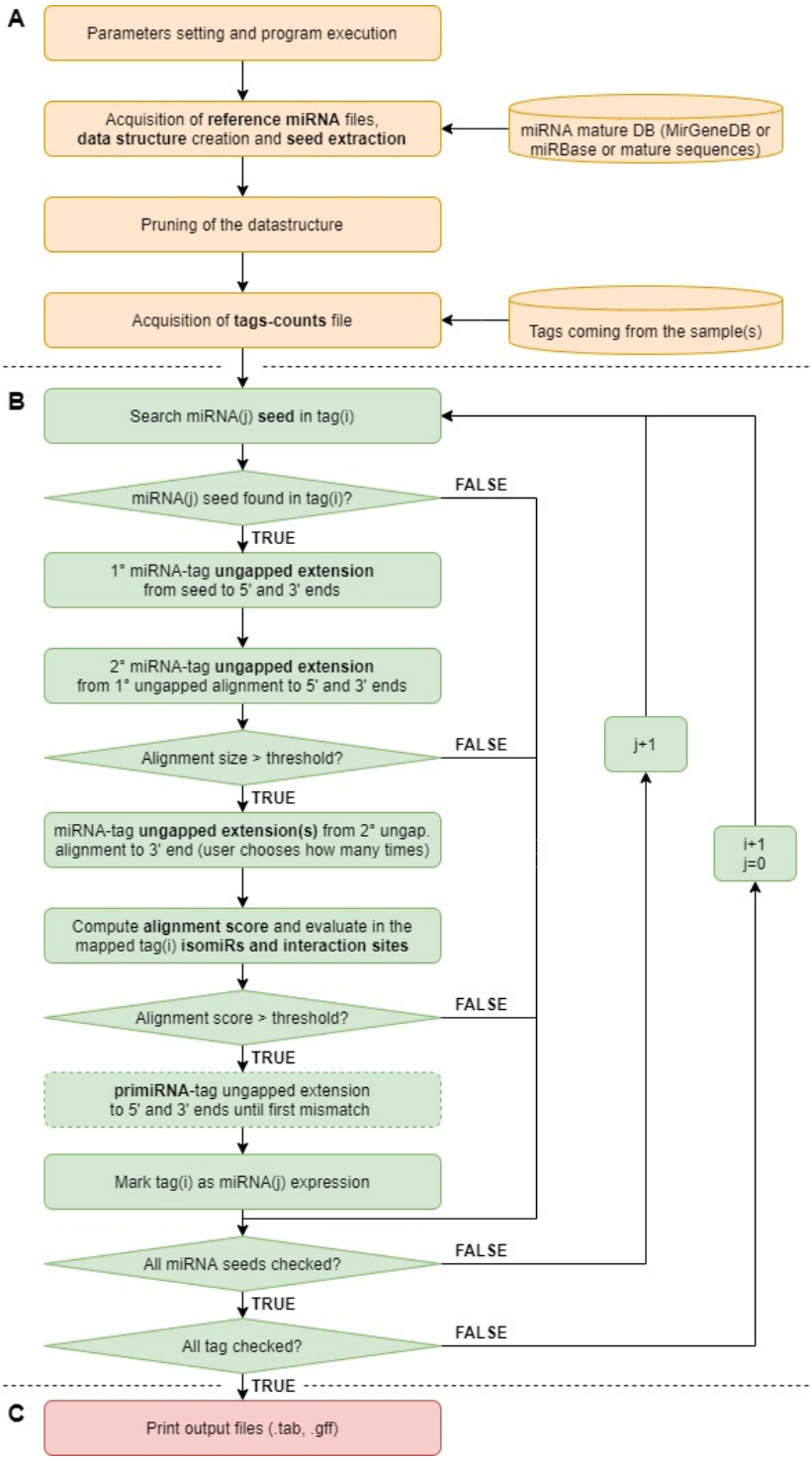


Figure 2.1: **isomiR-SEA algorithm flowchart**. Block A reports on parameters setting and input files preprocessing. Block B describes the alignment procedure together with isomiRs classification. Block C shows the output files generation.

precursor (if present) by counting the number of matches up to the first mismatch. Now the tag is marked as a miRNA expression and all the information about its alignment is recorded within the data structure, ready to be printed.

Output file generation (*Figure 2.1.C*) Finally, isomiR-SEA generates two main output files (.tab, .gff) and a log file (.log) which reports execution statistics and details. The first two files contains, line by line, all the tags-miRNAs alignments and, for each of them, many details about the kind of alignment. These rows includes information such as the alignment score, the position and the number of mismatches, the tags counts of the detected isomiRs, their interaction sites and so on. For a complete description of file formats and row fields, see *Section 2.5*.

2.2 Porting from SeqAn2 to SeqAn3

isomiR-SEA algorithm was developed few years ago in a non-optimized way and not completely tested so, in order to make it usable by the bioinformatics community, there was a strong need for software re-engineerization and bug-correction. Therefore, in remodelling I decided to conform to the modern C++17 programming standard and to SeqAn3, the today's latest version of the library for the analysis of biological sequences. This open source library replaces SeqAn2 which was used in the old version of isomiR-SEA. However, with respect to the previous one, SeqAn3 is a completely new library and unfortunately there isn't any automated way of porting SeqAn2 applications to the current release, so I re-wrote the software by manually making the necessary changes.

During the porting process I followed the snake_case writing standard instead of CamelCase for naming all entities, in order to be very close to the standard library. Then the first significant renewal concerns the use of the new alphabets, types and related functions introduced with SeqAn3, allowing to use alphabets symbols inside STL containers. Here are some detailed examples to better understand:

- CharString type (SeqAn2) has been replaced with std::string.
- IupacString type (SeqAn2) has been replaced with std::vector<seqan3::rna15>.
- StringSet type (SeqAn2) has been replaced with std::vector<std::string>>.
- SeqAn2 predicates (e.g. seqan::EqualsChar<' '>()) have been replaced with SeqAn3 predicates (e.g. seqan3::is_space) or Boost predicates (e.g. boost::is_any_of(" ")).
- SeqAn2 functions such as toLower(), appendValue(), lenght() and many more relied heavily on the data types just cited and defined by the old library

itself, so now they have been replaced with STL public member functions such as std::tolower, std::string::append, std::string::length and so on.

- Ranges have sometimes been adopted to facilitate transformations, such as: std::vector<seqan3::rna15> seq = string_of_nucl | view::char_to<rna15> | ranges::to<std::vector>.

- Other shrewdness such as boost::split instead of strSplit() (SeqAn2), std::variant instead of boost::variant and many many more.

Syntax aside, some modules have been completely revised and certainly the argument parser is one of those that has undergone major changes. In fact the seqan3::argument_parser class now provides a completely renewed interface that mainly allows to: parse command line arguments, define a list or a range of available values for each argument, provide nicely formatted help screens when your call the program with --help. Basically you have to create an object of this class and then you can for instance:

- set metadata containing information: parser.info.author = "Marco";;

- add options or flags to directly store the corresponding parsed value from the command line: parser.add_option(variable, 'n', "number", "This is the description."); which can be used in the command line like this: --number 1;

- setting options as required or hidden: parser.add_option(required_variable, 'n', "name", "This is the description.", seqan3::option_spec::REQUIRED);;

After adding all desired information to the parser object, the command line arguments parsing is triggered by calling the seqan3::argument_parser::parse member function.

In addition to this, major changes have been made regarding the management of input and output: SeqAn3 in fact models files as ranges over records, which certainly allows to handle files more easily, enabling conversions and filtering with very simple one-line commands. The whole mechanism is based on the construction of objects that manage read and write access to different file formats. For example if you want to read from a file in FASTA format you have to create an object seqan3::sequence_file_input fin{std::cin, format_fasta{}}; and cycle on fin as if it is a normal range, so with a for loop. If, on the other hand, you want to write a FASTQ file format, you must create an object seqan3::sequence_file_output fout{std::cout, seqan3::format_fastq{}}; pushing each value at the end of the queue just as if you are working with a std::vector. SeqAn3 supports several file formats currently used in the biomedical field (FASTA, FASTQ, SAM, ...) however, being a library still in development, it lacks some, such as the GFF one. But this kind of file is used as input of our isomiR-SEA tool and for this reason I chose to manually implement

the reading, writing and management of this format. This includes creating ad hoc functions to parse the file as if it is a simple text file, interpreting and storing in memory each single field, which has a specific meaning according to the position in which it is located (see *Section 2.5*).

The only thing that has not been translated is the algorithm used by isomiR-SEA to search for seeds within the tags. This procedure is an online patter matching algorithm based on Myers algorithm [39], but unfortunately in SeqAn3 the online pattern matching is completely replaced by the indexed pattern matching. At first, I tried to translate this algorithm using the syntax, types and constructs of the new SeqAn3 library, however this process was too time-consuming since the algorithm was very rooted in the library, and translating it would not have made any improvement in terms of performance. So, also due to the lack of concrete alternatives within the C++ library, I decided to directly include the SeqAn2 header containing the APIs for this algorithm.

2.3 A rivisited datastructure

The previous version of the tool had a rather complex and certainly not optimized internal data structure. Because of this it was necessary to acquire a large number of input files, often repeating the same contained information several times. Furthermore, the access to variables and structures was often too articulate, making the code less intuitive and less efficient due to redundancies and cyclical dependencies.

In the development of this new release I chose first of all to keep in separate header modules (.h) each set of structures that shares a well-defined logical link, for example: the structures, types and variables concerning microRNAs (miRNAs) are located in a different module with respect to those concerning precursors (pri-miRNA). While previously there was a single file containing all the constructs.

The most notable change, however, concerns the organization of the various data structures in the internal memory. The old data structure had a shape like the one in Figure 2.2 above where each unique miRNA sequence, belonging to a given organism, has a list of structures containing information about each individual miRNA 5p/3p sharing that particular nucleotide sequence. Starting from these structures there was then a further level of information, that is, a list of precursors for each miRNA. These precursor structures contain both pri-miRNA and pre-miRNA sequences. However, all this information is redundant as, for instance, a pri-miRNA sequence already contains within it the pre-miRNA sequence, so we could save only the start and end coordinates (offset) instead of the sequence itself (see Figure 2.3 for further details). In this way I reduced the size of the data structure which is now intuitive and less verbose (see Figure 2.2 below). Moreover, by saving only the pri-miRNA sequence instead of both pri-miRNA and pre-miRNA, you avoid occupying tens of bytes for each input string of nucleotides (which in case of

databases such as MirGeneDB or miRBase are in the order of tens of thousands).

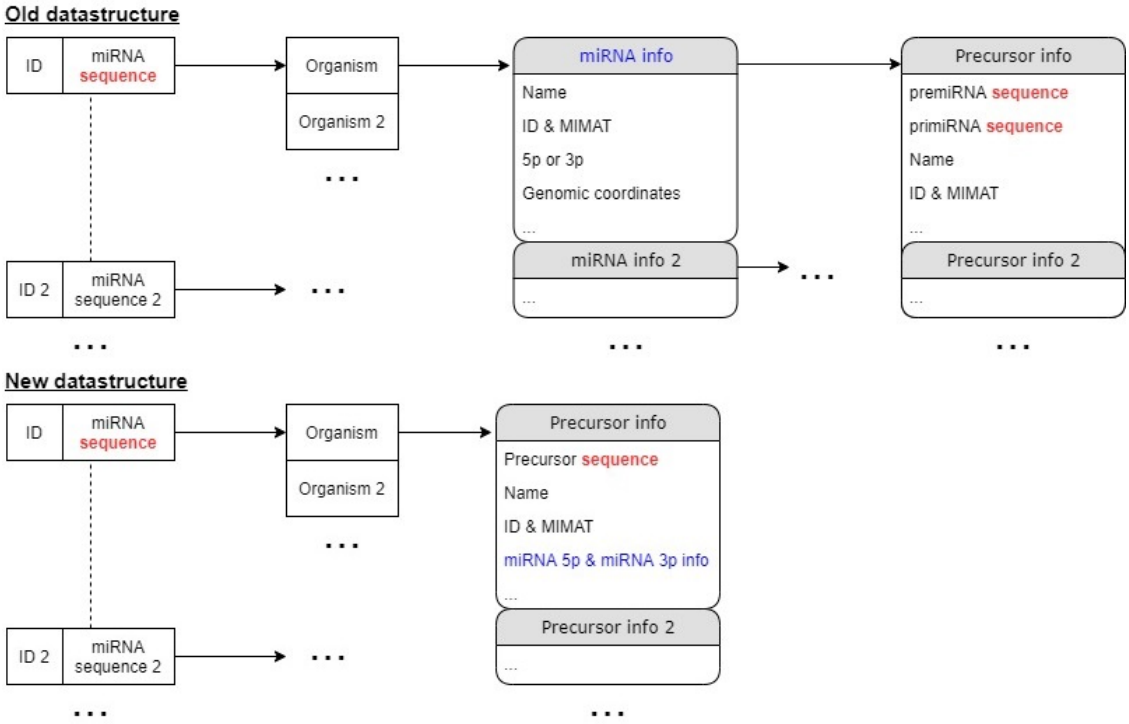


Figure 2.2: **Comparison between old and new data structure**. The old structure is composed of several layers and sequences (in red). The new structure has less layers and sequences, note also where is now located the miRNA structure (in blue).

