## 1 RNA-seq Problem Definition

Consider a reference genome $G$ and a RNA-seq read $R$. We aim to align the read $R$ onto the genome $G$ allowing multiple junctions. Precisely, we partition $R$ into segments $R_0, \ldots, R_p$ and align $R_j$ without gap and satisfy the following condition, for $j = 0, \ldots, p$, to $G[i_j..i_j + |R_j| - 1]$.

1. For each segment $R_j$ where $0 < j < p$, $|R_j| \geq \lambda$. **(How about the first and last segments?)**
2. The distance between any two consecutive segments $R_j$ and $R_{j+1}$ given by $d_j = i_{j+1} - i_j - |R_j|$ satisfies $D_1 < d_j < D_2$.
3. In the alignment, there exist at most $k$ mismatches for any $l(< \lambda)$ consecutive bases from $R$.

Our aim is to find an alignment of $R$ satisfying the above criteria which minimizes the number of junctions, then minimizes the number of mismatches.

## 2 Algorithms

### 2.1 Single Junctions

Let $G$ be a reference genome and $R$ any RNA-seq read where $|R| > 2l$. Let $H(S_1, S_2)$ denote the Hamming distance between two strings $S_1$ and $S_2$. This section describes an algorithm $Find\_Single\_Junc(R, G, D_1, D_2, l, k)$ which solves the RNA-seq problem assuming there is exactly one junction in $R$. For each junction it finds, it will report the start and end points of the intron contained in the junction, along with the corresponding position of the junction in $R$.

The algorithm has two steps. Step 1 tries to find all subsequences in $G$ whose $l$-mer prefix and $l$-mer suffix differ by at most $k$ mismatches from the $l$-mer prefix and $l$-mer suffix of $R$ respectively, and of sufficient length to contain a junction. For each such read found, Step 2 extends the $l$-mer prefix to the right and the $l$-mer suffix to the left to identify the junction point. Figure **??** describes the algorithm for $Find\_Single\_Junc$. Lines 1-2 perform Step 1. Line 3 will perform Step 2. Lines 4-7 will report the information about the location of the junction in $R$ and $G$.

## 3 Find All Junctions

This section discusses the problem of mapping a RNA-seq read $R$ onto a reference genome $G$ without assuming the number of junction is one. The basic idea is to find the junctions in the read $R$ from left to right one by one. The algorithm is based on the following lemmas.

**Lemma 3.1.** *Consider a read $R$ which aligns to the genome $G$ according to the criteria given in the problem definition. There are three possible cases for junctions occurring within $R[a, a + l - 1]$. They are,*

```
$Find\_Single\_Junc(R, G, D_1, D_2, l, k)$
1: $P_l(R)$ = $l$-mer prefix of $R$, $S_l(R)$ = $l$-mer suffix of $R$;
2: $X$={substrings $R'$ of $G$ with prefix and suffix having at most $l$-mismatches
   with $P_l(R)$ and $S_l(R)$ respectively and $D_1 \geq |R'| - |R| \geq D_2$.}
3: $Y$= $R', r$ in $X$ that minimise $H(P_r(R), P_r(R')) + H(S_{|R|-r}(R), S_{|R|-r}(R'))$;
4: for each $(R', r)$ in $Y$ do
5:     find genomic location $x$ of $R'$.
6:     report $(x + r, x + |R'| - |R| - r - 1, r)$.
7: end for
```

Figure 1: Algorithm *Find_Single_Junc*.

1. *There are no junctions in* $R[a, a + l - 1]$.
2. *There is exactly one junction in* $R[a, a + l - 1]$*, and* $R[a + l, \min(a + 2l - 1, |R|)]$ *has no junction.*
3. *There are two junctions, the first of which is in* $R[a, a + l - 1]$ *and the second is in* $R[a + l, \min(a + 2l - 1, |R|)]$.

*Proof.* Note that for cases 2 and 3, if there is a junction inside $R[a, a + l - 1]$, then $|R| > a + l$ since the right half of the read must contain at least $l$ bases. If there are more possibilities, we would need have more than one junction inside $R[a, a + l - 1]$ or $R[a + l, \min(a + 2l - 1, |R|)]$. i.e. at least one exon would be completely contained in one of these segments. Since we have the restriction of the minimum size of an exon being $\lambda > l$, this would be impossible. Note that if $\lambda > 2l$, case 3 is not possible. □

We will next give another lemma that will be useful later.

