International Journal of Foundations of Computer Science

SpliceTAPyR - An efficient method for transcriptome alignment

Andreia Sofia Teixeira
IDSS Lab, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa,
Rua Alves Redol - 9, 1000-029 Lisboa, Portugal
sofia.teixeira@tecnico.ulisboa.pt

Francisco Fernandes
VIMI group, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa,
Rua Alves Redol - 9, 1000-029 Lisboa, Portugal
francisco.fernandes@tecnico.ulisboa.pt

Alexandre P. Francisco
IDSS Lab, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa,
Rua Alves Redol - 9, 1000-029 Lisboa, Portugal
alexandre.francisco@tecnico.ulisboa.pt

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

RNA-Seq is a Next-Generation Sequencing (NGS) protocol for sequencing the RNA in a cell and generates millions of sequence fragments, reads, in a single run. These reads can be used to measure levels of gene expression and to identify novel splice variants of genes. One of the critical steps in an RNA-Seq experiment is mapping NGS reads to the reference genome. Because RNA-Seq reads can span over more than one exon in the genome, this task is challenging. In the last decade, tools for RNA-Seq alignment have emerged, but most of them demands a considerable amount of time or/and memory. The most common approach takes two phases: first, the map of the reads that have a direct match in the reference, setting aside the others as initially unmapped reads; second, the use of heuristics based approaches, clustering or even annotations, to decide where to align the later. This work presents an efficient computational solution for the problem of transcriptome alignment, named SpliceTAPyR. It relies on compressed full-text indexing methods and succinct data structures to efficiently align RNA-Seq reads in a single phase. This way we are able to achieve the same or better accuracy than other tools while using considerably less memory and time to the most competitive tools.

Keywords: Transcriptome; RNA-Seq; Next-Generation Sequencing; Sequence alignment; Splice junctions.

1. Introduction

The complete genome sequencing of various organisms, including the human genome, has been one of the centers of the attention of the scientific community in the last decade. All this sequencing efforts, which aims at determining which

molecules constitute the deoxyribonucleic acid (DNA), result in a large amount of biological information that must be stored in databases and manipulated in order to convert this raw data into usable knowledge.

For many years, the standard methods for determining the sequence of transcribed genes, consisted in the sequencing of messenger ribonucleic acid (mRNA) using complementary DNA (cDNA) through the conventional method of Sanger [19], using Expressed Sequence Tags [1] or via microarrays [4]. The RNA-Seq method has brought many advantages over the other conventional methods. This new method uses Next Generation Sequencing (NGS) technologies, yielding sequences of messenger RNA (mRNA) with fewer errors. This new approach produces more data from each experiment, allowing this data to be used as a direct measure of the gene expression levels. RNA-Seq experiments not only capture the transcriptome, i.e., all RNA sequences present in a cell, but they also replace the conventional experiences with microarrays. RNA-Seq is probably one of the most complex next-generation sequencing applications. Expression levels, differential splicing, allele-specific expression, RNA editing and fusion transcripts yield important information when comparing samples for disease-related studies.

One of the critical steps in an RNA-Seq experiment is the mapping of the reads, generated by NGS technologies, to a reference genome, or a reference transcriptome. Currently, there are several [12] software tools developed to make the direct alignment of these fragments in a reference genome. However, due to the splicing phenomenon of eukaryotic cells, that regards a transcript that may contain parts of more than one exon, these tools may fail, since the case in which a read spans to more than one exon is not covered. Because RNA-Seq reads can be short, the task is challenging. In order to achieve these fragments' alignment, it is necessary to develop methods that can identify the splice junctions, that is, the boundaries between exons and introns, thereby allowing a correct identification of the source of the transcript in question.

In the last decade, tools for RNA-Seq alignment have emerged [8] but most of those tools find junctions by mapping reads to the reference in two phases [20, 21]. In the first phase, the pipeline tries to map all the reads to the reference genome, and all the reads that do not map are set aside as initially unmapped reads. Then, they use heuristic-based approaches, or even annotations, to decide where to align them.

This work addresses the problem of transcriptome alignment and presents an efficient computational solution for it, named SpliceTAPyR, which identifies splice junctions and relies on compressed full-text indexing methods and succinct data structures to deliver a fast and accurate read mapping. SpliceTAPyR extends the functionality of the existing DNA-only sequence alignment tool TAPyR [9], being able to align RNA-Seq reads in only one phase.

2. Problem

RNA-Seq [22] is an approach to transcriptome profiling that uses deep-sequencing technologies. In general, a population of RNA (total or fractionated) is converted to a library of cDNA fragments with adapters attached to one or both ends. Each molecule, with or without amplification, is then sequenced in a high-throughput manner to obtain short sequences from one end (single-end sequencing) or both ends (pair-end sequencing). RNA-Seq generates millions of sequence fragments in a single run. These fragments, or reads, are typically 30 to 800 base pairs (bp) long, but can go up to more than 1000bp, depending on the DNA-sequencing technology used. They can be used to measure levels of gene expression and to identify novel splice variants of genes. The main goal is to produce a transcriptional map in genomic scale with the transcriptional structure and the expression level of each gene.

