UNIVERSITÉ DE
SHERBROOKE

Multiple Spliced Alignment : A Consistency Based Approach

AUTHOR

Chakirou ALABANI (Group H)

BIN702 - Autumn 2024
Nadia Tahiri

Contents

1 Project description

Alternative splicing is now recognized as a fundamental process in eukaryotic organisms, enabling a single gene to generate multiple distinct transcripts. This mechanism affects the majority of human genes (*[Harrow et al., 2006, Tress et al., 2007, Kim et al., 2008, Wang et al., 2008, Chen and Manley, 2009]*) and is widely pervasive, playing a crucial role in enhancing the diversity of the proteome. Recent genome-wide studies indicate that 40–60% of human genes have alternatively spliced forms (*[Modrek and Lee, 2002]*). This extensive occurrence underscores the need to investigate the evolutionary and conservation patterns of transcript sets, as these insights are essential for a deeper understanding of the mechanisms that influence gene evolution.

Understanding the evolution of sets of alternative transcripts is a challenging task and requires automated methods and tools to compare sets of alternative transcripts from homologous genes. Alternative transcripts from homologous genes have traditionally been compared using pairwise spliced alignments (PSpAs). A PSpA aligns either a spliced RNA sequence or its DNA equivalent, the coding DNA sequence (CDS), with an unspliced DNA sequence. This approach helps identify homologous or corresponding exons between sequences, providing crucial insights for genome annotation and gene prediction (*[Stanke et al., 2006, Dunne and Kelly, 2018]*). Various methods have been developed to tackle different versions of the PSpA problem, which involves finding the best PSpA between two sequences based on a specific optimization function (see *[Jammali et al., 2019]*). However, PSpA is limited to comparing only two sequences at a time, making it unsuitable for examining the evolution of alternative splicing. This restriction also renders it ineffective for analyzing large databases, where multiple sequence comparisons are essential.

A logical extension of pairwise spliced alignment (PSpA) for studying the evolution of alternative spliced RNA sets is multiple spliced alignment (MSpA). This approach aligns a collection of spliced RNA sequences with their corresponding unspliced genomic sequences, allowing for detailed analysis of splicing and exon structures within the gene sequences. Unlike traditional multiple sequence alignment (MSA), which focuses solely on sequence similarity, MSpA incorporates the splicing and exonic architectures of the input genes. Just as MSA has greatly advanced our understanding of sequence evolution, MSpA is anticipated to reveal new insights into the evolution of alternative splicing and the relationships among alternative spliced RNA sets.

The MSpA framework also has practical applications for genome annotation by facilitating the identification of exons homologous to those in well-characterized species, thereby aiding in the prediction of conserved isoforms in newly annotated genomes.

In the current state of art, there are a few methods available for coomputong MSpAs, with SplicedFamAlignMulti (SFAM) being a leading approach (*[Jammali et al., 2022]*). SFAM include three extensions : SFAM_mblock, SFAM_tcoffee_p, and SFAM_tcoffee_m, each providing tailored functionalities for different alignment scenarios. A summary of SFAM methods mechanisms is shown in Figure 1.

In this report, we present SplicedFamAlignMultiCBA (SFAM_CBA), a greedy heuristic method developed to merge all pairwise spliced alignments (PSpAs) of known coding sequences (CDSs) and

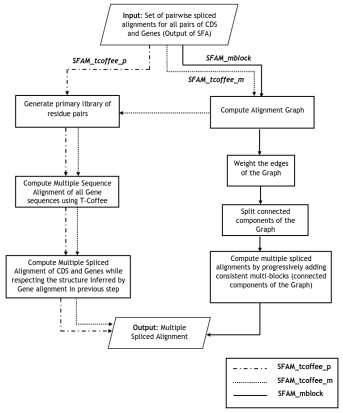

Figure 1: overview SFAM methods

gene sequences within a gene family into a multiple spliced alignment (MSpA).