**Lemma 3.2.** *Let there be a single junction in* $R[x, x + \lambda - l - 1]$*, where* $|R| \geq x + \lambda$*. Then the* $l$ *bases* $[x + \lambda - l, x + \lambda - 1]$*, allowing* $k$ *mismatches, is part of an exon to the right of the junction.*

*Proof.* If the junction starts at $x$, then $R[x, x + \lambda - 1]$ allowing $k$ mismatches is part of an exon. If the junction occurs at $x + \lambda - l - 1]$, $R[x + \lambda - l, \min(x + 2\lambda - 1, |R|)]$, allowing $k$ mismatches, will be a part of an exon to the right of the junction. Therefore, their intersection given in the lemma is part of an exon for any junction occurring between $R[x, x + \lambda - l - 1]$. □

The algorithm *Find_All_Junc*$(R, \Psi)$ will return all possible transcripts of $R$ having minimum number of junctions satisfying the problem definition. First, it will trim $R[1, i]$ from the $R$ until $R[i, i + l - 1]$ can be mapped allowing $k$ mismatches. This is done to skip a junction that might occur in the first $l - 1$ bases. Rather than trimming base by base, it uses Lemma **??** to skip by $\lambda - l$ bases in this interval to narrow down the junction. It calls the routine *Seek_Junc* to enumerate all possible transcripts of $R[i, |R|]$ having the least number of junctions.

Finally, if transcripts are found, *Remove_Duplicates* will remove transcripts that are duplicated.

*Seek_Junc* returns putative transcripts for $R$ and the minimum number of junctions in a transcript in $Max\_Junc$. It is a recursive algorithm that searches $l$ or $2l$ bases of a read at a time greedily for the least number of junctions. If a transcript for $R$ with $N$ junctions is successfully found at some point, all subsequent searches will look for a transcript having at most $N$ junctions. This way it is able to find the transcripts with the least number of junctions.

Each transcript is represented as a set of junctions $J_1, J_2, \ldots J_{Max\_Junc}$. A Junction $J_i$ is a represented as a 3-tuple $(p, q, r)$. $(p, q)$ are the leftmost and the rightmost genomic locations of the intron between the junction. $r$ is the length of the portion of exon to the left of the junction present in $R$. Each junction $J_i$ is preceded by $J_k$ for $0 \leq k < i$. For each valid transcript, $J_{Max\_Junc}$ is one of the two special markers. We use the marker *End_Transcript* to signal the end of a transcript, and the marker *dummy_junction* to specify a possible unknown junction at the end of the read.

$Seek\_Junc(S, a, L, j)$ takes a prefix $S$ of $R$, and three numbers $a, L$ and $j$ as arguments. $S[1, a]$ will be part of a putative exon. $L$ is the number of putative exons found in the current partial transcript preceding $S[1, a]$. If $L \neq 0$, $j$ will contain the genomic location of the start of the exon corresponding to $S[1, a]$.

The routine will first check if there are less than $l$ bases to be processed. If this is the case, and a junction has already been found ($L > 0$) we try to extend the junction allowing $k$ mismatches. If the extension is successful, we mark it as the end of transcript. If the extension fails, it could be due to an unknown junction, and we add a dummy junction. (Lines 2-9).

Otherwise, a greedy search is performed to find junctions. The greedy search is based on the three cases given in Lemma **??**. As shown in figure 1, We first try to extend a known junction or the $a$-mer prefix of $S$ right by at most $l$ bases allowing $k$ mismatches(Lines 11 and 16 respectively).

Next we consider case 2, where a single junction will be added. We first check if the addition of a single junction will not exceed the minimum junction count (Line 21). As shown in figure 2 (i) we find substrings in $G$ where $S[1, a]$ can be anchored as a prefix, and $S[a + l, \min(a + 2l, |S|)]$ can be anchored as a suffix, and use *Find_Single_Junc* to deduce junctions between $S[a, a + l - 1]$ (Line 22). If a junction is found, its location is verified to see if it is consistent with the last junction found and is checked to see if it satisfies the requirement of minimum exon size(Line 23). If the junction is consistent, $S$ is trimmed near the junctions, and *Seek_Junc* is called recursively to search for the remaining junctions on the trimmed read.