One of the critical steps in an RNA-Seq experiment is mapping NGS reads to the reference transcriptome. However, because transcriptomes are incomplete even when it comes to well studied species such as human and mouse, RNA-Seq analyses are forced to map to the reference genome as a proxy for the transcriptome. In a typical application, we may have to align hundreds of millions of reads to a reference genome that can be as large as few gigabases and with RNA-Seq reads each read can span through more than one exon. This creates a new challenge: we have to be able to map reads that do not have a sequential alignment in the reference genome, allowing the read to be partitioned in two or more fragments, and we have to be able to identify boundaries between exons and introns, splice sites, to be certain that seeds are aligned correctly. This is a job that can not be efficiently achieved through standard dynamic programming procedures. Robust indexing methods and succinct data structures are needed for an efficient mapping.

In this work we present SpliceTAPyR, based on TAPyR [9], which uses an implementation of the FM-Index [10] optimized for the DNA alphabet and employs a multiple seed approach to anchor the best candidate alignment, followed by an efficient strategy to identify splice signals. This information about splice evidence is then used to guide the correct alignment of the transcriptomic reads.

3. Approach

TAPyR is a method for the alignment of high-throughput next generation DNA sequencing reads, like the ones produced by pyrosequencing using the 454 GS FLX platform [16], by semiconductor-based sequencing on Ion Torrent [18] instruments, or using Illumina's [3] sequencing-by-synthesis approach. Its procedure manages to explore specific data characteristics to achieve improved performance over other mainstream methods. Like many of those methods, such as BWA [14] or Bowtie [13], TAPyR also builds an index of the reference sequence to accelerate the alignment. It then employs a multiple seed approach to anchor the best candidate alignment. Contrary to other seed-based alignment tools, TAPyR's strategy adds more flexibility by dispensing the need to determine the number and length of the seeds beforehand.

The approach relies on the fact that, for re-sequencing projects, the optimal alignments are mostly composed of relatively large chunks of exact matches interspersed by small, possibly gapped, divergent regions. A banded dynamic programming is used to finish up the candidate multiple seed alignments, considering user-specified error constraints.

The choice of TAPyR is due to the fact that the number of aligners that support long reads, like GS FLX pyrosequencing or Ion Torrent data, is relatively scarce compared to other technologies, most notably Illumina. Moreover, some of these tools find their origins before the advent of the new sequencing technologies and were then adapted to cope with new kinds of data, as well as others that target multiple kinds of data which are not optimized for sequencing data. Given this state of affairs, it's reasonable to think that there is still room for improvement in the realm of available aligners specifically designed for high-throughput sequencing data [9].

Taking advantage of all the characteristics of this computational model, we created a new tool named SpliceTAPyR, that aligns both DNA and RNA reads. In the second case, SpliceTAPyR reports the existence of splice evidence, if there is any, in a sequential manner, i.e., the reads are processed only once.

3.1. *Algorithm*

To better understand the algorithmic details behind SpliceTAPyR's approach, we begin by formally defining some important concepts and introducing some basic notions about strings, sequences and indexes in the field of Bioinformatics.

Let $\Sigma = \{\alpha_1, \ldots, \alpha_{|\Sigma|}\}$ be a finite ordered alphabet, and let Σ^* be the set of all strings over Σ, including the empty string ε. Let T be a string or text over Σ^* which is always terminated by a special alphabet character \$, which is lexicographically smaller than any other character in Σ and does not occur anywhere else in T. Let $T[i]$ denote the character at position i in T, for $0 \leq i < n$, where $n = |T|$. This way, we define $T[i \ldots j]$ as the substring of length $(j - i + 1)$ starting at the i-th position and ending at the j-th position of T, where $0 \leq i \leq j < n$. We call T_i the i-th suffix of T, i.e., the substring $T[i \ldots (n-1)]$, with $0 \leq i < n$. In the same way, the substring $T[0 \ldots i]$, $0 \leq i < n$ corresponds to a prefix of T.

The *suffix array* (SA) [15] of T is an array of size n of numbers corresponding to the lexicographical ordering of the n suffixes of T, i.e. a permutation of the integers $\{0, \ldots, (n-1)\}$ such that $T_{SA[0]} < T_{SA[1]} < \ldots < T_{SA[n-1]}$. An example of the SA for a small string is presented in Figure 1(a).

