

Appendix

Problem 1

Our entry data are :

- segment matches for pairwise splice alignment for a set of genes and cdss
- gene_ids and their cds_ids

We currently have the exons positions in the match, we need to find the cds position in the paired aligned gene. Let's take this line as example :

cds0_ppan ptro 222 396 618 1230 1452 1.00

In this example, the cds.id : cds0_ppan, the start position is 396 and the end position 618, those are the position in the cds, we need to find the position in the gene of the cds (in our case, find the position in the gene of cds0_ppan).

How to find the corresponding position of a cds

When a CDS ID matches with its own gene, we know that the positions of the gene on the line correspond to those of the CDS in the gene. Therefore, when we are looking for the position of a CDS within its gene, we simply need to look for that CDS with the corresponding positions in its parent gene and retrieve the gene start and end positions.

Problem 2 : implement the refinement of segment matches

The algorithm we are implementing is described in a paper titled Segment-based multiple sequence alignment in section 2.2.

As entry data, we have :

- a set of non-repeating segment matches

The segment match refinement algorithm computes a minimal subdivision of the segments, i.e. it refines the segment matches such that all parts of all segment matches can be used [1].

Algorithm 1 Retrieve the position of the aligned exon in the other gene of the alignment

```
1: Input:
2:   data: segment matches for pairwise splice alignment for a set of genes and
   cdss
3:   source_to_target_file_path: gene_ids and their cds_ids
4: Output:
5:   processed: A dictionary with additional exon position information added
   to the alignment data
   FindExonPosInPairedGenedata, source_to_target_file_path
6: cds_gene_map  $\leftarrow$  get_cds_gene_map(source_to_target_file_path)
7: result  $\leftarrow$  {}
8: for each key, values in data do
9:   for each match in values do
10:    cdsId  $\leftarrow$  match[0]
11:    geneb  $\leftarrow$  match[1]
12:    genea  $\leftarrow$  cds_gene_map[cdsId]
13:    search_key  $\leftarrow$  concatenate(cdsId, genea)
14:    if genea  $\neq$  geneb then
15:      result  $\leftarrow$  search_exon_pos_in_pair_aligned_gene(data[search_key], [match[3], match[4]], line_length =
        10)
16:      if length(result) == 1 then
17:        append(result[0][5], result[0][6]) to match
18:        formatted_match  $\leftarrow$  iteration_result_to_refinement_entry_format_file(match, cds_gene_map)

19:        append(formatted_match) to processed['all']
20:      else
21:        append("-", "-") to match
22:      end if
23:    else
24:      append("-", "-") to match
25:    end if
26:  end for
27: end for
28:
29: return processed
```

Algorithm 2 Search Exon Positions in a Pair of Aligned Genes

Input:

data: List of lines to search through

cds_segment: Tuple containing the start and end positions of the exon

line_length: Minimum length of the line to consider (default is 8)

Output:

result: List of lines that match the search criteria based on the exon positions

search_exon_pos_in_pair_aligned_genedata, *cds_segment*, *line_length* = 8

(*cds_start*, *cds_end*) \leftarrow *cds_segment*

cds_start \leftarrow int(*cds_start*)

cds_end \leftarrow int(*cds_end*)

result \leftarrow []

for each *line* **in** *data* **do**

if (int(*line*[3]) == exonStart \wedge int(*line*[4]) == exonEnd) **then**

 append *line* to *result*

end if

end for

return *result*

Algorithm 3 mult_seg_match_refinement

Require: file_path: string

Ensure: refined multi-segment matches

segment_matches \leftarrow original_data[2:]

Vi \leftarrow build_Vi(segment_matches)

del_duplicates_sort_Vi(*Vi*)

visualize_Vi(*Vi*, output)

Ti \leftarrow build_Ti(segment_matches)

for all match **in** segment_matches **do**

 boundaries \leftarrow get_match_boundaries(match)

for all w **in** boundaries **do**

 gene1 \leftarrow gene_of_boundary(index, match)

Vi \leftarrow refine(w, *Ti*, *Vi*, gene1)

end for

end for

del_duplicates_sort_Vi(*Vi*)

visualize_Vi(*Vi*, output, True)