A further advantage of this new structure is that you can have less input files. In particular only a file of precursor (pri-miRNAs) and one of genetic coordinates. In fact, the latter (GFF) contains the coordinates of both precursors and its miRNAs 5p/3p. These coordinates can be used to extract the miRNA sequences directly from their precursor, as a simple substring (see Figure 2.3). In the previous version of the tool, instead, it was required to provide, in order to have the same level of information: a file of pri-miRNAs, a file of pre-miRNAs, a GFF file containing the genomic coordinates (eventually followed by another file of coordinates in BED format), a file of mature sequences and eventually a file of star sequences.

In case you didn't have precursors, how could this data structure work? For this purpose I have introduced the possibility, when having only mature sequences, to create *fictitious precursors*. These kind of precursors are constructed starting from the miRNA 5p, 3p or both, and adding a constant number of irrelevant nucleotides (N characters) to the left of miRNA 5p, to the right of miRNA 3p and in the middle between the two (see Figure 2.3).

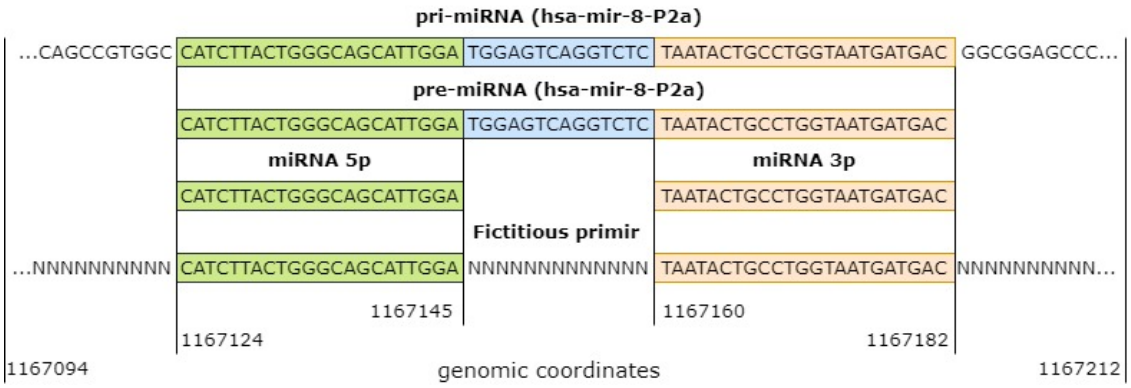


Figure 2.3: **Hairpin precursor (pri-miRNA) and its mature microRNAs (miRNAs)**. Each pri-miRNA contains within itself the pre-miRNA sequence (green+blue+orange), which in turn contains both miRNA 5p (green) and miRNA 3p (orange). If you do not have precursors, a fictitious one can be built starting from the miRNA sequences.

Bug fixing

miRNA without precursor With the mechanism just described it was possible to deal with cases in which there are miRNA sequences not corresponding to any precursor, which previously produced incorrect or missing data. With the trick of the fictitious precursors, in fact, if a miRNA is found without any precursor, it is assigned a fictitious one as seen, with a name that originates from the root of the miRNA name: Hsa−let−7_5p -> Hsa−let−7_pri.

Incorrect coordinates Cases have been found in which some miRNA coordinates were incorrect, i.e. they did not identify the sequence within their precursor. For this reason, if MirGeneDB database is used and the miRNA sequence has a precursor, the tool looks for the position of the sequence inside its precursor, checking if the coordinates of the GFF file are consistent. If they are not, they are updated with those that have been discovered.

Non-unique names In the previous version of the tool each sequence was uniquely identified within the data structure using its name (e.g. Hsa−let−7_5p). This information is certainly unique within certain databases such as MirGeneDB, however it does not appear to be so within miRBase, where different sequences (for example identical sequences but located in different points of the genome) sometimes share the same name. This situation caused data overwriting, therefore producing incorrect results. To remedy this, if miRBase is used the sequences are identified not by their name but by their MIMAT, which is a unique code introduced by miRBase itself.

Unknown microRNAs In miRBase it is possible to find mature sequences that are not identified neither as 5p nor as 3p. To resolve this lack of information I have decided, every time this situation arises, to search within the relative precursor for such sequences in order to understand in which half they are. In fact, depending on its position within the precursor, a sequence can easily be traced back to a 5p or a 3p miRNA (see Figure 2.4).

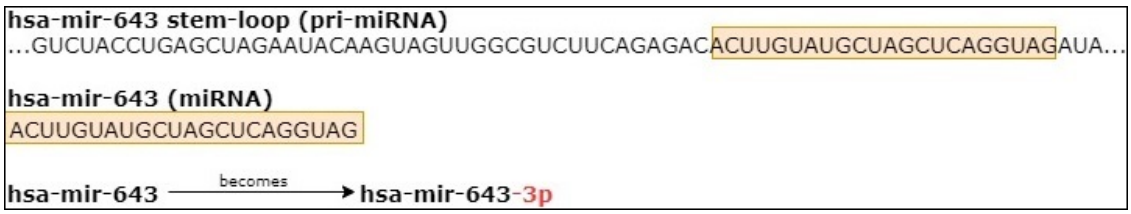


Figure 2.4: **Unknown miRNA detection within its precursor**. *hsa-mir-643* is not identified neither as 5p nor as 3p within miRBase. This lack of information can be solved by looking for *hsa-mir-643* within its precursor: as you can see the sequence is in the right part of the precursor so it can be identified as a miRNA 3p, and its name can be updated in *hsa-mir-646-3p*.

2.4 New features and improvement

2.4.1 Input serialization

Since isomiR-SEA is used to detect and quantify miRNAs and isomiRs within a sample, that is the tags-counts file containing a set of sequences of genomic material coming from an organism, it is necessary to repeat the execution of the tool for each new sample that the user wants to analyse. If this user has to analyse many samples, the number of tool executions would increase accordingly. However, between one run and the next one, the tool uses the same reference database, unless the user decides to do differentiated analysis using a different miRNA database each time. This means that it is necessary each time to provide, read, process and store in memory exactly the same input files (e.g. precursor files and gene coordinate files from the MirGeneDB database), producing an always identical runtime data structure.

Precisely for this reason, and since this repeated operation wastes time, I decided to implement the possibility of serializing the data structure, saving a binary version of it on disk. In this way it can be directly reloaded into memory during all subsequent executions without having to specify and process the same files again. So you can run isomiR-SEA specifying a path where to save the internal data structure filled with your input data (option: −−store−serialized chosen_path) and

then, the next times you run the tool, you can directly load this serialized database (option: −−load−serialized chosen_path), saving execution time.

Besides, this arrangement is particularly efficient in those cases in which, during the input preprocessing phase, expensive operations are performed such as checking the correctness of the coordinates of each single sequence as discussed is Section 2.3, introduced to fix errors inevitably present in the reference databases available online. In this way, in fact, the correction can be made only once during the first execution of the tool, after which the correct data structure is serialized and can be reused by the program in all the subsequent executions.

To serialize the entire data structure I used *cereal* [40], which is an open source C++ serialization library. It is very easy to use considering that is an header-only library, which means that you simply have to include header files (depending on the type of data you want to serialize) and write very basic serialization functions (e.g. defined in the type/struct to be serialized). Moreover cereal provides serialization support for almost all types in the standard library, it is usually faster than many others serialization libraries such as Boost and it is compact, in the sense that produces binary representations that take up less space.

In our case the data structure is serialized in its entirety, without any previous filtering. In this way if a user is interested in an alignment performed only on the sequences of a certain species (e.g. Homo sapiens, code: hsa) and in the subsequent execution to those of a different one (e.g. Mus Musculus (house mouse), code: mmu), he could reuse the same serialized binary file. Anyway, to avoid making the program work with miRNA sequences not relevant to the user, I have introduced a pruning mechanism performed only after acquiring the serialized file, so as to eliminate from the data structure all the sequences and seeds belonging to organism that the user is not interested in (i.e. which he has not specified as a command line parameter), saving a lot of time in the search and alignment phase.

2.4.2 Multi-sample analysis

isomiR-SEA must receive as a command line parameter the path to a tag file. This file certainly represents the most interesting input data because it contains the nucleotide sequences to be analyzed, followed by an integer that identifies the number of times that this string is found within its sample. In these cases the file is in the FASTA or TAG format, while if in addition to the above information we also have the string representing the quality of the sequencing process, then the format can be FASTQ or TAGQ.

In the simplest configurations, isomiR-SEA is ran once for each single tag file, generating a different output for each sample. So, in case you want to analyze a large number of samples you should run the tool as many times. However, it is important to note that the same RNA sequence could be in more than one sample (with an equal or different number of occurrences), especially in the case in which

the samples come from the same organism (for instance at different periods of its existence) or from similar organisms. Therefore, the information contained in the various samples is not univocal but, on the contrary, they are often repeated and this means that, during the various executions of the tool, reads already treated are analyzed again, producing identical alignments every time.

A possible solution to this problem would be to collapse all the repeated reads across the various samples generating a single tags-counts file containing only unique reads. In this way, we no longer have a file for each sample and we greatly reduce the number of sequences to be analyzed. The further advantage of doing so is that isomiR-SEA can be executed only once, not mapping each unique sequence to the same reference miRNA multiple times, saving both memory and execution time (see *Section 3*).

In order to perform such collapsing I have used BioSeqZip [33], an exact collapser for Second Generation Sequencing datasets which, among other things, allows you to collapse into a single file recurrent reads from different samples, associating with each read the number of times it appears within the samples and its relative average quality. The format of this collapsed file can be specified by the user by choosing from some of the most used formats in this field. However, performing the collapse, we would lose the information about the position of the various reads within their sample. To overcome this, when performing multi-samples collapsing a further tabular file is provided by BioSeqZip reporting the detail of how many time each read was found in which sample.

So in conclusion, to make isomiR-SEA aware of all this, I made two important changes:

- First I modified the acquisition of the tags-counts file in input so that I could correctly read the collapsed files produced by BioSeqZip (see *Figure 1.9b*). In particular, the headers of each sequence must be parsed so that useful information such as ID and number of reads can be extrapolated;

- The second change is the most important and allows the tool to generate, with a single execution, several output files, each of which contains the alignments related to a specific sample. In order to do this it is necessary to provide isomiR-SEA, in addition to the collapsed file of reads, also the tabular file (option: $--$multi$-$sample$-$tab path). In this way, during the alignments printing phase, every time you have to print a mapping between a tag and a reference miRNA you read this tabular file up to the line corresponding to that particular tag. In this line you will find the indication of the samples in which this tag appeared (and their relative number of occurrences), so that the tool can print that particular alignment only in the output files related to those samples.