The *Burrows-Wheeler Transform* (BWT) [5] of T is a permutation of the characters of T such that $BWT[i]$ corresponds to the character preceding the i-th lexicographically ordered rotation of T, i.e. $BWT[i] = T[SA[i] - 1]$ if $SA[i] \neq 0$ and $BWT[i] =$ \$ otherwise. If we consider the conceptual matrix M consisting of all the sorted rotations of T, the BWT array corresponds to the last column of M. In the previous example shown in Figure 1(b), the BWT of the illustrated string is

i	SA[i]	$T_{SA[i]}$
0	13	$
1	4	AACTGCAGT$
2	5	ACTGCAGT$
3	1	AGCAACTGCAGT$
4	10	AGT$
5	3	CAACTGCAGT$
6	0	CAGCAACTGCAGT$
7	9	CAGT$
8	6	CTGCAGT$
9	2	GCAACTGCAGT$
10	8	GCAGT$
11	11	GT$
12	12	T$
13	7	TGCAGT$

(a)

i	occ(c,i)				BWT	F		L
	A	C	G	T				
0	0	0	0	1	T	$	CAGCAACTGCAG	T
1	0	1	0	1	C	A	ACTGCAGT$CAG	C
2	1	1	0	1	A	A	CTGCAGT$CAGC	A
3	1	2	0	1	C	A	GCAACTGCAGT$	C
4	1	3	0	1	C	A	GT$CAGCAACTG	C
5	1	3	1	1	G	C	AACTGCAGT$AC	G
6	1	3	1	1	$	C	AGCAACTGCAGT	$
7	1	3	2	1	G	C	AGT$CAGCAACT	G
8	2	3	2	1	A	C	TGCAGT$CAGCA	A
9	3	3	2	1	A	G	CAACTGCAGT$C	A
10	3	3	2	2	T	G	CAGT$CAGCAAC	T
11	4	3	2	2	A	G	T$CAGCAACTGC	A
12	4	3	3	2	G	T	$CAGCAACTGCA	G
13	4	4	3	2	C	T	GCAGT$CAGCAA	C

(b)

Fig. 1. **(a)** Suffix array for the string "CAGCAACTGCAGT$". **(b)** Matrix of all the rotations of the same string evidencing its BWT, the first and last columns F and L, and the counts of each alphabet character c along the BWT in $occ(c, i)$. The last characters of the rotations are grayed out to show the difference from each corresponding suffix.

"TCACCG$GAATAGC".

Using the BWT together with some extra information about the SA and character occurrences, we can build another indexing structure called the *FM-Index* [10, 11]. One of its key concepts is the Last-to-First column mapping (LF-mapping), which finds, for each position i, the position j such that $SA[j] = SA[i]-1) \bmod n$. Like the name suggests, it simply maps the k-th occurrence of each symbol in the last column L to the k-th occurrence of the same symbol in the first column F. In other words, and noting that the BWT is in fact the last column L, if $L[i] = BWT[i] = T[(SA[i] - 1) \mod n] = c$ is the k-th occurrence of the character c in the last column L, then we will have $LF[i] = j$ where $F[j] = T[SA[j]] = c$ is the k-th occurrence of the same character in the first column F. For the example in Figure 1(b), $LF[0] = 12$ for character 'T' and $LF[1] = 5$ for character 'C'. The LF-mapping can be efficiently computed by setting:

$$LF[i] = C[c]+occ(c, i)-1, \text{ where: } \begin{cases} c = BWT[i] \\ C[c] \text{ is the total number of occurrences in } T \\ \quad \text{of all the characters strictly smaller than } c \\ occ(c, i) \text{ is the number of occurrences of } c \\ \quad \text{in } BWT[0 \dots i] \end{cases}$$

Since the BWT stores characters and not numbers, the FM-Index space requirements are $\mathcal{O}(n \log(|\Sigma|))$. Pattern matching on a pattern P of size m is done accordingly to the procedure *BackwardSearch* in Algorithm 1. The search is done in

Algorithm 1 BackwardSearch.

```
1: procedure BackwardSearch(pattern P of size m)
2:     c ← P[m − 1] ; first ← C[c] ; last ← C[c + 1] − 1 ; i ← m − 2
3:     while ((first ≤ last) and (i ≥ 0)) do
4:         c ← P[i]
5:         first ← C[c]+Occ(c, first − 1)
6:         last ← C[c]+Occ(c, last)−1
7:         i ← i − 1
8:     end while
9:     if (last < first) then
10:        return [ ]
11:    else
12:        return [first, last]
13:    end if
14: end procedure
```

i	SA	LF	BWT	$T_{SA[i]}$
0	13	12	T	$
1	4	5	C	AACTGCAGT$
2	5	1	A	ACTGCAGT$
3	1	6	C	AGCAACTGCAGT$
4	10	7	C	AGT$
5	3	9	G	CAACTGCAGT$
6	0	0	$	CAGCAACTGCAGT$
7	9	10	G	CAGT$
8	6	2	A	CTGCAGT$
9	2	3	A	GCAACTGCAGT$
10	8	13	T	GCAGT$
11	11	4	A	GT$
12	12	11	G	T$
13	7	8	C	TGCAGT$