2 Materials and methods

2.1 Characterization of the MSpA Problem

2.1.1 Pairwise Spliced Alignment (PSpA)

A Pairwise Spliced Alignment (PSpA) refers to the alignment of a Coding Sequence (CDS) with a gene sequence while accounting for the splicing structure. This alignment aims to identify homologous exon sequences and is formulated as a chain of blocks, where each block corresponds to pairwise alignments of segments from both the CDS and gene.

The significance of PSpA lies in its ability to highlight macroscopic alignments at the splicing level (exon intron structure) rather than at the nucleotide level, emphasizing differences in splicing trends and exon usage across gene families.

Definition: A PSpA consists of blocks, which are conserved segments where both the CDS and gene are included in the alignment. If a CDS is aligned with a gene in a block, it is termed a *conserved block*. Conversely, if only the CDS is included, it is referred to as a *deleted block*.

2.1.2 Multiple Spliced Alignment (MSpA)

An MSpA extends the concept of PSpA to include multiple CDSs and gene sequences, facilitating the identification of homologous exons across a broader set of sequences.

Definition: An MSpA of a set of CDSs C and genes G is represented as a chain of multiblocks $A = \{A[1], \ldots, A[n]\}$. Each multiblock $A[i]$ includes a key set key$(A[i])$, which is a subset of $C \cup G$,

mapping each sequence to its respective start and end locations (s_i^x, e_i^x).

The MSpA must satisfy several conditions:

- Each multiblock must contain at least one element.
- Segments from the same sequence must not overlap across multiblocks.
- The segments induced by the MSpA must fully cover the original CDSs.
- Segments from aligned genes must be consistent within the same multiblock.

2.1.3 Multiple spliced alignment problem

Input: A set of CDSs C; a set of genes G, a set of PSpAs $X = \{X_{c,g} \mid (c, g) \in C \times G\}$ for all pairs in $C \times G$.

Output: An MSpA A of C on G that maximizes the sum of the scores of induced PSpAs:

$$\sum_{(c,g)\in C\times G} S(A_{c,g})$$

(*Jammali et al. [2022]*).

Several methods are available for computing PSpAs between a gene and a CDS (*Jammali et al. [2019]*).

2.2 The SFAM-CBA algorithm

We now present our algorithm for constructing an MSpA for a set of CDSs C and a set of genes G, given a set of PSpAs $X = \{X_{c,g} \mid (c, g) \in C \times G\}$ for all pairs in $C \times G$ (the MSAPproblem). MSA methods commonly employ greedy heuristics, with the progressive alignment strategy (*Feng and Doolittle [1987]*) being one of the most widely used approaches. Distinctively, our method incorporates both the consistency approach and the triplet approach outlined in *CÃ©dric Notredame [2000]* for pairwise scoring. The consistency-based strategy evaluates how well the alignment of two residues or segments in a given pairwise alignment is supported by other precomputed pairwise alignments. Applying this consistency-based approach within a progressive alignment allows for the integration of information from all pairwise alignments at each step, helping to overcome the typical limitations associated with progressive alignment.

2.2.1 Boundaries refinement

Boundary refinement is a crucial step in the consistency based approch for MSpA, aiming to handle overlapping segment matches effectively. The refinement algorithm extends the principles established by *Halpern et al. (2002)*, ensuring that all parts of the original segment matches can be utilized.

As described in *Feng and Doolittle [1987]*, let $M = \{M_0, M_1, \ldots, M_{m-1}\}$ represent the set of segment matches, where each match $M_k = (S_i^{uv}, S_j^{xy})$ consists of segments S_i^{uv} from sequence S_i and S_j^{xy} from sequence S_j. The goal is to refine these segment matches into a set of submatches $M^* = \{M_0^*, M_1^*, \ldots, M_{m'-1}^*\}$ that cover the original matches.

In this context, the refinement process involves ensuring that the set of submatches M^* satisfies the conditions of tiling the original matches:

$$[u, v-1] = \bigcup_{M'_k \in M'_*} [u', v'-1] \quad \text{and} \quad [x, y-1] = \bigcup_{M'_k \in M'_*} [x', y'-1].$$

This ensures that each original match is fully covered by the refined submatches in M^*.