For case 3, two junctions will be added. We first check if the addition of a two junction will not exceed the minimum junction count (Line 30). We will try to anchor the $a$-mer prefix of $S$ and find an $l$-mer in $S[a, a+2l-\lambda]$ as the second anchor. We find junctions between these two and check for their consistency as in case 2, and when a consistent junction is found, trim $S$ and recursively apply $Seek\_Junc$(Lines 32-38).

Next according to cases 2 and 3, a search is made to find a single junctions in the current $l$ bases. We need to handle case 2 only if $L$ is less than $Min\_Junc$, and case 3 only if $L+1$ is less than $Min\_Junc$ as otherwise the resulting transcript will have more than $Min\_Junc$ junctions. We will be identifying these junctions with $Find\_Single\_Junc$. To identify a junction with $Find\_Single\_Junc$ the junction must have at least $l$ bases to its right and left. Line 15 checks for the first condition. To find Junctions in case 1, $Find\_Single\_Junc$ will anchor $S[a-l, a]$ and $S[a+l, a+2l-]$ (or $S[|S|-l, |S|]$ if the rest of $|S|$ is too short) and find the junctions between them (Line 17). If the start of the current exon is known, the resulting junctions are checked to see if they lie inside $S$ in a correct location (Line 18). If junctions are found satisfying these conditions it is stored in $T[L]$, $Seek\_Jinc$ is called, setting the next new exon to start at the rightmost base of the junction, both in the read and the genome(Line 19-20).

Finally, a search is made for junctions due to small exons given by case 3. Note that for $\lambda > 2l$ this part of the code is not necessary. Our idea is to check each interval $S[a, a+i]$ for a junction, where $i = 0, .., 2l-\lambda$(or $i = 0, .., |S|-\lambda$ if the string is too short). According to the Lemma **??**, if there is a junction at $j \in [p, p+\lambda-l]$, then the $l$ bases starting at $p+\lambda-l$ must be part of an exon. Therefore, we scan the $l$ bases at points in the interval $[a, a+2l-\lambda]$ at steps of $\lambda-l$ allowing $k$ mismatches. If a match is found at point $j$ we use it as the left anchor, and use $R[j+\lambda-2l, j+\lambda-l]$ as the left anchor in $Find\_Single\_Junc$ (Line 25-27). If junctions are found, their consistency is checked and $Seek\_Junc$ is called on the new putative exon discovered as done in case 2.

## 4 Proof of correctness.

We will prove that given a Read $R$ with transcripts satisfying the problem definition, $Seek\_Junc$ will find those with the minimum number of junctions.

**Lemma 4.1.** *Suppose a read $R$ can be partitioned in to single junctions satisfying the problem definition, and that there are $n$ such possible partitions. $Seek\_Junc(R, 1, \Psi, X, L, Transcripts, T, Min\_Juncs, Trans\_Count)$ will find all these single junctions.*

*Proof.* Let $R[1, x]$ and $R[x+1, |R|]$ be the exons involved some partitioning of $R$ into a single junction. Then $R[1, l\lfloor x/l \rfloor]$ and $R[x+1, |R|]$ can be mapped allowing at most $k$ mismatches per $l$ bases. During the recursive calling of $Seek\_Junc$ in Lines 8-10, when $a = l\lfloor x/l \rfloor$, the junction $x$ will be found under case 2 or 3 by Lemma **??**. Then, call to $Seek\_Junc$ in Lines 20 or 30 will repeatedly execute Lines 1-10 until the end of $|R|$ is reached, and a transcript is signalled. i.e. All the single junctions of $R$ can be found. $\square$

**Lemma 4.2.** *Suppose a read $R$ can be partitioned into $N$ junctions satisfying the problem definition, and that $N$ is the least number of such junctions that $R$ can be partitioned into. Then, $Seek\_Junc(R, 1, \Psi, X, 0, Transcripts, T, Min\_Juncs, Trans\_Count)$ will find all the junctions in all such partitions.*