(a)

i	SA	LF	BWT	$T_{SA[i]}$
0	13	12	T	$
1	4	5	C	AACTGCAGT$
2	5	1	A	ACTGCAGT$
3	1	6	C	AGCAACTGCAGT$
4	10	7	C	AGT$
5	3	9	G	CAACTGCAGT$
6	0	0	$	CAGCAACTGCAGT$
7	9	10	G	CAGT$
8	6	2	A	CTGCAGT$
9	2	3	A	GCAACTGCAGT$
10	8	13	T	GCAGT$
11	11	4	A	GT$
12	12	11	G	T$
13	7	8	C	TGCAGT$

(b)

i	SA	LF	BWT	$T_{SA[i]}$
0	13	12	T	$
1	4	5	C	AACTGCAGT$
2	5	1	A	ACTGCAGT$
3	1	6	C	AGCAACTGCAGT$
4	10	7	C	AGT$
5	3	9	G	CAACTGCAGT$
6	0	0	$	CAGCAACTGCAGT$
7	9	10	G	CAGT$
8	6	2	A	CTGCAGT$
9	2	3	A	GCAACTGCAGT$
10	8	13	T	GCAGT$
11	11	4	A	GT$
12	12	11	G	T$
13	7	8	C	TGCAGT$

(c)

i	SA	LF	BWT	$T_{SA[i]}$
0	13	12	T	$
1	4	5	C	AACTGCAGT$
2	5	1	A	ACTGCAGT$
3	1	6	C	AGCAACTGCAGT$
4	10	7	C	AGT$
5	3	9	G	CAACTGCAGT$
6	0	0	$	CAGCAACTGCAGT$
7	9	10	G	CAGT$
8	6	2	A	CTGCAGT$
9	2	3	A	GCAACTGCAGT$
10	8	13	T	GCAGT$
11	11	4	A	GT$
12	12	11	G	T$
13	7	8	C	TGCAGT$

(d)

Fig. 2. Backward search of the pattern "TGCA" in the BWT index of the text "CAGCAACT-GCAGT$". The active SA ranges in each step are highlighted as well as the current matched substring.

$\mathcal{O}(m)$ time and is performed backwards from $P[m-1]$ to $P[0]$, using two pointers, $first$ and $last$, that define the top and bottom boundaries of an interval in the BWT array, and iteratively applying the LF-mapping rule (lines 5 and 6). Since the interval in the BWT is exactly the same interval in the SA, the positions of the pattern in the text are given by $SA[first], SA[first+1], \ldots, SA[last]$. At each step i, all the $(last - first + 1)$ suffixes $T_{SA[first]}, \ldots, T_{SA[last]}$ share the same prefix $P[(i+1)\ldots(m-1)]$ of length $(m-i-1)$. The LF-mapping rule is then applied on that interval to obtain the next interval corresponding to suffix $P[i\ldots(m-1)]$.

Following Figure 2, the algorithm works in reverse by first matching the last character of the pattern "TGCA", the letter 'A'. It is now at the SA interval $[1, 4]$ which corresponds to all the four suffixes of the text starting by the letter 'A'. For the next letter to the left in the pattern, 'C', $LF[1]$ and $LF[4]$ give us the next

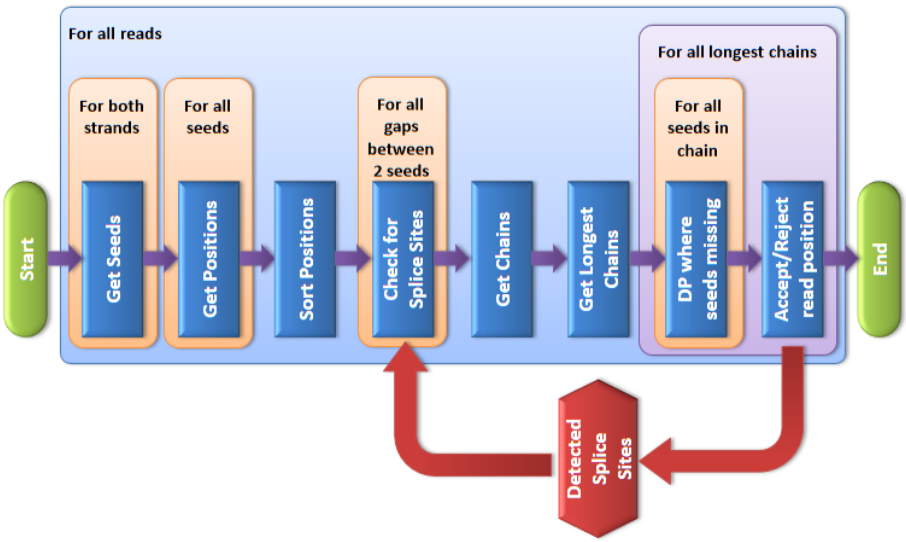

Fig. 3. SpliceTAPyR's main workflow.