A resolved set of matches, denoted as R, is desired, where all segments are either disjoint or identical. In such a set, any $(S_i^{uv}, S_j^{xy}) \in R$ satisfies:

$$[u, v] \cap \text{supp}_S^i(R) = \{u, v\} \quad \text{and} \quad [x, y] \cap \text{supp}_S^j(R) = \{x, y\}.$$

The algorithm proceeds to process segment matches sequentially, building a node set V_i for each sequence S_i, initialized to the support set of the segments. By recursively identifying boundary positions and ensuring necessary cuts are made, the algorithm guarantees a minimum cardinality refinement without introducing superfluous cuts.

2.2.2 Computation of pairwise scores

The computation of pairwise scores of the set of PSpAs $X = \{X_{c,g} \mid (c, g) \in C \times G\}$ is vital for quantifying the similarity between aligned segments of sequences. This process relies on the refined segment matches obtained in the previous step.

Given a set of refined matches M^*, the pairwise score S_{ij} between sequences S_i and S_j can be calculated as follows:

$$S_{ij} = \sum_k \text{idty}(M_k^*),$$

where $\text{idty}(M_k^*)$ represents the identity score of the match M_k^* that covers the segment (S_i^{uv}, S_j^{xy}). The identity score can be derived from established scoring matrices adjusted for the context of the segments being compared.

To incorporate information from multiple sequences, a weight system can be applied, similar to the one discussed by Notredame et al. (1998). By considering intermediate sequences that support the alignment, the weights associated with each residue pair are aggregated, enhancing the reliability of the pairwise scores.

The overall weight W for a pair of aligned residues is computed by examining triplets of sequences and accumulating scores based on their alignments. The final weight reflects both the similarity of the sequences and their consistency across the dataset, facilitating a more robust alignment process.

2.2.3 MSpA Computation

The final stage involves the computation of the multiple sequence alignment (MSA) itself. This process is executed through a recursive algorithm that utilizes the refined pairwise scores obtained in the previous steps. The goal is to construct an alignment that maximizes the overall score while adhering to the constraints imposed by the segment matches.

Let T represent the phylogenetic tree that organizes the sequences based on their evolutionary relationships. The MSpA can be computed recursively as follows:

$$C(T_i) = \text{Align}(C(T_{\text{left}}), C(T_{\text{right}}), S),$$

where $C(T_i)$ is the column list of aligned sequences at node T_i and S is the pairwise score matrix derived from the previous computations.

Each internal node of the tree represents an alignment of its child nodes. The alignment function Align combines the column lists from the left and right child nodes, optimizing the alignment based on the accumulated pairwise scores:

$$\text{Align}(C(T_{\text{left}}), C(T_{\text{right}}), S) = \max_{\text{alignment}} \sum_{\text{pairs}} S_{ij},$$

where the sum is taken over all pairs of aligned segments, and the maximum is computed over all possible alignments.

This recursive approach allows the algorithm to build the MSpA incrementally, ensuring that the final alignment is optimal with respect to the defined scoring system. The use of a refined weight system ensures that the alignment reflects not only the local similarities but also the global relationships among all sequences, thereby enhancing the quality and reliability of the MSpA.

A comprehensive and detailed pseudocode of each step in the algorithm is provided in the appendix. The implementation of the algorithm, along with the data used, is available on GitHub at *https://github.com/chakirouAlabani/BIN702/tree/main/code*.

2.3 Experimental setup

2.3.1 Dataset

To assess the performance and demonstrate the utility of SFAM, we used three datasets consisting of 20 simulated gene families, as well as a dataset of 20 real gene families from the Ensembl database (*Zerbino et al. [2018]*).