2.4.3 On-the-fly output generation

During the alignment phase, isomiR-SEA searches for and finds all the reads that map to one or more specific reference miRNAs, so as to be able each time to find out whether this read corresponds to a known miRNA or to an isoform thereof. In all these cases in which matching occurs, all the various information regarding each single produced alignment is stored in memory into an optimized data structure. In particular, for each read, called tag, a list of structures is saved where each structure contains the miRNA sequence that has been mapped on that tag and all the information regarding their alignment, such as: score, start position (offset) inside the miRNA and tag, size, boolean indicating whether or not there is conservation of the interaction sites, text strings that summarize the alignment and many more. This set of information is cleverly organized in memory trying to keep references (pointers) to the most full-bodied data such as the sequences themselves or the structures containing information about each single miRNA or pri-miRNA, so as not to occupy more memory than necessary.

Subsequently, however, while printing the results, these references must be resolved. This means that each of the data structures previously described and containing the data of a single alignment must be "expanded", saving the several information in the form of strings, characters and numbers in temporary variables in order to be more easily converted into text and correctly printed within the textual output files.

Anyhow, in the previous version of the tool, these expanded data structures were not directly printed in output, but were temporarily queued into an array of structures. This array was first filled with all the alignments produced by the program and only then scanned to print each alignment one by one. As you can easily imagine, this solution is highly inefficient, since each structure contains up to 50 fields to be converted into text, thus occupying a lot of RAM during execution.

Probably this problem went unnoticed as working with relatively small read files the RAM was used in large quantities but without altering the operation of the program. Instead, in my tests I used much larger read files, also thanks to the new multi-sample feature that allows you to collapse multiple read files into one and give it as input to the program, thus realizing that in some cases the process went into starvation and, after a while, it was killed by the operating system.

To remedy this situation, I decided to print each data structure individually as soon as it is expanded (on-the-fly), without the need to temporarily save it within an array. In this way the result is absolutely identical but there is the enormous advantage of being able to discard this data structure as soon as it is printed, freeing up memory and proceeding with the expansion of the subsequent structure. Using this approach, I obviously had to modify the individual printing functions as they previously worked by cycling on a list of structures while now they receive a single structure at a time.

2.4.4 YARA support

When you run the tool providing an input tag file, not all the reads contained within it are considered during alignment. This happens because a tag sequence can be of variable length and depending on this the read can be discarded or not. In particular, during the acquisition of the databases, isomiR-SEA annotates the maximum length found for a miRNA and then uses this value to discard, during the tag acquisition phase, the longer reads, because it means that any alignment on that tag would certainly contain too many indels. At the same time and for similar reasons, all reads that are too short are also discarded, i.e. those that are shorter than the minimum default value or indicated by the user during system configuration. Furthermore, during the alignment phase, there will almost certainly be some reads that are not mapped on any miRNA, these sequences are inserted at the end of the discard list too.

However it is not said that these discarded sequences are not relevant, they could in fact align on other locations of the genome considered less important for this type of analysis, but certainly not to be overlooked. We are talking about sequences that could map to "secondary " positions such as the loop site, that is the part of the precursor that lies between the two miRNA sequences, or even at the ends of the precursor (sometimes called flanks).

For this reason, in the previous version of isomiR-SEA there was a second alignment phase based on a dynamic programming algorithm in which the discarded sequences were aligned directly on the pri-miRNAs. However, this computation was definitely expensive and since often the user was not interested in the results of that particular analysis, it was decided at first to put it under condition, that is to execute it only if explicitly specified by the user during the configuration phase.

Nevertheless, even under conditional execution, this research phase was poorly optimized, but above all it would have been much more efficient to directly use other tools already available online and specifically designated for that. For this reason I completely removed this second phase of alignment and I decided to interface isomiR-SEA directly to Yara [41], an exact tool for aligning DNA and RNA sequences to indexed reference genomes. In fact, Yara needs first to create an index of the reference genome (if it has already been calculated previously, it can be simply reused without wasting time) and then maps the reads directly on that indexed genome, producing sub-optimal end-to-end alignments in SAM/BAM format.

The user can choose to use Yara simply by indicating that he wants to output the list of discarded reads (option −−path−discarded−tags path). At this point, in the reference manual (see *Section 2.5*), there is a python script that receives this file in FASTQ format, generates (if necessary) the genome index and finally configures and runs the Yara tool. For completeness, the discarded read file has a format identical to that produced by BioSeqZip, with the addition of a new field in the header: it is the RS (ReaSon) field which indicates whether the read was discarded

because it was too long/short (RS: LENGTH) or because it has not been aligned on any miRNA (RS: NOTALL). Furthermore, starting from this file of discarded reads, if you want to go back to the sample relating to a tag, it would be enough to look at the ID index, the indices in fact are still those originally established by BioSeqZip.

2.5 Usage & configurations: a reference manual

All the material including source code, test files, manual and post analysis scripts is available online in the GitLab repository [42].

2.5.1 Setup

This manual is intended for operating systems of the GNU / Linux family. However this does not prevent you from using the operating system you prefer.

Software requirements

In order to build and run isomiR-SEA you need a modern C++ compiler with OpenMP extensions, CMake tool and Git. Furthermore this tool makes use of a modern C++ library for sequence analysis named SeqAn. In particular isomiR-SEA uses both the current version of the library (SeqAn3) which is automatically included on the fly during compilation and the previous one (SeqAn2) which instead can be installed manually with a command below (or by following the tutorials on its official website).

gcc >= 7

```
# sudo apt install g++-7
```

cmake >= 3.4

```
# sudo apt-get install cmake
```

git

```
# sudo apt install git
```

SeqAn2

```
# sudo apt install libseqan2-dev
```

Note: this package is available in Ubuntu since 18.04 LTS, if you have an older version or another OS you can alternatively refer to SeqAn2 installation guide [43] or download the latest version [44], extract the content and copy the /include/seqan directory into the project include folder.

ZLIB, BZip2 and Boost libraries (depending on your distribution you may also need to install these libraries):

```
# sudo apt install zlib1g-dev libbz2-dev
# sudo apt-get install libboost-dev
```

Yara (optional)
If you want to align isomiR-SEA discarded reads directly on the miRNA precursors you need to install Yara tool following its Github documentation [45] and then proceed with reading this manual.

Build

Create a build directory using CMake as follows:

```
# mkdir cmake-build-release
# cd cmake-build-release
# cmake ..
```

Note: if you have installed g++(>=7) but your current default compiler is different, you can invoke cmake as follows:

```
# cmake -D CMAKE_CXX_COMPILER=g++-7 ..
```

Invoke make as follows:

```
# make
```

You will find your executable here: cmake-build-release/apps/isomir_sea

2.5.2 Input & Output

Input files

You can provide different kind of input file to isomiR-SEA:

File of precursors (option: –in-file-primir) Precursor files are FASTA files (.fasta, .fas, .fa) available online, you can alternatively download them from *MirGeneDB* download-page (precursor with 30nt-flank), from *miRBase* download-page (hairpin file), or with *wget* command (see *Section 2.5.3*). Please note that *if you provide a precursor file you must also provide a genomic coordinates file.*

File of genomic coordinates (option: –in-file-gff) Genomic coordinates files are GFF files (.gff, .gff3) available online, you can alternatively download the GFF file (v2.0) from *MirGeneDB* download-page, all .gff3 files (v22) from *miRBase* download-page, or with *wget* command (see *Section 2.5.3*). Please

note that *if you provide a genomic coordinates file you must also provide a precursor file.*

File of mature and/or star sequences (options: –in-file-mature, –in-file-star) Mature and/or star sequences files are FASTA files (.fasta, .fas, .fa) available online from *MirGeneDB* or *miRBase* download-pages or with *wget* command (see *Section 2.5.3*), but you can also provide *your own sequences* only paying attention to stick to the format used in the files of the sites just mentioned.

Serialized reference database (options: –store-serialized, –load-serialized) You can run isomiR-SEA specifying a path where to save the internal data structure filled with your input data (option: ––store–serialized) as explained in *Section 2.4.1*. Doing so, the next times you run the tool, you can directly load the serialized database (option: ––load–serialized) instead of re-reading the same input files, saving execution time.

File of tags/reads (options: –in-file-tags, –multi-sample-tab) You can provide your own tags/reads file (FASTA, FASTQ, TAG or TAGQ) even multi-sample (see *Section 2.4.2*). In this last case you will also provide the tabular file (option: ––multi–sample–tab) containing informations about the number of reads per sample of each sequence. As already said, to produce this kind of collapsed file we used BioSeqZip, so we kindly suggest you to use it.

To understand the several possible input files combinations to use see *Section 2.5.3*.

Output files

isomiR-SEA generates three main output files: .log, .tag, .gff. Each of them contains some or all of the following fields (see also *Figure 2.5*):

tag_id (TI): an integer value that represents a unique index given to each tag while reading the input tags-counts file, an incremental number is given for each new row read in that file;

tag_seq (TS): the tag sequence;

tag_qual (TQ): the average quality (for each nucleotide) of the tag sequence;

tag_count (TC): the number of identical reads of each tag;

tag_start_gen (TSG): the start position (coordinate) of the tag into genome;

tag_end_gen (TEG): the end position (coordinate) of the tag into genome;

org_code (ORG): a 3-letters code to identify the miRNA sequence organism;

mir_id (MI): an integer value that represents a unique index given to each miRNA sequence while reading the input reference files, an incremental number is given for each new sequence;

mir_seq (MS): the miRNA sequence which has been aligned to the tag;

align_mark (AM): the alignment score;

align_iupac_tag (IT): the iupac string of the alignment w.r.t. the tag;

align_iupac_mir (IM): the iupac string of the alignment w.r.t. the miRNA;

align_cigar (CI): a compressed representation of the alignment;

align_length (AL): the length of the alignment;

mir_tag_size_diff (SD): the difference in length between the tag and the miRNA;

iso_exact (IEX): a boolean indicating whether the sequence is identical the canonical mature sequence reported in the database;

iso5p (I5P): an integer indicating whether the sequence is an isoform resulting from insertion or deletion in the 5p-end (+: insertion, -: deletion);

iso_m_snp (IMS): stands for *multiple nucleotide polymorphism isoforms* and it's a boolean indicating if the sequence presents more than a mismatch with respect to miRNA sequence;

iso_snp (ISN): stands for *single nucleotide polymorphism isoforms* and it's a boolean indicating if the sequence presents a mismatch w.r.t. the miRNA;

iso3p (I3P): an integer indicating whether the sequence is an isoform resulting from insertion or deletion in the 3p-end (+: insertion, -: deletion);

mismatch_in_seed (INS): a boolean indicating whether there are mismatches in the seed or not;

off_site (IOS): a boolean indicating whether the offset site (nucleotide 8) is conserved in the tag sequence;

suppl_site (ISS): a boolean indicating whether the supplementary site (from nucleotide 13 to 16) is conserved in the tag sequence;

compens_site (IPS): a boolean indicating whether the compensatory site (from nucleotide 12 to 20 approximately) is conserved in the tag sequence;

central_site (ICS): a boolean indicating whether the central site (from nucleotide 4 to 16 approximately) is conserved in the tag sequence;

a2i_in_seed (AIS): *adenine to inosine in seed*, it's a boolean indicating if, inside the seed, an 'A' in the miRNA correspond to a 'G' in the tag;

a2i_out_seed (AIO): *adenine to inosine out seed*, it's a boolean indicating if, outside the seed, an 'A' in the miRNA correspond to a 'G' in the tag;

num_of_mir_same_seq (MI4S): an integer indicating the number of miRNAs the tag TI was aligned with;

score_diff_mir0_mir1 (MSD): an integer indicating the score difference between two subsequent alignment of miRNAs (same organism) with the same tag sequence (ex: between the score of the 3rd best aligned miRNA and the score of the 2nd one);

mir_info_index (MII): an integer that represents a unique index given to each miRNA while reading the input reference files, an incremental number is given for each new sequence (it differs from MI because identical miRNA sequences from different organisms have different MII but same MI);