SA interval $[5, 7]$ which correspond to the three suffixes in the text starting by the substring "CA". The algorithm proceeds in the same way until it reaches the singular interval $[13, 13]$ meaning that the pattern "TGAC" has only one occurrence at position $SA[13] = 7$ in the text.

3.2. *Read mapping*

SpliceTAPyR aligns RNA reads against the reference genome by relying on both the FM-Index and on possible previously identified splice sites to find dynamic seeds of variable length. This way, we are able to detect splice sites and bypass substitution, insertion and deletion errors. The basic layout for the global process is shown in Figure 3.

It begins by finding the seeds for each read by doing the following:

- Starting from the right side of the read (because the search in the FM-index is done backwards, the characters are matched one by one against the index of the reference genome, using the previously described *BackwardSearch* procedure (Figure 2), until a mismatch occurs, i.e. until it returns an empty interval. This means that the substring up until the penultimate checked character still exists in the reference, but with the last one added it does not. We now have our first seed, represented by the last non-empty interval found.
- The last seen character is skipped over, because it very likely corresponds to a mismatch or an *indel* (insertion or deletion), and we start searching

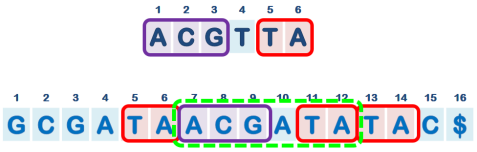

Fig. 4. Example of the seeds found by the algorithm. The top string represents the read, and the bottom string the genome.

for the next seed on the next character to the left.

- This process is repeated until we reach the left end of the read.

A short example of the seeds generated by this algorithm is shown in Figure 4. We start matching characters starting from position 6 of the read, and get the seed "TA" since the substring "TTA" no longer exists in the genome. We skip the second 'T' and we start the second seed at position 3 which goes up to the beginning of the read. Based on the occurrences of the two seeds found, we could easily infer that the read occurs at position 7 of the genome with one mismatch error.

For each read R we will end up with a set $s_1, \ldots, s_n$ of n seeds, each with length m_i, starting at position q_i in the read, corresponding to the substring $R[q_i, \ldots, q_i + m_i - 1]$ and defined by the interval $[l_i, r_i]$ in the index, for $1 \leq i \leq n$. If the read occurs in the genome with no errors, we will have a single seed s_1 corresponding to the entire read. In general, if a read has k errors the algorithm will generally produce $(k + 1)$ seeds. This seed searching process is repeated for both forward and reverse complemented strands of the read and the strand with the less and consequently larger seeds is selected to proceed, or both of them if they share the same number of seeds.

Then, for each seed s_i, we get all its $(r_i - l_i + 1)$ occurrences in the genome by retrieving from the index the positions $SA[l_i], SA[l_i + 1], \ldots, SA[r_i]$. This is done for all the n seeds and the resulting N total positions are stored into a single array using pairs (p, i) where each position p is associated with its originating seed i, and then sorted by position. This list is then processed once from left to right to find the chains of seeds that occur in the same order both in the read and in the genome, and in which each pair of consecutive seed occurrences satisfies one of these conditions:

(a) both seeds have similar distance in the read and in the genome,
(b) the right end of the left seed and the left end of the right seed correspond to previously identified splice site positions,
(c) the distance between the two seeds is consistent with the size of an intron and the positions in the genome right next to the inner ends of the seeds display splice signals.

These chains of seeds correspond to possible candidates of occurrences of the read in the genome. Formally, a chain of size h is defined as the maximal set of h consecutive seed positions $(p_j, i_j), \ldots, (p_{j+h-1}, i_{j+h-1})$, starting at a seed occurrence j and ending at seed occurrence $(j + h - 1)$, with $1 \leq j \leq (j + h - 1) \leq N$,

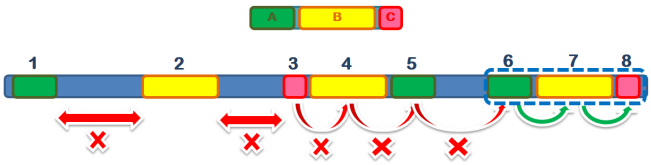

Fig. 5. Example of how the occurrences of the 3 seeds A, B and C of a read are processed to find the chain of seeds that corresponds to the read position in the genome.

that satisfies the conditions:

$$\begin{cases} p_k < p_{k+1} \\ i_k < i_{k+1} \\ |(p_{k+1} - p_k) - (q_{i_{k+1}} - q_{i_k})| \leq \epsilon \\ p_{k+1} - (p_k + m_{i_k}) \leq \theta \end{cases}$$

for all $j \leq k \leq (j + h - 1)$ and some values ϵ and θ, and where neither the seed occurrence $(j - 1)$ to the left, if existent, neither the seed occurrence $(j + h)$ to the right, if existent, can be included in the mentioned chain. Additionally, a single seed that verifies these last circumstances does also qualify as a chain. The given threshold ϵ depends on the minimum read identity percentage or the maximum number of errors we want to allow, and the threshold θ sets limits on the size of the gap between two seeds, according to the defined minimum and maximum intron length. Figure 5 shows this process in action, exemplifying how the seed chains are obtained from a set of seed occurrences. Some seeds are at invalid distances, others are not in the correct order, and the three final ones form the largest chain in the proper position.