*Proof.* We prove by induction. By Lemma **??** the claim is true for $N = 1$. Let the claim be true for $N = n$. Let $R$ have a partition containing a minimum of $n+1$ junctions. Let $R[1, x]$, $R[x+1, |R|]$ contain the $n+1^{th}$ junction. From the induction hypothesis,$Seek\_Junc(R[1, x], 1, \Psi, X, 0, Transcripts, T, Min\_Juncs, Trans\_Count)$ will find all the $n$ junctions preceding the last junction. We note that the junctions are found in Lines 17 and 27, and if $S$ is replaced by another string having $S$ as a prefix, the algorithm will still find all the junctions associated with $S$. i.e. Replacing $R[1, x]$ with $R$ will still find the same $n$ junctions, and will replace the $S$ in calls to $Seek\_Junc$ in lines 20 and 30 with $R[x - |S| - r, |R|]$. From Lemma **??** this would find the last junction. From the principle of mathematical induction, we can see that the claim is true for all $N$. $\square$

## 5 Implementation

In the practical implementation we set $k = 1$ and $l = 18$. This will closely match MapSplice's choice of partitioning a read into 25bp regions and allowing one mismatch in each. In human RefSeq there are only about 400 exons of length 18. About 1.7% of the exons have a size less than 36bp. It might be better to set $L = 25$, which will miss less than 1% of the exons while making scanning faster. As the majority of reads will have one or two junctions, the algorithm should initially search for these cases. When searching junctions canonical junctions should be given priority.

```
$Find\_All\_Junc(R)$
Require: Global variables $G, l, k, D_1, D_2, \lambda, Max\_Junc$.
 1: $Max\_Juncs = \infty$;
 2: $Transcripts = \emptyset$;
 3: for $r = 1, .., l-1$ step $\lambda - l$ do
 4:    $Transcripts = Seek\_Junc(R[r, |R|], 1, 0, 0)$;
 5:    if $Transcripts \neq \emptyset$ then
 6:       $Remove\_Duplicates(Transcripts)$
 7:       return $Transcripts$;
 8:    end if
 9:    return $Transcripts$;
10: end for
```

```
$Seek\_Junc(S, a, L, j)$
Require: Global variables $G, l, k, D_1, D_2, \lambda, Max\_Junc$.
 1: $T = \emptyset$;
 2: if $L > 0$ then
 3:    if $a > |S| - l$ then
 4:       $Max\_Junc = L$;
 5:       if $S[a+1, |S|]$ can be aligned with $G[j+a, j+|S|-1]$ allowing $k$
          mismatches then
 6:          return $\{End\_Transcript\}$;
 7:       else
 8:          return $\{dummy\_junction\}$;
 9:       end if
10:    else
11:       if $G[j+a, j+a+l-1]$ can be aligned with $S[a+1, a+l]$ allowing $k$
          mismatches with then
12:          $T = Seek\_Junc(S, a+l, L, j)$;
13:       end if
14:    end if
15: else
16:    if $S[1, a+l-1]$ occurs in $G$ allowing $k$-mismatches per $l$ bases then
17:       $T = Seek\_Junc(S, a+l, L, j)$;
18:    end if
19: end if
20: if $a > l$ then
21:    if Case 2 and $L < Max\_Junc$ then
22:       for each single junction $(p, q, r)$ between $S[a, a+l]$ do
23:          if $L = 0$ or $(p = j + r$ and $r \geq \lambda)$ then
24:             $T_1 = Seek\_Junc(S[r+1, |S|], 1, L+1, q)$;
25:             if $T_1 \neq \emptyset$ add $(p, q, r)$ to each partial transcript in $T_1$;
26:             $T = T \cup T_1$;
27:          end if
28:       end for
29:    end if
30:    if Case 3 and $L + 1 < Max\_Junc$ then
31:       for $i = a$ to $a + 2l - \lambda$ do
32:          for each junction $(p, q, r)$ between $S[i, i+l]$ do
33:             if $L = 0$ or $(p = j + r$ and $r \geq \lambda)$ then
34:                $T_1 = Seek\_Junc(S[r+1, |S|], 1, L+1, q)$;
35:                if $T_1 \neq \emptyset$ add $(p, q, r)$ to each partial transcript in $T_1$;
36:                $T = T \cup T_1$;
37:             end if
38:          end for
39:       end for
40:    end if
41: end if
42: return $T$
```