mir_info_info (MIN): a string indicating the full information of the miRNA;

mir_info_ref (MRF): a string indicating the reference chromosome of the miRNA;

mir_info_start_gen (MSG): start position/coordinate of the miRNA into genome;

mir_info_end_gen (MEG): end position/coordinate of the miRNA into genome;

mir_info_mimat (MIM): miRBase identifier for the miRNA sequence;

mir_info_strand (MSR): the strand of the miRNA sequence;

prx_index (PII): an integer value that represents a unique index given to each precursor sequence while reading the input reference files, an incremental number is given for each new sequence;

prx_info (PIN): a string indicating the full information of the precursor;

prx_ref (PRF): a string indicating the reference chromosome of the precursor;

prx_seq (PS): the precursor sequence from which the miRNA originates;

prx_start_gen (PSG): start position/coordinate of the precursor into genome;

prx_end_gen (PEG): end position/coordinate of the precursor into genome;

prx_mimat (PMI): the MIMAT (miRBase identifier) of the precursor sequence;

prx_strand (PSR): the strand of the precursor sequence;

num_of_prx_same_seq (PR4S): an integer indicating the number of different precursors in which the miRNA sequence appears;

prx_align_mark (PAM): the score of the alignment extended to the precursor (increased with each new match);

iso5p_canonic (IC5): a boolean indicating whether the 5p end of the tag (insertion/s) is aligned on the precursor;

iso3p_canonic (IC3): a boolean indicating whether the 3p end of the tag (insertion/s) is aligned on the precursor;

score_diff_prx0_prx1 (PSD): an integer indicating the score difference between two subsequent alignment of the tag with different precursor sequences;

mir_tag_out_id (MTOID): an integer value that represents a unique index given to each miRNA-tag alignment in output, an incremental number is given for each new row;

mir_matched_id (MMI): an integer value that represents a unique index given to each subsequent miRNA aligned with the same tag sequence, an incremental number is given for each different miRNA;

mir_info_id (MIID): an integer value that represents a unique index given to each miRNA sequence while reading the input reference files, an incremental number is given for each new sequence;

premir_id (PID): an integer value that represents a unique index given to each precursor sequence while reading the input reference files, an incremental number is given for each new sequence;

align_iupac_prx (IP): the iupac string related to the alignment with respect to the precursor;

single_multiple_discarded (SMD): an integer value indicating if the current row is a unique-mapping, multiple-mapping or a mapping to be discarded.

TI	173298						**PII**	7644	
TQ	,9-75783688:2:<4439;5;59						**PIN**	Mmu-Mir-17-P1a_pri	
TC	1						**PRF**	chr14	
TS	AACAGUGCUUACAGUGCAGGUAGU						**PS**	GACUGAAGCUGUGACCAGUCAGAAUAAU GUCAAAGUGCUUACAGUGCAGGUAGU GAUGUGUGCAUCUACUGCAGUGAGGGC ACUUGUAGCAUUAUGCUGACAGCUGCC UCGGUGGGAGCC	
MS	CAAAGUGCUUACAGUGCAGGUAG								
IT	MAMAGUGCUUACAGUGCAGGUAGu								
IM	MAMAGUGCUUACAGUGCAGGUAG								
CI	AMC20MI						**PSG**	115043653	
AM	17						**PEG**	115043773	
AL	23						**PMI**	MI0000687	
TSG	115043682						**PSR**	+	
TEG	115043706						**PR4S**	1	
MI	385						**PAM**	1	
MII	4576						**IC5**	F	
ORG	mmu						**IC3**	T	
MIN	Mmu-Mir-17-P1a_5p						**PSD**	? (tag was not aligned on any other precursors)	
MRF	chr14						**MTOID**	24536	
MSG	115043683						**MMI**	0	
MEG	115043706						**MIID**	385	
MIM	MIMAT0000649						**PID**	0	
MSR	+	**IMS**	T	**IOS**	T	**AIS** F	**IP**	? (tag was not aligned on the precursors)	
SD	-1	**ISN**	F	**ISS**	T	**AIO** F	**SMD**	1	
IEX	F	**I3P**	1	**IPS**	T	**MI4S** 5			
I5P	0	**INS**	T	**ICS**	T	**MSD** 1			

Figure 2.5: **Example of alignment features**. Features of an alignment produced by isomiR-SEA. In particular the tag has been aligned to *Mmu-Mir-17-P1a_5p* miRNA. Each acronym has its value immediately to its right, then: differences between the sequences are highlighted in red, miRNA starting location within the genome is in blue, precursor starting location within the genome is in orange.

.log file
Contains the list of options and parameters used, followed by all the alignment features detected during the alignment of a Tag over miRs, and a small report that shows execution times in the several phases together with some information like the number of Tags used, discarded, aligned and the number of miRs aligned.

.tag file
Each line of this file contains the information about a specific Tag-miR alignment. All these informations are tab-separated and the overall output is ordered by sequence alphabetically (see *Figure 2.6*).

TI	TS	TQ	TC	TSG	TEG	ORG	MI	MS	AM	IT	IM	CI	AL	SD	IEX	I5P	IMS	ISN	I3P	INS	IOS	ISS	IPS	ICS
84	AAAAAAACAUGGGGCACUUCUUU	:114A1DDFHBA3AEFHGGEGHI	1	101033802	101033825	hsa	1542	AAACAAACAUGGUGCACUUCUU	16	AAAI
87	AAAAAAACAUGGUGCACUUCUU	64939=ACDCDE@DEDDFGFGG	20	101033803	101033825	hsa	1542	AAACAAACAUGGUGCACUUCUU	18	AAAI
88	AAAAAAACAUGGUGCACUUCUUA	8875<>CDCFGG=EE@DGHIHIG	1	101033802	101033825	hsa	1542	AAACAAACAUGGUGCACUUCUU	17	AAAI
91	AAAAAAACAUGGUGCACUUCUUU	55849<ABEEEDCEFDFHHGHGH	13	101033802	101033825	hsa	1542	AAACAAACAUGGUGCACUUCUU	17	AAAI
93	AAAAAAACAUGGUGCACUUCUUUU	11+44@DDHHHHHIHIIIIIIIII	1	101033801	101033825	hsa	1542	AAACAAACAUGGUGCACUUCUU	16	AAAI
653	AAAAAGCUAGAUAACCAAUC	A33<FIGFFADABA2441++	1	156420394	156420414	hsa	2792	AUAAAGCUAGAUAACCGAAAGU	10	AWA/
653	AAAAAGCUAGAUAACCAAUC	A33<FIGFFADABA2441++	1	88666904	88666924	hsa	2792	AUAAAGCUAGAUAACCGAAAGU	10	AWA/
653	AAAAAGCUAGAUAACCAAUC	A33<FIGFFADABA2441++	1	89368072	89368092	hsa	2792	AUAAAGCUAGAUAACCGAAAGU	10	AWA/
682	AAAAAGGAAUCCCCAGAGCAGA	C@@DFFFFGHHHFJJJHFHGGG	1	49020631	49020653	hsa	423	CAACGGAAUCCCAAAAGCAGCUG	9	aMA/

Figure 2.6: **isomiR-SEA output: .tag file**.

.gff file
Each line of this file contains, similarly to the .tab file, the information about a specific Tag-miR alignment but in GFF format. All these informations are tab-separated a part from the last column which is the compression of several fields together separated by a semicolon. The overall output is ordered by tag sequence alphabetically, and then by score in descending order (see *Figure 2.7*). In particular:

Column 1: mir_info_ref (MRF);

Column 2: name of miRNA database;

Column 3: type of the record;

Column 4: tag_start_gen (TSG);

Column 5: tag_end_gen (TEG);

Column 6: align_mark (AM);

Column 7: mir_info_strand (MSR);

Column 8: prx_strand (PSR);

Column 9: is composed of several fields: TI, TS, TC, PIN, CI, MIN, ISO (which is a concatenation of IEX, *IC5*, I5P, IMS, ISN, I3P, *IC3*), INT (which is a concatenation of INS, IOS, ISS, IPS, ICS) and FILTER which is a field that can be "Pass" or "NotPass". For instance if there are two or more different miRNAs belonging to the same organism that have matched with the same tag, only one of them will be "Pass" and only if it has a strictly higher score

than the others. On the other hand, if there are two equal miRNAs but belonging to different precursors (and therefore with the same score) and these scores are higher than the other miRNAs then they will both have "Pass".

```
chr14   .   isomiR   101033803 101033825 16   +   .   TI=84;TS=AAAAAAACAUGGGGCACUUCUUU;TC=1;PIN=Hsa-Mir-154-P19_pri;CI
chr14   .   isomiR   101033804 101033825 18   +   .   TI=87;TS=AAAAAAACAUGGUGCACUUCUU;TC=20;PIN=Hsa-Mir-154-P19_pri;CI
chr14   .   isomiR   101033803 101033825 17   +   .   TI=88;TS=AAAAAAACAUGGUGCACUUCUUA;TC=1;PIN=Hsa-Mir-154-P19_pri;CI
chr14   .   isomiR   101033803 101033825 17   +   .   TI=91;TS=AAAAAAACAUGGUGCACUUCUUU;TC=13;PIN=Hsa-Mir-154-P19_pri;C
chr14   .   isomiR   101033802 101033825 16   +   .   TI=93;TS=AAAAAAACAUGGUGCACUUCUUUU;TC=1;PIN=Hsa-Mir-154-P19_pri;C
chr1 .  isomiR   156420395 156420414 10   -   .   TI=653;TS=AAAAAGCUAGAUAACCAAUC;TC=1;PIN=Hsa-Mir-9-P1_pri;CI=MA14MAMUC
chr5 .  isomiR   88666905  88666924  10   -   .   TI=653;TS=AAAAAGCUAGAUAACCAAUC;TC=1;PIN=Hsa-Mir-9-P2_pri;CI=MA14MAMUC
chr15   .   isomiR   89368073  89368092  10   +   .   TI=653;TS=AAAAAGCUAGAUAACCAAUC;TC=1;PIN=Hsa-Mir-9-P3_pri;CI=MA14
chr3 .  isomiR   49020632  49020653  9    -   .   TI=682;TS=AAAAAGGAAUCCCCAGAGCAGA;TC=1;PIN=Hsa-Mir-191_pri;CI=IA2MA8MC
chr19   .   isomiR   53708794  53708817  8    +   .   TI=732;TS=AAAAAGGGCUUCUGUCGUGAGUGG;TC=1;PIN=Hsa-Mir-430-P21_pri;
```

Figure 2.7: **isomiR-SEA output: .gff file**.

2.5.3 Usage and configurations

This section will guide you step-by-step in the use of isomiR-SEA tool and its various configurations. For each one of them there will be indicated the shell-commands to use and, in the last use case, there will be for sake of completeness the ipython script to directly run the configuration with pseudo-realistic tag input files. These files are already present in the io_files_test folder but depending on which configuration you want to execute you have also to download the reference files already mentioned in *Section 2.5.2*. For simplicity, here are the wget-commands (take only what you need).

MirGeneDB

```
1 # mkdir ./io_files_test/mirgenedb
```

Precursor:

```
1 # wget -O ./io_files_test/mirgenedb/ALL--pri-30-30.fas https://
    mirgenedb.org/static/data/ALL/ALL--pri-30-30.fas
```

Mature:

```
1 # wget -O ./io_files_test/mirgenedb/ALL--mature.fas https://
    mirgenedb.org/fasta/ALL?mat=1
```

Star:

```
1 # wget -O ./io_files_test/mirgenedb/ALL--star.fas https://mirgenedb.
    org/fasta/ALL?star=1
```

GFF:

```
# wget -O ./io_files_test/mirgenedb/ALL.gff https://mirgenedb.org/gff/ALL?sort=pos&all=1
```

miRBase

```
# mkdir ./io_files_test/mirbase
```

Precursor:

```
# wget -O - ftp://mirbase.org/pub/mirbase/CURRENT/hairpin.fa.gz | gunzip -c > ./io_files_test/mirbase/hairpin.fa
```

Mature:

```
# wget -O - ftp://mirbase.org/pub/mirbase/CURRENT/mature.fa.gz | gunzip -c > ./io_files_test/mirbase/mature.fa
```

GFF:

```
# mkdir ./io_files_test/mirbase/gff
# wget -r --no-parent -P ./io_files_test/mirbase/gff -nd --reject "index.html*" ftp://mirbase.org/pub/mirbase/CURRENT/genomes/
```

Note: if you want to run these scripts using your personal input data remember to replace the paths with those of your input and/or output files!