When checking the seed occurrences' end positions for formerly identified splice sites (case *(b)* of chain formation conditions), if no seed occurrence is detected at the opposite position of the intron, the read is directly matched against the reference to check if a seed could be constructed there, and if yes, a new seed occurrence is created and added to the corresponding array. The chains are then checked and we take the ones whose combined seed length is the largest, i.e. with the highest value for $(m_{i_j} + \ldots + m_{i_{j+h}})$. This is done because not all the seeds might be present in each chain, for example, due to multiple sequential mismatch errors, and we are interested in the regions with as much similarity as possible. If one or more seeds are missing, a semi-global alignment [17] is done by using banded dynamic programming to align that missing area in the read and the corresponding area in the genome. Based on the final number of errors of each chain, the corresponding read position is accepted or discarded. If a read is correctly aligned and it contains newly detected splice sites (case *(c)* of chain formation conditions), which involves valid intron lengths and verified splice signals, they are stored and taken into account for the following reads.

Fig. 6. Position of the splice junctions relatively to the found seeds. Splice signals can be found (1) exactly after the ends of the seeds, (2) shifted to the left, or (3) shifted to the right.

3.3. *Splice Junction Detection*

For splice junction detection we consider the splice signals "GT-AG", "GC-AG" and "AT-AC", their reverse complements, and some others non-canonicals [6]. When the distance that separates two consecutive seeds in a chain is in the range of an intron size (defined by the user or the default values - from 50bp to 500000bp), we look for splice evidence in the genome. For this, we look for the two splice signals in both the first and the last positions of the gap in the genome. If they both occur, the existence of an intron is later reported in the SAM format with the code "*N*", as well as its length, corresponding to the size of the gap between those two seeds.

When looking for splice signals, three cases must be considered (Figure 6), since the splice junctions can take place in the position where the gap in the genome starts and ends, or a few positions to the left, or a few positions to the right. The last two cases can happen when the seeds did not end at exactly the edge of the exons, but instead one of the seeds was extended beyond the limit of its exon, causing the other seed to start with a small "delay" after the beginning of its exon. This can happen when the characters immediately after the end of one exon match exactly the first characters of the opposite exon.

4. Discussion

Our approach brings many advantages that are crucial for a good performance. The first advantage, with great impact in the running time, is the fact that this approach analyses the set of reads only once, unlike other tools for transcriptome alignment, which have two phases and re-process the reads that were not initially aligned. Other major aspect is that, like some of the most competitive tools in this area, SpliceTAPyR does not require any splice junctions model, any annotations or any junctions library. This allows it to discover new splice sites without any bias. Unlike others tools, SpliceTAPyR has yet another advantage in what concerns the maximum length of read match in the genome. Because of the way seeds are formed, there is no limitation in the size of the match between the read and the reference, enabling the alignment of reads of any length without size constraints. Also, if a read has a full alignment on the genome with no errors, only one seed is generated.

For the experimental evaluation we choose to compare SpliceTAPyR with three other RNA alignment software tools – TopHat2, MapSplice2 and STAR – that

rely on different approaches, but are considered among the most competitive in the area. As can be seen in the next subsections, SpliceTAPyR presents the best relation between required time and memory while maintaining an exceptionally high amount of aligned reads.

4.1. *Other Computational Approaches*

In the last few years, TopHat [20], MapSplice [21] and STAR [7], became some of the most used tools for RNA-Seq alignment [8, 2], but they are very demanding with respect to time and/or memory. Another possible drawback is the fact that TopHat and MapSplice are dependent of external packages. They use Bowtie as a proxy for a first mapping phase, which is also based on a FM-Index indexing structure. But Bowtie is highly restrictive in some aspects, e.g. it does not allow reads with more than 1024 bp. For the alignment of reads containing more than one exon, they use different heuristic approaches for the initially unmapped reads. For every splicing junction found, TopHat searches the unmapped reads to find the ones that span the junction by a few bases on each side using a seed-and-extend strategy. MapSplice also fragments the reads and searches for anchors that span an exon. Then it performs a double-anchored spliced alignment if a match is found. STAR performs the mapping in only one phase, but it uses an uncompressed SA as an index, which is characterized by a high memory consumption. It also uses variable length seeds, called "Maximum Mappable Prefixes", which are somewhat similar to SpliceTAPyR's own seeds, but since their matching is done from left to right, it can result in different seeds.