2.3.1.1 Real dataset We selected 20 gene families from the Ensembl database, each containing genes from six amniote species: human, mouse, dog, dingo, cow, and chicken. For each family, only the genes from these six species were retained, and the dataset includes their sequences and all corresponding CDSs. CDS sets for each gene family were obtained from Ensembl releases 97 and 98, as

release 98 introduced an updated gene annotation for dog, resulting in an increase from 19,857 to 20,257 coding genes and from 39,074 to 60,994 transcripts. This dataset was used to assess the ability of SpliceFamAlignMulti to predict the new dog CDSs added in release 98, based on spliced alignments calculated from release 97 data. Table 1 (*Jammali et al. [2022]*) provides a detailed description of this real dataset.

Gene family (tree) ID	No. of total genes	No. of total CDS (9798)	No. of dog genes	No. of dog CDS (9798)	PID
ENSGT00390000063205	19	4245	3	36	0.098
ENSGT00390000008371	12	2728	2	23	0.604
ENSGT00390000000715	6	1215	1	14	0.474
ENSGT00940000157909	6	3135	1	15	0.623
ENSGT00530000063187	17	2830	3	35	0.177
ENSGT00950000182978	24	88103	4	419	0.106
ENSGT00950000182681	43	7989	7	919	0.147
ENSGT00950000182875	37	3367	6	410	0.112
ENSGT00950000182728	35	7583	6	614	0.073
ENSGT00950000182931	24	4657	4	48	0.168
ENSGT00950000182705	39	111115	5	59	0.087
ENSGT00390000063023	23	4146	4	611	0.103
ENSGT00940000153241	14	1619	3	36	0.06
ENSGT00950000182956	23	5671	4	520	0.257
ENSGT00950000182783	29	7277	4	49	0.045
ENSGT00950000183192	8	6972	3	36	0.051
ENSGT00950000182727	24	7384	5	617	0.047
ENSGT00390000004965	7	78	1	12	0.89
ENSGT00390000003967	6	912	1	12	0.792
ENSGT00390000005532	6	1213	1	12	0.545

Table 1: Description of the real dataset of 20 gene families (PID)

2.3.1.2 Simulated data Since true MSpAs of real gene families were unavailable for validation, we generated three datasets of 20 simulated gene families with gene sequences and CDSs using SimSpliceEvol (Kuitche et al., 2019). SimSpliceEvol simulates the evolution of gene splicing structures and transcript sets through alternative splicing events. Starting with a gene tree with evolutionary rates represented by branch lengths, it uses distributions from 189 eukaryotic species to model exon and intron lengths, alternative transcript numbers, and exon counts per gene.

The simulation begins with an ancestral gene sequence with exon intron structure and associated

CDSs at the root of the tree, which evolves along the branches to produce gene sequences and CDSs for each node. The model simulates two levels of evolution: gene level changes, such as exon duplication, gain, loss, and sequence mutations in exonic (coding) and intronic (noncoding) regions, and transcript-level changes, including the creation, loss, and modification of isoforms via alternative splicing.

We generated three datasets: (1) "Small" with low evolutionary rates on a tree of five primates, (2) "Medium" with moderate rates using a tree of five primates and rodents, and (3) "Large" with high rates using a tree of five amniote species. The gene trees used in these simulations, obtained from the Ensembl Compara database (Zerbino et al., 2018), are described in Supplementary Material. SimSpliceEvol also provides the true MSA of all CDS and gene sequences and identifies groups of splicing orthologs. These groups consist of CDSs that evolved from a common ancestral CDS without alternative splicing events. Table 2 (*Jammali et al. [2022]*) provides further details on the simulated datasets.

	Small	**Medium**	**Large**
Number of families	20	20	20
Number of genes per family	5	5	5
Avg. number of CDS	8.8	8.55	9.35
	(3.02)	(3.04)	(2.70)
Avg. CDS length	802.65	820.49	778.13
	(204.41)	(288.37)	(281.77)
Avg. gene length	2358.71	2724.82	2418.22
	(713.76)	(813.09)	(601.44)
Avg. pairwise PID (%)	71.24	54.39	41.61
	(4.36)	(2.52)	(2.4)

Table 2: Summary of simulated datasets.