Use Case 0: Reference manual

Show the integrated reference manual:

From project directory

```
# ./cmake-build-release/apps/isomir_sea --help
```

From executable directory

```
# ./isomir_sea --help
```

Use Case 1: Precursors and genomic coordinates

This is the most basic configuration: you provide a file of precursors and a file of genomic coordinates. In this way the tool, knowing the coordinates of each precursor and its microRNAs, can extract these sequences from the precursor file. Here are the two examples of this: one using MirGeneDB and one using miRBase. For both of them the tag file is the BioSeqZip-collapsing of trimmed ENCFF005EAG.fastq.

MirGeneDB

```
# ./cmake-build-release/apps/isomir_sea --in-file-primir
mirgenedb_primir_file --in-file-gff mirgenedb_gff_file --in-file-
tags tag_file --specie-codes mmu --mismatches-in-seed 1 --
verbose 1
```

miRBase

```
# ./cmake-build-release/apps/isomir_sea --in-file-primir
mirbase_primir_file --in-file-gff mirbase_gff_file --in-file-tags
tag_file --specie-codes mmu --mismatches-in-seed 1 --verbose 1
```

Use Case 2: Mature and/or Star sequences

You can provide MirGeneDB (mature and/or star) or miRBase (only mature exists) sequences or alternatively you can provide your own sequences, but please pay attention to stick to the format used in the files of the sites just mentioned.

The example shown here uses MirGeneDB mature and star sequences and the tag file is the BioSeqZip-collapsing of trimmed ENCFF005EAG.fastq.

```
# ./cmake-build-release/apps/isomir_sea --in-file-mature mature_file
[--in-file-star star_file] --in-file-tags tag_file --specie-codes
mmu --mismatches-in-seed 1 --verbose 1
```

Use Case 3: MirGeneDB precursors and mature/star sequences

In case you are using MirGeneDB precursors and genomic coordinates as input files for isomiR-SEA (Use Case 1, MirGeneDB section) you can also provide, in addition, a file of mature sequences and/or a file of star sequences (like Use Case 2). Note that in this way, the tool takes each mature/star sequence and checks the precursor file: if there are sequences that do not belong to any precursor, a "fictitious" precursor is created for them (see *Section 2.3*). If, on the other hand, they correspond to a precursor, the tool searches for the position of the sequence within the precursor and checks if the coordinates of the gff file are consistent, if they are not, they are updated with those that have been discovered.

This prevents and corrects any errors in the MirGeneDB database and allows you to add any of your personal sequences to the analysis.

The tag file is the BioSeqZip-collapsing of trimmed ENCFF005EAG.fastq.

```
# ./cmake-build-release/apps/isomir_sea --in-file-primir
mirgenedb_primir_file --in-file-gff mirgenedb_gff_file --in-file-
mature mature_file [--in-file-star star_file] --in-file-tags tag_file
--specie-codes mmu --mismatches-in-seed 1 --verbose 1
```

Use Case 4: Serialization

Specify a path where to save the internal data structure filled with your input data (option: −−store−serialized), doing so, the next times you run the tool, you can directly load the serialized database (option: −−load−serialized) instead of re-reading the same input files, saving execution time.

The example shown here uses miRBase sequences and the tag file is the BioSeqZip-collapsing of trimmed ENCFF005EAG.fastq.

First execution, store serialized

```
# ./cmake−build−release/apps/isomir_sea −−in−file−primir primir_file −−in−file−gff gff_file −−in−file−tags tag_file −−store−serialized serialization_path −−specie−codes mmu −−mismatches−in−seed 1 −−verbose 1
```

Subsequent executions, load serialized

```
# ./cmake−build−release/apps/isomir_sea −−load−serialized serialization_path −−in−file−tags tag_file −−specie−codes mmu −−mismatches−in−seed 1 −−verbose 1
```

Use Case 5: Alignment parameters

You can set many parameters to run the tool according to your preferences. Here are described some of the most important (verbosity and alignment), and then there is an example of execution with them. For more information and to find out other parameters please refer to the output of the −−help command.

The example shown here uses miRBase sequences and the tag file is the BioSeqZip-collapsing of trimmed ENCFF005EAG.fastq.

-V, –verbose 0: no outputs; 1: displays global statistics; 2: displays extensive statistics for each batch of reads; 3: debug output.

-n, –specie-codes Specie codes to be evaluated during the alignment at the same time (usually only one is used).

-o, –min-size-aln-stp1 Minimum size of ungapped alignment, starting from the seed, extending the alignment.

-p, –min-align-score Minimum alignment score for considering a tag expression of a miR.

-q, –seed-start Start position of the seed.

-r, –seed-end End position of the seed.

-s, –max-start-pos-tag Max index in tag position for starting the seed alignment.

-t, –min-size-tag Minimum size of tag to be considered for the alignment.

-u, –thr-select-tags Threshold used to select select high quality multimapped tags.

-v, –mismatches-out-seed Number of mismatches allowed between miRNA and tags.

-z, –mismatches-in-seed Number of mismatches allowed between miRNA seed and tags.

```
# ./cmake-build-release/apps/isomir_sea --in-file-primir primir_file
  --in-file-gff gff_file --in-file-tags tag_file --specie-codes hsa:
  mmu --min-size-aln-stp1 10 --min-align-score 7 --seed-start 1
  --seed-end 6 --max-start-pos-tag 5 --min-size-tag 15 --thr-
  select-tags 12 --mismatches-out-seed 3 --mismatches-in-seed 1
  --verbose 1
```

Use Case 6: Multi-Sample

You can provide multi-sample tags/reads file: to make isomiR-SEA aware of this you have to provide also the tabular file (option: --multi-sample-tab) containing informations about the number of reads per sample of each sequence. To produce this kind of collapsed file we used BioSeqZip, so we kindly suggest you to use it.

The example shown here uses miRBase sequences, while the tag and tabular file are the BioSeqZip outputs (truncated to the first 500000 reads) of trimmed ENCFF005EAG.fastq and ENCFF008VOH.fastq.

```
# ./cmake-build-release/apps/isomir_sea --in-file-primir primir_file
  --in-file-gff gff_file --in-file-tags multi_sample_tag_file --multi
  -sample-tab tabular_file --specie-codes mmu --mismatches-in-
  seed 1 --verbose 1
```

Use Case 7: isomiR-SEA & Yara

isomiR-SEA tool performs some checks on tags length, discarding those that are too short or too long compared to the microRNA sequences used as reference. Moreover, during the alignment phase, the tool also discards all those tags for which no alignment has been found.

You can map all these isomiR-SEA discarded or not aligned tags directly on precursors with Yara mapper. For sake of completeness it is reported here an exhaustive call to isomiR-SEA: multi-sample is enabled (like the previous configuration) and the path to a tabular file is provided. After that Yara is called.

Shell commands

isomiR-SEA

```
# ./cmake-build-release/apps/isomir_sea --in-file-primir primir_file
  --in-file-gff gff_file --in-file-tags tag_file --multi-sample-tab
  tabular_file --path-discarded-tags path_discarded --specie-codes
  mmu --mismatches-in-seed 1 --verbose 1
```

Yara

```
# gzip -c precursor_file > REF.fasta.gz
# gzip -c isomirsea_discarded_tags_file > READS.fastq.gz
# yara_indexer REF.fasta.gz -o REF.index
# yara_mapper REF.index READS.fastq.gz -o READS.bam
```

or (if index already calculated)

```
# gzip -c isomirsea_discarded_tags_file > READS.fastq.gz
# yara_mapper index_already_calculated READS.fastq.gz -o READS.bam
```

Script

isomiR-SEA

```python
import subprocess
import sys

# Replace these paths with yours
exe_file = "./cmake-build-release/apps/isomir_sea"
primir_file = "./io_files_test/mirbase/hairpin.fa" # miRBase
gff_file = "./io_files_test/mirbase/all_gff.gff3" # miRBase
tag_file = "./io_files_test/tag/ENCFF005EAG_ENCFF008VOH.collapsed.fq"
tabular_file = "./io_files_test/tag/ENCFF005EAG_ENCFF008VOH.collapsed.tab"
path_discarded = "./io_files_test/tag/ENCFF005EAG_ENCFF008VOH.collapsed/discarded_tags.fq"

proc = subprocess.Popen([exe_file,
                         "--in-file-primir", primir_file,
                         "--in-file-gff", gff_file,
                         "--in-file-tags", tag_file,
                         "--multi-sample-tab", tabular_file,
                         "--path-discarded-tags", path_discarded,
                         "--specie-codes", "mmu",
                         "--mismatches-in-seed", "1",
                         "--verbose", "1"],
                         stdout=subprocess.PIPE, stderr=subprocess.PIPE)
for c in iter(lambda: proc.stderr.read(1), b''): # stderr(=debug_stream) is used for all program outputs
    sys.stdout.write(c)
for c in iter(lambda: proc.stdout.read(1), b''): # stdout used for "-h/--help"
    sys.stdout.write(c)
print("End.")
```

Figure 2.8: **Python script to run isomiR-SEA in an exhaustive configuration**. In the upper part you can see the paths to the several input files and output directory, in the central part there is the actual call to isomiR-SEA tool.

Yara

```
import gzip
import shutil
import os
import subprocess

# Replace these paths with yours
out_dir = "./io_files_test/tag/ENCFF005EAG_ENCFF008VOH.collapsed/yara"
discarded_tags = "./io_files_test/tag/ENCFF005EAG_ENCFF008VOH.collapsed/discarded_tags.fq" # IsomiR-SEA output
primirs = "./io_files_test/mirbase/hairpin.fa" # miRBase

print("Creating .gz file...")
if not os.path.exists(out_dir):
    os.mkdir(out_dir)
with open(discarded_tags, 'rb') as f_tag_in, gzip.open(out_dir + "/READS.fastq.gz", 'wb') as f_tag_out:
    shutil.copyfileobj(f_tag_in, f_tag_out)
with open(primirs, 'rb') as f_ref_in, gzip.open(out_dir + "/REF.fasta.gz", 'wb') as f_ref_out:
    shutil.copyfileobj(f_ref_in, f_ref_out)
print("Creating index file...")
proc1 = subprocess.Popen(["yara_indexer", out_dir + "/REF.fasta.gz", "-o", out_dir + "/REF.index"])
proc1.wait()
print("Mapping...")
proc2 = subprocess.Popen(["yara_mapper", out_dir + "/REF.index", out_dir + "/READS.fastq.gz",
                          "-o", out_dir + "/READS.sam"])
proc2.wait()
print("End.")
```

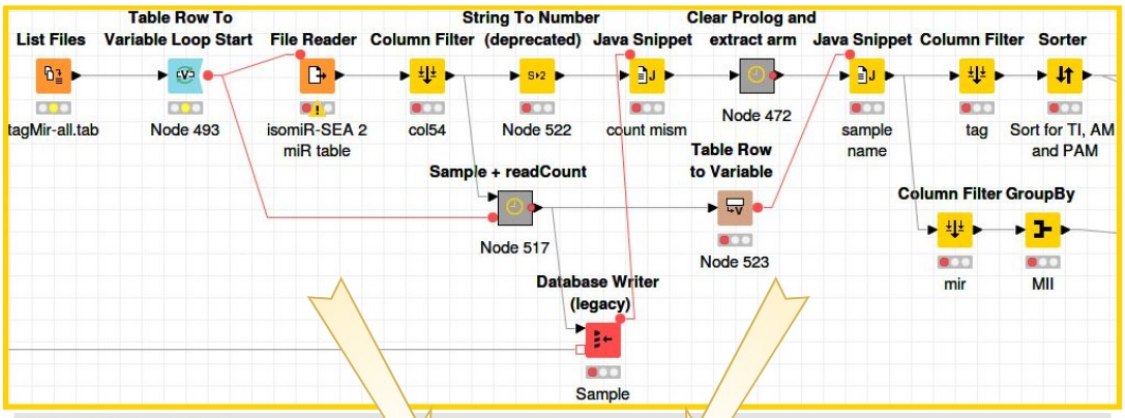
Figure 2.9: **Python script to run Yara on isomiR-SEA discarded tags**. In the upper part you can see the paths to the several input files and output directory, in the central part the compressed version of discarded tags and precursor files are produced, then there is the creation of the genome index and finally the mapping.