4.2. *Experimental Evaluation*

For all datasets, SpliceTAPyR was run with the default parameters: minimum intron size 20, maximum intron size 500000, percentage of read identity 90%. We tunned each one of the other tools, whenever possible, to also run with these characteristics. Lastly, we processed each alignment file (in the standard SAM format) with all the results of every tool, to obtain not only the percentage of aligned/mapped reads, but also the statistics of errors and the proportions of splice signals found.

4.2.1. *Datasets*

The used datasets were taken from publicly available real sequencing experiments and from computationally simulated reads. The first real dataset, "1M_100bp" (Table 1) was downloaded from the MapSplice website `http://www.netlab.uky.edu/p/bioinfo/MapSpliceManual` and has 999991 reads. The second real dataset, "SRR534289" (Table 2), was downloaded from SRA database. Because this dataset has reads with size above 1024bp, we had to crop the reads' size so it could be used as input for TopHat and MapSplice, due to a limitation in read lengths in these tools. Furthermore, we only used the first one million reads because these two tools

Software	Time (s)	Memory (KB)	% Reads Aligned	bp/error	splice events
SpliceTAPyR	19	287 168	99.17	544	382 491
STAR	36	4 544 636	99.84	316	350 746
TopHat2	635	659 608	99.69	762	407 630
MapSplice2	459	3 346 444	99.91	700	388 779

Table 1. 1M_100bp statistics.

Software	Time (s)	Memory (KB)	% Reads Aligned	bp/error	splice events
SpliceTAPyR	85	6 687 376	94.87	94	249 763
STAR	99	28 954 396	87.05	87	211 659
TopHat2	2732	5 855 832	89.10	231	221 350
MapSplice2	5589	5 767 152	79.68	156	244 686

Table 2. SRR534289 statistics.

Software	Time (s)	Memory (KB)	% Reads Aligned	bp/error	splice events
SpliceTAPyR	618	8 542 688	98.81	160	1 315 064
STAR	657	54 301 676	96.07	144	1 063 664
TopHat2	12 150	7 251 688	95.70	196	1 473 653
MapSplice2	31 193	15 013 740	98.01	176	1 366 489

Table 3. Human_T1R1 statistics.

require a very long running time. The third dataset, "Human_T1R1" (Table 3) was obtained from the benchmarking evaluation done in [2] and contains ten million reads.

4.3. *Performance and Alignment Evaluation*

For each tool, the reference sequence indexes were previously generated. The first dataset is from the chromosome 20 genome reference and the other two from the entire human genome. For each dataset we evaluated time, memory, percentage of reads aligned, rate of correctly aligned base pairs per alignment error (bp/error, the higher the better) and number of detected splice events among all the aligned reads. Because some tools allow the alignment of some fragments/seeds of a read even when not finding an alignment for all of them, instead of reporting the read as not aligned, such reads still appear in the output SAM file through *clipping*, meaning that the segment is "present, but not aligned". Since this would not be a fair comparison against tools that found a fully-aligned correct position of the read, all aligned reads through clipping events were ignored from the presented statistics.

For all datasets, SpliceTAPyR presents the best overall performance. It is faster than competing tools, while it demands less memory and the percentage of reads aligned is similar, if not higher than that reported by the other tools. We observe

that there is a great gain in what concerns to the capacity of aligning a large amount of RNA-Seq reads in significantly less time, as is the case for dataset "Human_T1R1" (Table 3), since the alignment is done in a single phase compared to the double-phase approach used in MapSplice and TopHat. Comparing with STAR, the second fastest tool, SpliceTAPyR also requires significantly less memory in all the conducted experiments, since it relies on a memory-efficient FM-Index, instead a more memory-heavy SA, as in the former. We also observe that the rate of base pairs per error is admissible and that the number of splicing events is similar to the others.

5. Final Remarks

With the rapid growth of the amount of data generated by sequencing technologies, it is expected that alignment tools for RNA-Seq data can keep up in what concerns throughput and performance. We need tools that can align millions of reads in the shortest time possible not putting at risk the quality of the alignment. SpliceTAPyR provides a new alternative tool for RNA-Seq, able to align reads from any sequencing technology, demanding considerably less time and a small amount of RAM memory, while competing fairly in the quality of the alignment with the most popular tools. This allows it to analyze experiments in a simple desktop or laptop computer, without having to resort to expensive server-like machines.