2.3.2 Evaluated methods

To the best of our knowledge, SFAM-CBA has no direct competitors for MSpA, aside from the original SFAM method. Therefore, the performance of SFAM-CBA algorithms will be compared against the three SFAM extensions (SFAM_mblock, SFAM_tcoffee_p, and SFAM_tcoffee_m) as well as other popular multiple sequence alignment (MSA) methods, using the resulting multiple CDS alignments. This comparison is relevant because the quality of a multiple CDS alignment induced by an MSpA is closely linked to the overall quality of the MSpA itself.

MSA methods SFAM-CBA was compared with widely used MSA methods, including *MUSCLE* [Edgar, 2004], *MAFFT* [Katoh and Standley, 2013], *MACSE* [Ranwez et al., 2018], *CLUSTAL_O* [Sievers et al., 2011], *PRANK* [Löytynoja and Goldman, 2005, 2008], *T-Coffee* [Notredame et al.,

2000], and *Mirage* [Nord et al., 2018]. *MUSCLE* uses a log-expectation scoring system to accelerate its progressive alignment process. *MAFFT* identifies homologous regions using the fast Fourier transform to improve alignment efficiency. *MACSE* is specifically designed for coding sequences and accounts for the codon structure, making it ideal for aligning protein-coding nucleotide sequences. *CLUSTAL_O* utilizes guide trees and a profile-profile comparison based on hidden Markov models to generate MSAs. *PRANK* is a probabilistic alignment tool that uses maximum likelihood estimation, incorporating evolutionary distances between sequences. *T-Coffee* employs a consistency-based progressive alignment strategy, leveraging a library of pairwise residue matches obtained from initial pairwise alignments. *Mirage* focuses on aligning alternatively spliced protein isoforms by mapping protein sequences to their genomic counterparts and aligning based on these mappings.

2.3.3 Performance metrics

Based on the true MSAs of the CDSs for each gene family in the simulated datasets, we calculated the precision, recall, F-score, and computation time for each method and each gene family. *Precision* is defined as the proportion of nucleotide pairs in the predicted alignment that also appear in the true alignment. *Recall* represents the proportion of nucleotide pairs in the true alignment that are also found in the predicted alignment. The *F-score* is the harmonic mean of precision and recall, providing a balanced measure of both.

To account for the fact that SFAM-CBA and Mirage use the gene sequences as guides to enhance the alignment of CDSs, we evaluated two sets of results. The first set was obtained by aligning only the CDSs directly with other MSA methods (MUSCLE, MAFFT, MACSE, CLUSTAL_O, PRANK, T-Coffee). The second set of results was was produced by aligning both gene sequences and CDSs with these MSA tools (referred to as *MUSCLE_G*, *MAFFT_G*, *MACSE_G*, *CLUSTAL_O_G*, *PRANK_G*, *T-Coffee_G*), and then extracting the CDS alignments from the full gene MSAs. For both sets of results, we computed precision, recall, F-score, and computing time. For *SFAM-CBA*, the total computing time includes the time required for generating pairwise spliced alignments (PSpAs).

List of Figures

List of Tables

References

M. Chen and J. L. Manley. Mechanisms of alternative splicing regulation: insights from molecular and genomics approaches. *Nature Reviews Molecular Cell Biology*, 10:741–754, 2009.

Jaap Heringa CÃ©dric Notredame, Desmond G. Higgins. T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302:205–217, 2000. doi: 10.1006/jmbi.2000.4042.

M. P. Dunne and S. Kelly. OMGene: mutual improvement of gene models through optimisation of evolutionary conservation. *BMC Genomics*, 19:307, 2018.

R.C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004. doi: 10.1093/nar/gkh340.

D.-F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25:351–360, 1987.

J. Harrow, F. Denoeud, A. Frankish, et al. Gencode: producing a reference annotation for encode. *Genome Biology*, 7 Suppl 1:S4.1–S9, 2006.