2.6 Post-analysis: from Knime to Python

The last part of the pipeline provides for a post-analysis phase in which the output files of isomiR-SEA are processed, in particular only the .tab files. In this way it is possible to extract only the most interesting information necessary, for instance, to perform downstream machine learning analysis or even to produce particular graphic reports which show, in a clearer way than a simple text file, the spectrum of isoforms and the interaction sites of miRNAs within a sample (see *Section 1.3*).

Previously, a tool called Knime was used for this post analysis phase, it is an open source software with a very intuitive graphical interface which offers a platform designed for data analytics and reporting. Although Knime is very convenient for prototyping, it flaws in terms of efficiency and performance essentially because it makes heavy use of disk memory, temporarily saving the results of each individual operation. Consequently, it is advisable to use Knime only to have a high-level view of the operations to be performed in order to obtain the desired result (in the form of a network composed of operational nodes). Once this is done and the model is consolidated, however, it is better to translate all the operations into code by adopting scripting languages. In this way, execution times are greatly shortened, avoiding the need to continuously use the internal memory. For this reason I chose to adopt Python3 programming language, translating what had previously been developed through interactive views into code instructions (see *Figure 2.10*).

Going more into detail, all the information of isomiR-SEA output files are now




```python
# Drop last column
df = df.drop(columns='Unnamed: 55')

# DataFrame of [ tag_name | iteration ]
df_sample_features = pd.DataFrame(data={'sample_isea_db_id': [os.path.splitext(filename)[0]], 'sample_isea_id': [i]})

# Add column of features
if os.path.exists(features_table):
    df_features = pd.read_csv(features_table, sep=",") # to be considered for the file structure
    df_sample_features = df_sample_features.join(df_features.set_index('uuid'), on='sample_isea_db_id')

# DataFrame of [ TC_sum | TI_count ]
df_sample_readcount = df[['TI','TC']].copy()
df_sample_readcount = df_sample_readcount.drop_duplicates(['TI', 'TC'])
TC_sum = df_sample_readcount['TC'].sum()
TI_count = df_sample_readcount['TI'].count()
df_sample_readcount.drop(df_sample_readcount.index, inplace=True) # clean DataFrame

# Concat df_sample_readcount to df_sample_features
df_sample_features['TC_sum'] = TC_sum
df_sample_features['TI_count'] = TI_count

# Save df_sample_features to Sample table
df_sample_features.to_sql(name='Sample', con=conn, if_exists='append', index=False) # write to SQL database
```

Figure 2.10: **Visual comparison between Knime project and Python script**. Above, an example of Knime project in which all the operations, including SQL queries and small code snippets, are shown in the form of nodes within a network. Below a Python script that performs similar operations being less understandable at first glance, but certainly much more efficient.

acquired and stored into SQL tables, which are subsequently transformed by performing operations similar to SQL queries. In order to do so I have chosen to rely on the Pandas library [46], written specifically for the Python language and useful for manipulating tabular data (which in Pandas are called *Dataframes*), also thanks to its interface with the SQLite database engine [47].

In particular this data processing phase is divided into three main steps:

- The first is a Python script that has the task of dividing the different reference miRNAs into three categories: *unique mapped, multiple mapped* and *discarded*. A miRNA is defined as *uniquely mapped to a tag* if their alignment has the maximum score and this score is strictly higher than all the other alignments of that particular tag, *multiple mapped to a tag* if there are more alignments with identical and maximum score, *discarded* in all other cases or

when the alignments have too many mismatches or too low scores. All these miRNAs, together with their alignment information, are stored in distinct tables forming a first SQL database. However, this filtering process was previously done *incorrectly* in Knime, with several inaccuracies (incorrect SQL queries) and oversights so, during Python porting, it has been completely revised. For example the SMD field (see *Section 2.5.2*) has been added as an additional column into isomiR-SEA output files, allowing an easier filtering in Python.

- The second step is another Python script that computes, *for every miRNA* that has not been discarded, several statistics useful to express the spectrum of isoforms and the conserved interaction sites in mathematical terms. In particular this script reads the SQL database created in the first step and then generates, for each miRNA with a different MII, some fields that summarize the type of isoform and the conserved (or not) interaction sites (e.g. collapsing INS, IOS, ISS, IPS, ICS fields into a string that can be similar to "FTTTT"). After that, if the miRNA under investigation is multiple mapped (has several valid alignments), we calculate statistics like average align score (AM) and length (AL), tag count (TC) sum and so on, producing a second SQL database. There is then a second part of the script in which we calculate some statistics to evaluate how many and which types of nucleotide substitutions occurred by comparing the miRNA sequences to the tag.

- Starting from this second SQL database, we can carry out different types of downstream analysis. These statistics in fact can be very useful for machine learning studies or, more in general, can be used to generate many different graphical reports, for instance in the form of stacked barchart (see *Section 1.3*) or pie chart, useful to better understand the detailed picture provided by isomiR-SEA tool about miRNAs, isomiRs and miRNA-mRNA interaction sites characterizing the samples under investigation.

Chapter 3

Results

We report here the results in term of execution time and max RAM consumption of three different isomiR-SEA releases. The oldest one dates back to 2016, it was a first prototype of the tool written in a non-optimized way and still not completely tested, we refer to it as *isomiR-SEA 1.6*. The intermediate version instead is defined *isomiR-SEA 2.0* and is from a couple of years ago. It has certainly undergone some improvements starting from the possibility of running the alignment algorithm in parallel exploiting concurrency, but still contained several bugs, one of which caused high RAM usage. The third and *new version of isomiR-SEA* is the one discussed in this thesis work, with this we have switched to a working and tested release, providing a product currently usable by an end bioinformatician user.

After that we will talk briefly about the decrease in execution times of the post analysis phase thanks to the use of Python code

Finally, we show some examples of stacked barchart generated directly from the post-processed output data of isomiR-SEA, to better understand the ability of the tool to provide detailed results, in the form of a concise and exhaustive chart.

3.1 Testing material and procedure

We tested all the three isomiR-SEA releases on a set of 10 samples coming from different house mouse (*Mus musculus*) tissues (see *Table 3.1*). These samples belongs to different datasets freely available online from the ENCODE project [48], and contain raw sequencing data (sequences with adapters), so each one of them was trimmed in order to remove the adapters, and then collapsed with BioSeqZip, generating 10 sample files each one containing unique sequences.

All these sequences have been analysed by adopting two miRNA reference databases one after the other: *miRBase* release 22 (v22), characterized by 48860 mature microRNAs from 271 species [49], and *MirGeneDB2.0*, which contains more than 10900 miRNAs from 45 organisms [50]. The configuration parameters of the

different isomiR-SEA versions have been set in the same way, in particular: 1 mismatch in the seed sequence and 3 outside, seed length to 6 (from nucleotide 2 to 7), minimum alignment threshold to 10 and minimum alignment score to 7.

The experiment were performed on a Linux machine with Intel Core i7-920 (4 cores, 8 threads) clocked at 2.67 GHz, 8 GB RAM and 1 TB HDD ATA Disk. As you can see these specifications are similar to those of today's laptops, meaning that these tests and analyzes can also be carried out at home, without advanced equipment like that of the research laboratories.

Sample name	Size (GB)
ENCFF005EAG.fastq	1.2
ENCFF008VOH.fastq	2.0
ENCFF009AAB.fastq	1.9
ENCFF013IGT.fastq	2.2
ENCFF015FMJ.fastq	1.5
ENCFF020DSK.fastq	1.0
ENCFF028KUU.fastq	1.2
ENCFF044SLN.fastq	1.2
ENCFF049APP.fastq	0.97
ENCFF051KKO.fastq	1.8

Table 3.1: **Sample name and size**.

3.2 Execution time and RAM consumption

The first tests were made using miRBase as the reference database for all the three versions of isomiR-SEA. Firstly, the execution times of the three releases were measured for each individual sample, as shown in *Figure 3.1*. As you can see *isomiR-SEA 1.6* took in almost all cases twice the time of the new release, and this is mainly due to the adoption of concurrency. Instead *isomiR-SEA 2.0*, which already adopted the concurrency, took a time similar to the current version only when the samples were small, while with larger samples a substantial difference begins to be seen in favour of the newest version. This is certainly due to the adoption of the new data structure, which allows to organize the data in memory more efficiently. However, in two out of ten cases *isomiR-SEA 2.0* was unable to complete the analysis ("killed" label), due to too high RAM usage that led the operating system to kill the process.

Then, the same type of analysis was made using MirGeneDB as the reference database, as shown in *Figure 3.2*. In this case however, only the current version of the tool and the 2.0 release were compared, as *isomiR-SEA 1.6* is not able to

work with databases other than miRBase. In this case the execution times are very similar mainly because the MirGeneDB database contains four times less miRNA sequences than miRBase, which means that the number of alignments produced is lower and the advantages of an optimized data structure are not highlighted. Anyway, you can still notice that the execution times of the *new isomiR-SEA* version are still shorter, on average, with respect to *isomiR-SEA 2.0* which, moreover, failed to complete the same two out of ten executions again.

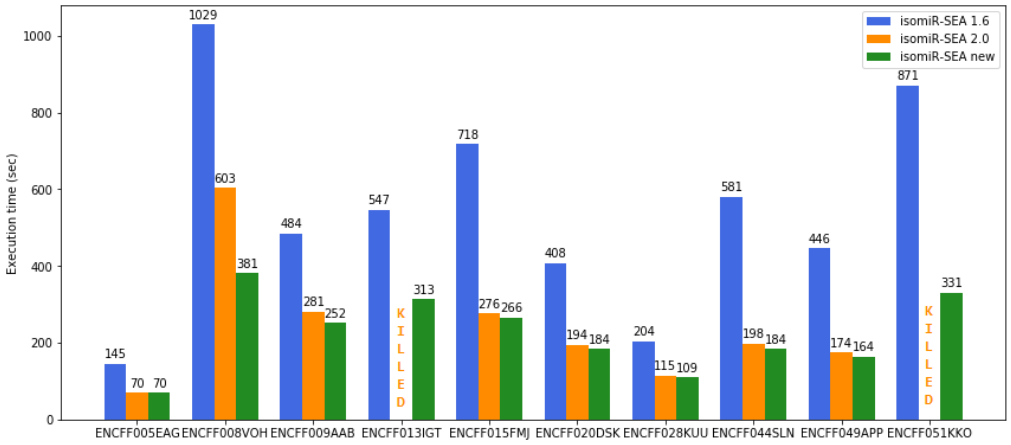


Figure 3.1: **Per-sample execution times of different isomiR-SEA releases (miRBase)**. On the y-axis are reported the execution times expressed in seconds while on the x-axis are reported the names of the 10 samples used in the analysis. Each group of three stacked bars refers to a single sample.