SpliceTAPyR has the additional advantages of not depending on any external packages or their maintenance, neither on genome annotations. Each read is processed only once, unlike similar tools, giving it an advantage in computation time. Its efficient FM-Index data structure also helps to keep a low memory footprint. The required user-defined parameterization is inexistent or very easy. It does not have the constraint of building a fixed number of seeds/fragments or with a fixed specific size, since the number and the length of its dynamic seeds are defined spontaneously by the construction algorithm: if a read has a complete match on the genome with no errors, only one seed spanning the entire read is generated.

SpliceTAPyR is available as free open source software under the GNU GPL v3 license, and it can be downloaded from http://github.com/fjdf/SpliceTAPyR.

Acknowledgments

This work has been supported by *Fundação para a Ciência e Tecnologia*, through the doctoral grant of Universidade de Lisboa, postdoctoral grant SFRH/BPD/111836/2015 and also by FCT Projects TUBITAK/0004/2014, PTDC/EEI-SII/1937/2014, and UID/CEC/50021/2013.

References

[1] M. Adams, J. Kelley, J. Gocayne, M. Dubnick, M. Polymeropoulos, H. Xiao, C. Merril, A. Wu, B. Olde, R. Moreno and a. et, Complementary dna sequencing: expressed sequence tags and human genome project, *Science* **252**(5013) (1991) 1651–1656.

[2] G. Baruzzo, K. E. Hayer, E. J. Kim, B. Di Camillo, G. A. FitzGerald and G. R. Grant, Simulation-based comprehensive benchmarking of RNA-seq aligners, *Nature Methods* (2016).

[3] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell *et al.*, Accurate whole human genome sequencing using reversible terminator chemistry, *nature* **456**(7218) (2008) 53–59.

[4] P. O. Brown and D. Botstein, Exploring the new world of the genome with dna microarrays, *Nature genetics* **21** (1999) 33–37.

[5] M. Burrows and D. Wheeler, A block-sorting lossless data compression algorithm, *DIGITAL SRC RESEARCH REPORT*, Citeseer (1994).

[6] M. Burset, I. Seledtsov and V. Solovyev, Analysis of canonical and non-canonical splice sites in mammalian genomes, *Nucleic acids research* **28**(21) (2000) 4364–4375.

[7] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson and T. R. Gingeras, Star: ultrafast universal rna-seq aligner, *Bioinformatics* **29**(1) (2013) 15–21.

[8] P. G. Engström, T. Steijger, B. Sipos, G. R. Grant, A. Kahles, G. Rätsch, N. Goldman, T. J. Hubbard, J. Harrow, R. Guigó *et al.*, Systematic evaluation of spliced alignment programs for RNA-seq data, *Nature methods* **10**(12) (2013) 1185–1191.

[9] F. Fernandes, P. G. da Fonseca, L. M. Russo, A. L. Oliveira and A. T. Freitas, Efficient alignment of pyrosequencing reads for re-sequencing applications, *BMC bioinformatics* **12**(1) (2011) p. 163.

[10] P. Ferragina and G. Manzini, Opportunistic data structures with applications, *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, IEEE (2000), pp. 390–398.

[11] P. Ferragina, G. Manzini, V. Mäkinen and G. Navarro, Compressed representations of sequences and full-text indexes, *ACM Transactions on Algorithms (TALG)* **3**(2) (2007) p. 20.

[12] N. A. Fonseca, J. Rung, A. Brazma and J. C. Marioni, Tools for mapping high-throughput sequencing data, *Bioinformatics* (2012) p. bts605.

[13] B. Langmead, C. Trapnell, M. Pop and S. L. Salzberg, Ultrafast and memory-efficient alignment of short dna sequences to the human genome, *Genome biology* **10**(3) (2009) p. R25.

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

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

[16] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y.-J. Chen, Z. Chen *et al.*, Genome sequencing in microfabricated high-density picolitre reactors, *Nature* **437**(7057) (2005) 376–380.

[17] S. B. Needleman and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *Journal of molecular biology* **48**(3) (1970) 443–453.

[18] J. M. Rothberg, W. Hinz, T. M. Rearick, J. Schultz, W. Mileski, M. Davey, J. H. Leamon, K. Johnson, M. J. Milgrew, M. Edwards *et al.*, An integrated semiconductor device enabling non-optical genome sequencing, *Nature* **475**(7356) (2011) 348–352.

[19] F. Sanger, S. Nicklen and A. Coulson, DNA sequencing with chain-terminating inhibitors, **74**(12) (1977) 5463–5467.

[20] C. Trapnell, L. Pachter and S. L. Salzberg, Tophat: discovering splice junctions with rna-seq, *Bioinformatics* **25**(9) (2009) 1105–1111.

[21] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou *et al.*, Mapsplice: accurate mapping of rna-seq reads for splice junction discovery, *Nucleic acids research* (2010) p. gkq622.

[22] Z. Wang, M. Gerstein and M. Snyder, Rna-seq: a revolutionary tool for transcriptomics, *Nature reviews genetics* **10**(1) (2009) 57–63.