S. Jammali et al. Splicedfamalign: Cds-to-gene spliced alignment and identification of transcript orthology groups. *BMC Bioinformatics*, 20:133, 2019.

Safa Jammali, AbigaÃ¯l Djossou, Wend-Yam D D OuÃ©draogo, Yannis Nevers, Ibrahim Chegrane, and AÃ¯da Ouangraoua. From pairwise to multiple spliced alignment. *Bioinformatics Advances*, 2 (1):vbab044, 2022. doi: 10.1093/bioadv/vbab044.

K. Katoh and D.M. Standley. Mafft multiple sequence alignment software version 7: improvements in performance and usability. *Molecular Biology and Evolution*, 30(4):772–780, 2013. doi: 10.1093/molbev/mst010.

E. Kim, A. Goren, and G. Ast. Alternative splicing: current perspectives. *BioEssays*, 30:38–47, 2008.

A. Löytynoja and N. Goldman. An algorithm for progressive multiple alignment of sequences with insertions. *Proceedings of the National Academy of Sciences of the United States of America*, 102 (30):10557–10562, 2005. doi: 10.1073/pnas.0409137102.

A. Löytynoja and N. Goldman. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, 320(5883):1632–1635, 2008. doi: 10.1126/science.1158395.

B. Modrek and C. Lee. A genomic view of alternative splicing. *Nature Genetics*, 30(1):13–19, Jan 2002. doi: 10.1038/ng0102-13.

A.L. Nord, M.L. DeMarche, C.M. McCullough, and F. Zanini. Mirage: a fast and flexible aligner for alternatively spliced protein isoforms. *Bioinformatics*, 34(15):2667–2669, 2018. doi: 10.1093/bioinformatics/bty151.

C. Notredame, D.G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, 2000. doi: 10.1006/jmbi.2000.4042.

V. Ranwez, E. Douzery, C. Cambon, N. Chantret, and F. Delsuc. Macse v2: toolkit for the alignment of coding sequences accounting for frameshifts and stop codons. *Molecular Biology and Evolution*, 35(10):2582–2584, 2018. doi: 10.1093/molbev/msy159.

F. Sievers, A. Wilm, D.G. Dineen, T.J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Soding, J.D. Thompson, and D.G. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Molecular Systems Biology*, 7:539, 2011. doi: 10.1038/msb.2011.75.

M. Stanke et al. Augustus at egasp: using est, protein and genomic alignments for improved gene prediction in the human genome. *Genome Biology*, 7:S11, 2006.

M. L. Tress, P. L. Martelli, A. Frankish, et al. The implications of alternative splicing in the encode protein complement. *Proceedings of the National Academy of Sciences of the United States of America*, 104:5495–5500, 2007.

E. T. Wang, R. Sandberg, S. Luo, I. Khrebtukova, L. Zhang, C. Mayr, S. F. Kingsmore, G. P. Schroth, and C. B. Burge. Alternative isoform regulation in human tissue transcriptomes. *Nature*, 456:470–476, 2008.

D. R. Zerbino et al. Ensembl 2018. *Nucleic Acids Research*, 46:D754–D761, 2018.

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 hind 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.

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

Input:
  data: segment matches for pairwise splice alignment for a set of genes and cdss
  source_to_target_file_path: gene_ids and their cds_ids
Output:
  processed: A dictionary with additional exon position information added to the alignment data
FindExonPosInPairedGene$data, source_to_target_file_path$
$cds\_gene\_map \leftarrow$ get_cds_gene_map($source\_to\_target\_file\_path$)
$result \leftarrow \{[]\}$
for each $key, values$ **in** $data$ **do**
  for each $match$ **in** $values$ **do**
    $cdsId \leftarrow match[0]$
    $gene_b \leftarrow match[1]$
    $gene_a \leftarrow cds\_gene\_map[cdsId]$
    $search\_key \leftarrow$ concatenate($cdsId, gene_a$)
    if $gene_a \neq gene_b$ **then**
      $result \leftarrow$ search_exon_pos_in_pair_aligned_gene($data[search\_key], [match[3], match[4]]$, line_length = 10)
      if length($result$) == 1 **then**
        append(result[0][5], result[0][6]) to match
        $formatted\_match \leftarrow$ iteration_result_to_refinement_entry_format_file($match, cds\_gene\_map$)
        append(formatted_match) to processed['all']
      else
        append("-", "-") to match
      end if
    else
      append("-", "-") to match
    end if
  end for
end for