During the several executions, RAM consumption was also measured in order to better understand why some analysis were interrupted by the operating system. In particular *Figure 3.3* and *Figure 3.4* show the maximum values of RAM consumption reached when using miRBase and MirGeneDB respectively. As you can see the same two samples (ENCFF013IGT, ENCFF051KKO) that led to the interruption of the execution are also the ones that have achieved the highest RAM usage. Moreover, this number is limited to 8 only because that is the maximum memory capacity available in our system, which means that it could have been even greater. In any case, these two graphs make us understand how the introduction of a more efficient results printing procedure (discussed in *Section 2.4.3*) has allowed to considerably limit the use of resources. We can see in fact that *isomiR-SEA 2.0* on average reaches with miRBase a maximum value of RAM consumption 75% higher than the average of the current release. With respect to *isomiR-SEA 1.6* instead the RAM usage is only slightly lower but, as we have seen, the execution times are drastically decreased.

To get a clearer idea of the gain in terms of performance, in figure *Figure 3.5* we report, for each tool release, the sum of the execution times of each sample

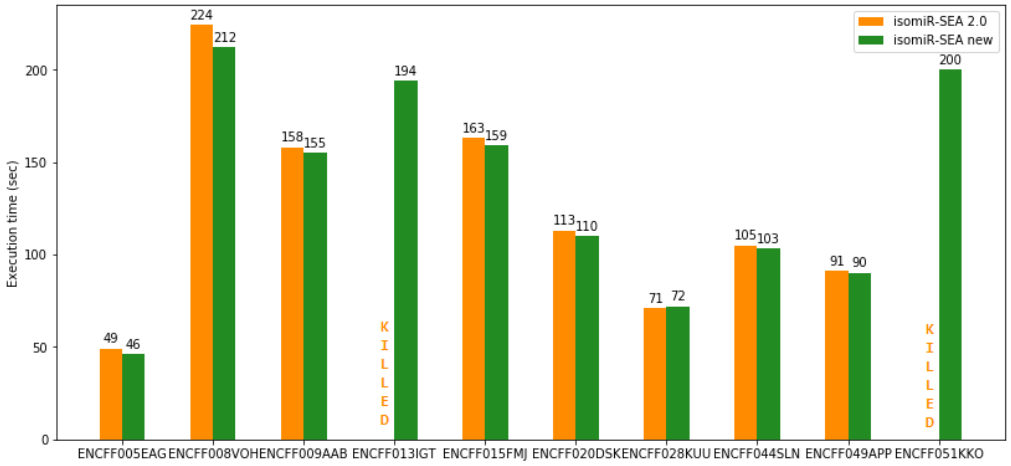


Figure 3.2: **Per-sample execution times of different isomiR-SEA releases (MirGeneDB)**. On the y-axis are reported the execution times expressed in seconds while on the x-axis are reported the names of the 10 samples used in the analysis. Each group of two stacked bars refers to a single sample.

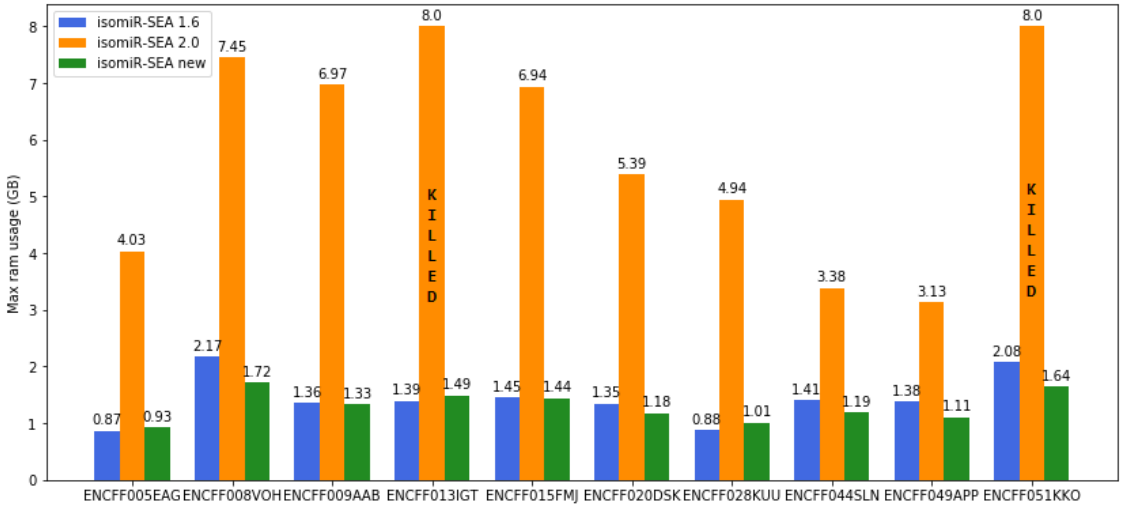


Figure 3.3: **Per-sample max RAM usage of different isomiR-SEA releases (miRBase)**. On the y-axis are reported the RAM usage values expressed in Gigabyte while on the x-axis are reported the names of the 10 samples used in the analysis. Each group of three stacked bars refers to a single sample.

(not considering the samples ENCFF013IGT and ENCFF051KKO that were problematic in *isomiR-SEA 2.0*). Looking at these graphs we can easily notice, for example, that the *new isomiR-SEA* release had a running time about 60% lower than *isomiR-SEA 1.6*, when using miRBase.

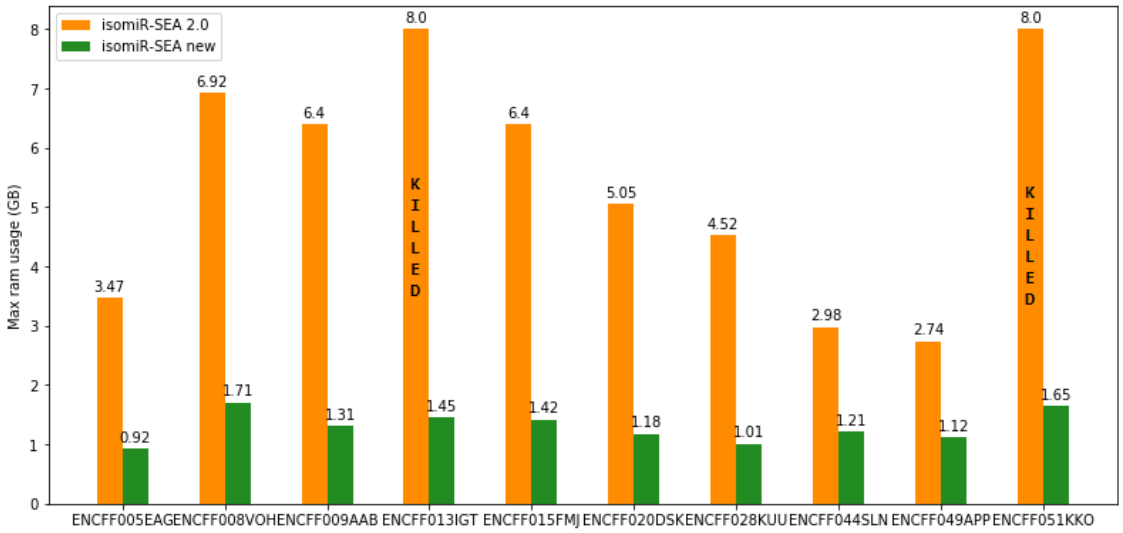
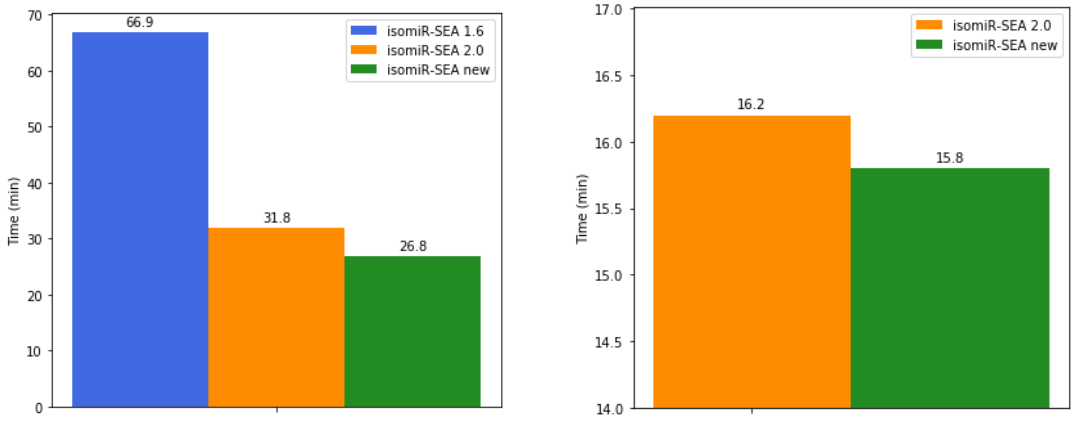


Figure 3.4: **Per-sample max RAM usage of different isomiR-SEA releases (MirGeneDB)**. On the y-axis are reported the RAM usage values expressed in Gigabyte while on the x-axis are reported the names of the 10 samples used in the analysis. Each group of two stacked bars refers to a single sample.

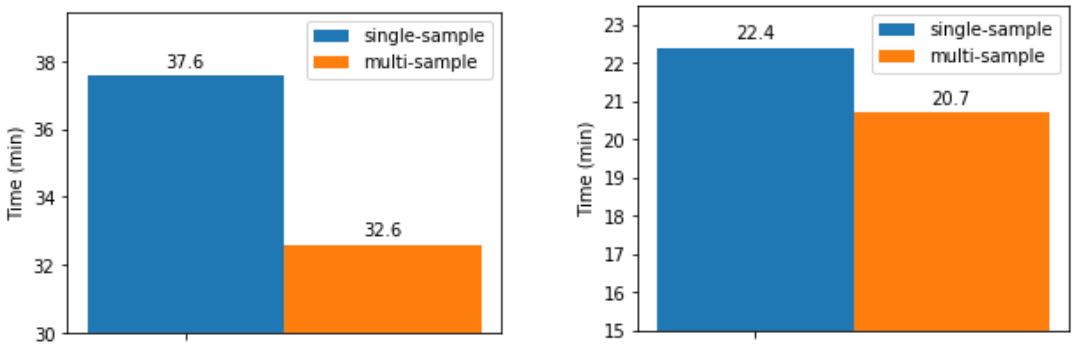


(a) miRBase (b) MirGeneDB

Figure 3.5: **Total execution time of different isomiR-SEA releases**. On the y-axis are reported the total execution times expressed in minutes. On the x-axis are reported the different isomiR-SEA releases.

Anyhow, considering only the version of isomiR-SEA described in this thesis, if instead of running the tool once for each sample we take advantage of the possibility (discussed in the *Section 2.4.2*) to run the program once on a single file that summarizes all the samples, execution time decreases even more. As you can see from the *Figure 3.6*, the total running time decreases considerably even just

collapsing our 10 samples used as tests.



(a) miRBase (b) MirGeneDB

Figure 3.6: **Single sample VS Multi sample total execution time**. Here are reported the total execution times expressed in minutes of isomiR-SEA when run in single sample and multi sample configuration.

Finally, as regards the post analysis phase, I measured the running times of the Python scripts used to generate the first two SQL databases which we discussed in *Section 2.6*, and from which it is possible to perform downstream machine learning analysis or produce comprehensive graphical reports. In particular I've not considered samples ENCFF013IGT and ENCFF051KKO that were problematic in *isomiR-SEA 2.0*. The results confirmed a significant gap in terms of execution time if the analysis is performed with Python scripts rather than with a Knime project:

- To generate the first SQL database: Knime 1090 s, Python 196 s.
- To generate the second SQL database: Knime 47180 s (>13 h), Python 1659 s (0.5 h).

This confirms that Knime, which is java-based, is a very useful tool in the prototyping phases, but once the model is established it is better to proceed with more performing languages, such as Python.

Chapter 4

Conclusions

This thesis presented a modern redesign of an alignment tool for RNA sequences and of the pipeline within which it is located. This pipeline is made up of several steps that range from genetic material extraction and sequencing up to the normalization of the tool outputs for differential analysis. We placed our attention on the final stages of this long process, in particular discussing the reimplementation of isomiR-SEA alignment tool, and the post processing of its results.