print Refinement output path : output

return *Vi*

Algorithm 4 refine

Require: $w, Ti, Vi, gene1$ **Ensure:** Refined Vi

```
1:  $stack \leftarrow [(w, gene1)]$ 
2: while  $stack \neq []$  do
3:    $(current\_w, current\_gene1) \leftarrow stack.pop()$ 
4:    $overlaps \leftarrow get\_overlaps(Ti, current\_w, current\_gene1)$ 
5:   for all  $lq \in overlaps$  do
6:     if  $lq[0].data == current\_gene1["gene\_id"]$  then
7:        $(u, v, u\_v\_gene) \leftarrow (lq[0].begin, lq[0].end, lq[0].data)$ 
8:        $(x, y, x\_y\_gene) \leftarrow (lq[1].begin, lq[1].end, lq[1].data)$ 
9:     else
10:       $(u, v, u\_v\_gene) \leftarrow (lq[1].begin, lq[1].end, lq[1].data)$ 
11:       $(x, y, x\_y\_gene) \leftarrow (lq[0].begin, lq[0].end, lq[0].data)$ 
12:    end if
13:     $h \leftarrow x + (current\_w - u)$ 
14:     $gene2\_id \leftarrow x\_y\_gene$ 
15:     $consider\_h \leftarrow x < h < y$ 
16:     $gene2 \leftarrow \{"gene\_id" : gene2\_id, "u" : u, "v" : v\}$ 
17:    if  $consider\_h$  then
18:      if  $gene2\_id \notin Vi$  then
19:         $Vi[gene2\_id] \leftarrow []$ 
20:      end if
21:      if  $h \notin Vi[gene2\_id]$  then
22:         $Vi[gene2\_id].append(h)$ 
23:         $stack.append((h, gene2))$ 
24:      end if
25:    end if
26:  end for
27: end while
28:
29: return  $Vi$ 
```

Problem 3 : alignment graph

The algorithm we are implementing is described in the same paper in section 2.1. The purpose of this algo is to construct an alignment graph from refined segment matches for genes.

Overall steps of the algorithm

1. Build vertices from the refined segments for each gene. A vertex is a tuple size 3:
 - 0 gene id
 - 1 start position
 - 2 length
2. Build edges from vertices.
 - (a) An edge exists if there is a match between the 2 vertices.
 - $\diamond \text{start}_i == \text{start}_j$
 - $\diamond \text{length}_i == \text{length}_j$
 - (b) The weight of the edge is found using the percent identity (PI) of the match represented by the edge. The triplet approach is the way:
 - i. Compute edge percent identity.
 - ii. Calculate the weight (triplet approach).
3. Generate the graph using the edges.
 - The edge structure (tuple):
 - 0 Vertex origin
 - 1 Vertex target
 - 2 Weight

Algorithm 5 build_alignment_graph

Require: Refined segment matches for genes

Ensure: Alignment graph

```
vertices  $\leftarrow$  build_vertices(Vi)  
edges  $\leftarrow$  build_edges(vertices)  
for all vertex in vertices do  
    Get weighted edges using vertices  
    Get trace from weighted edges  
end for  
build_graph
```

Algorithm 6 build_vertices

Require: Vi **Ensure:** $vertices$

```
1:  $vertices \leftarrow []$ 
2: for all  $(gene\_id, boundaries) \in Vi$  do
3:   for  $index \leftarrow 0$  to  $len(boundaries) - 2$  do
4:      $boundary \leftarrow boundaries[index]$ 
5:      $next\_boundary \leftarrow boundaries[index + 1]$ 
6:      $start\_boundary \leftarrow \text{int}(boundary)$ 
7:      $segment\_length \leftarrow \text{int}(next\_boundary) - start\_boundary$ 
8:      $vertex \leftarrow (gene\_id, start\_boundary, segment\_length)$ 
9:      $vertices.append(vertex)$ 
10:  end for
11: end for
12:
13: return  $vertices$ 
```

References

- [1] Tobias Rausch, Anne-Katrin Emde, David Weese, Andreas Döring, Cedric Notredame, and Knut Reinert. Segment-based multiple sequence alignment. *Bioinformatics*, 24(16):i187–i192, 08 2008.

Algorithm 7 build_edges

Require: *vertices*, *target_data***Ensure:** *edges*

```
edges ← []
weightless_edges ← []
for  $i \leftarrow 0$  to  $\text{len}(\text{vertices}) - 1$  do
    vertex_i ← vertices[i]
    vertex_matches ← get_vertex_matches(vertex_i, vertices)
    for all vertex_j ∈ vertex_matches do
        percent_identity ← compute_percent_identity(vertex_i, vertex_j, target_data)

        weightless_edges.append((vertex_i, vertex_j, percent_identity))
    end for
end for
for all weightless_edge ∈ weightless_edges do
    (origin_vertex, target_vertex, percent_identity) ← weightless_edge
    origin_matches ← get_vertex_matches(origin_vertex, vertices)
    to_sum ← []
    for all match ∈ origin_matches do
        (gene_id, start_pos, length) ← match
        if gene_id ≠ target_vertex[0] then
            triplet_1_pi ← search_edge(weightless_edges, origin_vertex, match)[2]

            triplet_2_pi ← search_edge(weightless_edges, match, target_vertex)[2]

            pi ← min(triplet_1_pi, triplet_2_pi)
            to_sum.append(pi)
        end if
    end for
    edge_weight ← round(percent_identity + sum(to_sum), 2)
    edges.append((origin_vertex, target_vertex, edge_weight))
end for

return edges
```