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_gene$data, cds_segment, line_length = 8$
$(cds\_start, cds\_end) \leftarrow cds\_segment$
$cds\_start \leftarrow \text{int}(cds\_start)$
$cds\_end \leftarrow \text{int}(cds\_end)$
$result \leftarrow []$
for each *line* **in** *data* **do**
  if $(\text{int}(line[3]) == \text{exonStart} \wedge \text{int}(line[4]) == \text{exonEnd})$ **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 ← `original_data[2:]`
Vi ← `build_Vi(segment_matches)`
`del_duplicates_sort_Vi(Vi)`
`visualize_Vi(Vi, output)`
Ti ← `build_Ti(segment_matches)`
for all match in segment_matches **do**
  boundaries ← `get_match_boundaries(match)`
  for all w in boundaries **do**
    gene1 ← `gene_of_boundary(index, match)`
    Vi ← `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

$stack \leftarrow [(w, gene1)]$
while $stack \neq []$ **do**
  $(current\_w, current\_gene1) \leftarrow stack.pop()$
  $overlaps \leftarrow get\_overlaps(Ti, current\_w, current\_gene1)$
  for all $lq \in overlaps$ **do**
    if $lq[0].data == current\_gene1$["gene_id"] **then**
      $(u, v, u\_v\_gene) \leftarrow (lq[0].begin, lq[0].end, lq[0].data)$
      $(x, y, x\_y\_gene) \leftarrow (lq[1].begin, lq[1].end, lq[1].data)$
    else
      $(u, v, u\_v\_gene) \leftarrow (lq[1].begin, lq[1].end, lq[1].data)$
      $(x, y, x\_y\_gene) \leftarrow (lq[0].begin, lq[0].end, lq[0].data)$
    end if
    $h \leftarrow x + (current\_w - u)$
    $gene2\_id \leftarrow x\_y\_gene$
    $consider\_h \leftarrow x < h < y$
    $gene2 \leftarrow$ {"gene_id" : $gene2\_id$, "u" : u, "v" : h}
    if $consider\_h$ **then**
      if $gene2\_id \notin Vi$ **then**
        $Vi[gene2\_id] \leftarrow []$
      end if
      if $h \notin Vi[gene2\_id]$ **then**
        $Vi[gene2\_id].append(h)$
        $stack.append((h, gene2))$
      end if
    end if
  end for
end while

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.
      - $\diamond$ start_i == start_j
      - $\diamond$ length_i == 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
  $vertices \leftarrow build\_vertices(V_i)$
  $edges \leftarrow build\_edges(vertices)$
  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: $vertices$

$vertices \leftarrow []$
for all $(gene\_id, boundaries) \in Vi$ **do**
  for $index \leftarrow 0$ **to** $len(boundaries) - 2$ **do**
    $boundary \leftarrow boundaries[index]$
    $next\_boundary \leftarrow boundaries[index + 1]$
    $start\_boundary \leftarrow \text{int}(boundary)$
    $segment\_length \leftarrow \text{int}(next\_boundary) - start\_boundary$
    $vertex \leftarrow (gene\_id, start\_boundary, segment\_length)$
    $vertices.append(vertex)$
  end for
end for

return $vertices$

Algorithm 7 build_edges

Require: $vertices$, $target\_data$
Ensure: $edges$

```
edges ← []
weightless_edges ← []
for i ← 0 to len(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
```