Our main goal was to provide a stable and efficient release of this software, in such a way to be easily used by a conscious end-user, also speeding up the post-analysis phase of its results.

After an introduction on the latest DNA and RNA sequencing techniques and on the evolution of alignment algorithms, we studied the features and functionalities of the peculiar small RNA sequences analyzed by isomiR-SEA, to highlight their fundamental role in the development of diseases and in human physiology. Subsequently we made a brief excursus on the libraries and programming languages adopted during the algorithm revisiting process, to point out the importance that the adoption of modern programming standards has in the development of a software. Such adaptation is actually crucial because it has allowed, as we have seen, to take advantage of the new features introduced with C++17 and C++20, and in the meantime to completely change the internal data structure of the program, which is now more streamlined and far more efficient.

This data structure has undergone numerous changes also thanks to the identification of some bugs, leading us towards the adoption of certain measures that really make the difference when working with huge amounts of data. By working in this way, and taking advantage of parallel execution (in search and alignment phases), we have reduced computational times up to ~60%.

Of all these changes, probably the most decisive concerns the new results printing process, which previously wasted a lot of RAM occupying the memory with temporary data and which instead now allows you to directly print every single

row of results without having to keep anything in memory. This measure drastically decreased the average max RAM consumption by ~75%.

In conclusion, this thesis has contributed to switch from a primitive version of the software to a stable release of a working tool, also thanks to the introduction of useful features such as input serialization or multi-sample analysis. But in order to be easily used by bioinformaticians, something was still missing. In fact, we discussed about the development of a reference manual which can be searched to understand how to configure the system according to personal needs, and about the whole downstream analysis porting into Python scripts. This last contribution is essential to generate useful data more quickly, in order to proceed with any machine learning analysis or graphical reports.

Bibliography

[1] B Hesper and P Hogeweg. "Bioinformatica: een werkconcept". In: *Kameleon* 1.6 (1970), pp. 28–29.

[2] Fatih Ozsolak and Patrice M Milos. "RNA sequencing: advances, challenges and opportunities". In: *Nature Reviews. Genetics* 12.2 (2011), pp. 87–98. DOI: 10.1038/nrg2934.

[3] Elisa Ficarra. "A glance to sequencing". University Lecture. 2016-2017. (accessed: 03.02.2020).

[4] URL: https://en.wikipedia.org/wiki/Small_RNA_sequencing#Small_RNA_sequencing. (accessed: 14.02.2020).

[5] Kary B Mullis. "The Unusual Origin of the Polymerase Chain Reaction". In: *Scientific American* 262.4 (1990), pp. 56–65. DOI: 10.1038/scientificamerican0490-56.

[6] Enzoklop. *Polymerase chain reaction*. URL: https://commons.wikimedia.org/wiki/File:Polymerase_chain_reaction.svg. (accessed: 03.02.2020).

[7] B Canard and RS Sarfati. "DNA polymerase fluorescent substrates with reversible 3' tags". In: *Gene* 148.1 (1994), pp. 1–6. DOI: 10.1016/0378-1119(94)90226-7.

[8] Nicole Rusk. "Torrents of sequence". In: *Nature Methods* 8.1 (2011), p. 44. DOI: 10.1038/nmeth.f.330.

[9] P Nyrén, B Pettersson, and M Uhlén. "Solid Phase DNA Minisequencing by an Enzymatic Luminometric Inorganic Pyrophosphate Detection Assay". In: *Analytical Biochemistry* 208.1 (1993), pp. 171–175. DOI: 10.1006/abio.1993.1024.

[10] Robert England and Monica Pettersson. "Pyro Q-CpG: quantitative analysis of methylation in multiple CpG sites by Pyrosequencing". In: *Nature Methods* 2 (2005). DOI: 10.1038/nmeth800.

[11] URL: https://seqan.readthedocs.io/en/master/Tutorial/DataStructures/Alignment/ScoringSchemes.html. (accessed: 07.02.2020).

[12] VI Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". In: *Soviet Physics Doklady* 10 (1966), p. 707. DOI: 1966SPhD...10..707L.

[13] RW Hamming. "Error detecting and error correcting codes". In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[14] MO Dayhoff and RM Schwartz. "A model of Evolutionary Change in Proteins, in Atlas of protein sequence and structure". In: *Nat. Biomed. Res. Found.* 5 (1978), pp. 345–358.

[15] S Henikoff and JG Henikoff. "Amino acid substitution matrices from protein blocks". In: *PNAS* 89.22 (1992), pp. 10915–10919. DOI: 10.1073/pnas.89.22.10915.

[16] URL: https://en.wikipedia.org/wiki/Gap_penalty#Types. (accessed: 07.02.2020).

[17] Gianvito Urgese et al. "Dynamic Gap Selector: A Smith Waterman Sequence Alignment Algorithm with Affine Gap Model Optimisation". In: (2014). URL: http://iwbbio.ugr.es/2014/papers/IWBBIO_2014_paper_143.pdf. (accessed: 11.02.2020).

[18] SB Needleman and CD Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–455. DOI: 10.1016/0022-2836(70)90057-4.

[19] URL: http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?. (accessed: 10.02.2020).

[20] TF Smith and MS Waterman. "Identification of common molecular subsequences". In: *Journal of Molecular Biology* 147.1 (1981), pp. 195–197. DOI: 10.1016/0022-2836(81)90087-5.

[21] SF Altschul et al. "Basic Local Alignment Search Tool". In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410. DOI: 10.1016/S0022-2836(05)80360-2.

[22] URL: https://en.wikipedia.org/wiki/BLAST_(biotechnology)#Algorithm. (accessed: 11.02.2020).

[23] M Burrows and DJ Wheeler. "A block-sorting lossless data compression algorithm". In: (1994).

[24] Ben Langmead et al. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". In: *Genome Biology* 10.3 (2009). DOI: 10.1186/gb-2009-10-3-r25.

[25] H Li and R Durbin. "Fast and accurate short read alignment with Burrows-Wheeler transform". In: *Bioinformatics* 25.14 (2009), pp. 1754–1760. DOI: 10.1093/bioinformatics/btp324.

[26] URL: https://www.slideshare.net/Fardin6600/blast-algorithm. (accessed: 11.02.2020).

[27] Chao Cheng, Jason Moore, and Casey Greene. "Applications of bioinformatics to non-coding RNAs in the era of next-generation sequencing". In: *Pacific Symposium on Biocomputing* 19 (Jan. 2014), pp. 412–6.

[28] Gianvito Urgese. *Computational Methods for Bioinformatics Analysis and Neuromorphic Computing: Efficient Gap Model for Sequence Alignment Algorithms, MiRNA and IsomiR Detection in RNA-seq Data, and Spiking Neural Network Simulations Placement in Neuromorphic Platfmorms: PhD Dissertation*. 2016. URL: https://books.google.it/books?id=8IzAuQEACAAJ.

[29] DP Bartel. "MicroRNA Target Recognition and Regulatory Functions". In: *Author manuscript* 136.2 (2009), pp. 215–233. DOI: 10.1016/j.cell.2009.01.002.

[30] Chanseok Shin et al. "Expanding the MicroRNA Targeting Code: Functional Sites with Centered Pairing". In: *Molecular cell* 38.6 (2010), pp. 709–802. DOI: 10.1016/j.molcel.2010.06.005.

[31] M Li et al. "MicroRNAs: Control and Loss of Control in Human Physiology and Disease". In: *World journal of surgery* 33.4 (2009), pp. 667–684. DOI: 10.1007/s00268-008-9836-x.

[32] GP George and Rama Devi Mittal. "MicroRNAs: Potential biomarkers in cancer". In: *Indian Journal of Clinical Biochemistry* 25.1 (2010), pp. 4–14. DOI: 10.1007/s12291-010-0008-z.

[33] Gianvito Urgese et al. "BioSeqZip: a collapser of NGS redundant reads for the optimisation of sequence analysis". In: *Bioinformatics* (Jan. 2020). ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa051.

[34] URL: http://samtools.github.io/hts-specs/SAMv1.pdf. (accessed: 13.02.2020).

[35] Matthias Dodt et al. "FLEXBAR-Flexible Barcode and Adapter Processing for Next-Generation Sequencing Platforms". In: *Biology* 1.3 (2012), pp. 895–905. DOI: 10.3390/biology1030895.

[36] Martin Marcel. "Cutadapt Removes Adapter Sequences From High-Throughput Sequencing Reads". In: *EMBnet.journal* 17.1 (2011), pp. 10–12. ISSN: 2226-6089. DOI: doi.org/10.14806/ej.17.1.200.

[37] URL: https://en.cppreference.com/w/cpp/17. (accessed: 02.03.2020).

[38] URL: https://github.com/seqan/seqan3. (accessed: 20.02.2020).

[39] Gene Myers. "A fast bit-vector algorithm for approximate string matching based on dynamic programming". In: *Journal of the ACM (JACM)* 46.3 (1999), pp. 395–415. DOI: doi:10.1145/316542.316550.

[40] W Shane Grant and Randolph Voorhies. *cereal - A C++11 library for serialization*. 2017. URL: http://uscilab.github.io/cereal/. (accessed: 26.02.2020).

[41] Enrico Siragusa, David Weese, and Knut Reinert. "Fast and accurate read mapping with approximate seeds and multiple backtracking". In: *Nucleic Acids Research* 41.7 (2013), p. 78. DOI: doi.org/10.1093/nar/gkt005.

[42] URL: https://philae.polito.it/gitlab/gurgese/isomiR-SEA.

[43] URL: https://seqan.readthedocs.io/en/master/Infrastructure/Use/Install.html#infra-use-install. (accessed: 28.02.2020).

[44] URL: http://packages.seqan.de/. (accessed: 28.02.2020).

[45] URL: https://github.com/seqan/seqan/blob/master/apps/yara/README.rst. (accessed: 28.02.2020).

[46] URL: https://pandas.pydata.org/. (accessed: 03.03.2020).

[47] URL: https://www.sqlite.org/index.html. (accessed: 03.03.2020).

[48] URL: https://www.encodeproject.org/. (accessed: 04.03.2020).

[49] Ana Kozomara, Maria Birgaoanu, and Sam Griffiths-Jones. "miRBase: from microRNA sequences to function". In: *Nucleic Acids Research* 47.D1 (2019), pp. D155–D162. DOI: 10.1093/nar/gky1141.

[50] Bastian Fromm et al. "MirGeneDB 2.0: The metazoan microRNA complement". In: (2018). DOI: doi.org/10.1101/258749.

Ringraziamenti

Desidero ringraziare il mio relatore Gianvito Urgese per la sua estrema disponibilità e per avermi accompagnato nella stesura di questa tesi, condividendo con me la sua preziosa esperienza nel campo della bioinformatica.

Insieme a lui vorrei ringraziare anche Emanuele Parisi per essermi stato di aiuto, anche a distanza, soprattutto nel primo periodo, quando ancora avevo bisogno di orientarmi.

Un ringraziamento speciale va ai miei genitori che sono sempre stati al mio fianco, supportandomi e permettendomi di concludere questi studi, assecondando sempre tutte le mie scelte.

Ringrazio mio fratello per essere stato sempre un punto di riferimento e avermi spinto, forse anche inconsciamente, a intraprendere questo percorso.

Ringrazio la mia ragazza Giulia per essere riuscita a sopportarmi durante questo ultimo anno di duro lavoro, sapere di averla accanto rende tutto più semplice.

Ringrazio i miei compagni per tutti i bei (e un po' meno bei) momenti passati insieme; in particolare Matteo e Nicolò con i quali ho trascorso innumerevoli ore, confrontandomi e condividendo pareri.

Vorrei ringraziare inoltre il mio storico coinquilino Matteo, per la spensieratezza con cui abbiamo affrontato la vita universitaria, e Hasan, per i suoi preziosi consigli e le interminabili chiacchierate.

Infine, una dedica particolare va a mia nonna, una persona speciale con cui ho avuto la fortuna di crescere, e che desidera vedere quella corona di alloro quasi più di me.