EXACT ALGORITHMS FOR COMPUTING PAIRWISE ALIGNMENTS AND 3-MEDIANS FROM STRUCTURE-ANNOTATED SEQUENCES (EXTENDED ABSTRACT)

P.A. EVANS

Faculty of Computer Science, University of New Brunswick, P.O. Box 4400, Fredericton, NB, E3B 5A3, Canada

H.T. WAREHAM

Department of Computer Science, Memorial University of Newfoundland, St. John's, NF, A1B 3X5, Canada

Given the problem of mutation saturation in ancient molecular sequences, there is great interest in inferring phylogenies from higher-order types of molecular data that change more slowly, such as genomic organization and the secondary and tertiary structures of ribosomal RNA and proteins. In this paper, we define edit distances based on two representations of RNA secondary structure, arc annotation and hierarchical string annotation, and give algorithms for computing these distances on pairs of annotated sequences, aligning pairs of annotated sequences, and computing 3-median annotated sequences from triples of annotated sequences. The 3-median algorithms can be used as part of a well-known iterative heuristic for inferring phylogenies. All given algorithms are adapted from algorithms for computing longest common annotated subsequences of pairs of annotated sequences.

1 Introduction

An *evolutionary tree* (or *phylogeny*) shows the set of speciation events by which a set of given species arose from a common ancestor. Molecular sequences are frequently used as data to infer such trees; however, accumulated sequence mutations invariably obscure evidence of common ancestry for species that diverged more than several hundred million years ago. Other types of molecular data that change more slowly, such as genomic organization and the secondary and tertiary structures of ribosomal RNA and proteins, are thus of interest in inferring such "deep" relationships.

Ideally, one would like to infer both the tree and information about hypothetical ancestors corresponding to internal vertices in that tree from given data relative to some measure of edit distance that approximates the evolutionary divergence between pairs of species. A first step in this direction is to infer internal-vertex information relative to a given tree (the *tree alignment* problem).[15] Unfortunately, even this problem is known to be NP-complete (and thus unlikely to be solvable in polynomial time) for simple edit distance on sequences when the tree is a star, *e.g.*, all given sequences are leaves and there is one internal vertex.[4] One way of dealing with this is to break up a tree into stars based on triples of species (*3-medians*) and optimally compute the internal-vertex sequences associated with such 3-medians in an iterative fashion until a (possibly locally) optimal tree is found.[14] However, though 3-medians can be computed in $O(n^3)$ time for sequences relative to simple edit distance,[13] computing 3-medians relative to many distances based on genomic organization is NP-complete (as is, in some cases, computation of that distance between pairs of species) (see DasGupta *et al*[1] and references).

In this paper, we define edit distances based on two representations of RNA secondary structure, arc annotation and hierarchical string annotation, and give algorithms for computing these distances on pairs of annotated sequences, aligning pairs of annotated sequences, and computing 3-median annotated sequences from triples of annotated sequences. The algorithms for computing these distances are adapted from algorithms given by Evans [2,3] for computing longest common annotated subsequences of pairs of annotated sequences as modified relative to classical algorithms for pairwise [9] and three-way [8] sequence alignment and tree alignment.[13] Conservation of sequence annotation (and hence structure) is important because sequence function depends on sequence structure; hence, a structural change with little change in the primary sequence is more significant than a large change in the primary sequence that still produces a similar structure. While the edit distance algorithms given here all use weights, they weight structural information more heavily because this information cannot be ignored.

Related Work: Sankoff [12] gave an algorithm that simultaneously aligns a given set of RNA sequences, constructs a phylogeny for those sequences, and reconstructs optimal secondary structures for both given and reconstructed sequences in that tree; unfortunately, it is not obvious how this algorithm can be adapted to work with given secondary-structure assignments. Various distance measures on RNA secondary structures have been proposed (see Moulton *et al*[7] and references); however, though many of these measures have been used to compare groups of or align pairs of structure-annotated sequences, none have been applied to infer phylogeny.

Background: An *arc-annotated sequence* (S, P) is a sequence S over an alphabet Σ and an arc-annotation $P \subset \{1, \ldots, |S|\}^2$. Such arcs are commonly used to represent pairwise molecular bonds in DNA, RNA, and protein sequences. Within such an annotation, two arcs (i_1, i_2) and (i_3, i_4) *cross* if $i_3 \leq i_2$, and *nest* if $i_1 \leq i_3 \leq i_4 \leq i_2$. There are many possible restrictions on arc annotation;[3] in this paper, we restrict P such that for all (i_1, i_2) and $(i_3, i_4) \in P$, $i_1 = i_3$ if and only if $i_2 = i_4$, and $i_2 \neq i_3$, *i.e.*, each sequence position can be an endpoint for at most one arc and is thus linked to at most one other sequence position. Note that arcs may still cross in such annotations, and hence can encode pseudoknots, an important RNA configuration that is seldom handled by known algorithms (see Lyngsø and Pedersen [6] and references).

An alternative to representing RNA structure as a collection of individual bonds is to assign labels to specific substrings of the sequence that have a particular structure. Such a representation is used in the European Ribosomal RNA Databases.[10,11] Since substrings can contain other substrings, the resulting hierarchy of layers is a *substring-annotated sequence* representation that is similar to the rooted ordered tree representation discussed in Zhang and Sasha,[16] though it supports different operations.

All weight-matrices $w(x,y)$ considered in this paper are semimetrics, *i.e.*, for all objects x, y in the matrix set, $w(x,y) \geq 0$, $w(x,y) = 0$ if and only if $x = y$, and $w(x,y) = w(y,x)$.

2 Arc Annotated Sequences

2.1 Pairwise Alignment with Arc Structure

Given two arc-annotated sequences (S_1, P_1) and (S_2, P_2) over an alphabet Σ such that $|S_1| = |S_2| = n$, a symbol edit weight matrix $w(x,y)$ for $x, y \in \Sigma$, a symbol indel penalty w_s, and an arc mismatch penalty w_a, the *arc-preserving edit distance* between (S_1, P_1) and (S_2, P_2) is the minimum cost over all sequences of operations that align (S_1, P_1) and (S_2, P_2). The arc mismatch penalty is incurred when positions i_1 and i_2 of S_1 are aligned with j_1 and j_2 of S_2, respectively, and either $(i_1, i_2) \in P_1$ or $(j_1, j_2) \in P_2$ but not both, *i.e.*, the arc is not preserved.

This edit distance is based on the problem of computing the longest common arc-preserving subsequence of a given pair of arc-annotated sequences. This problem is known to be NP-complete in general; [2,3] however, there is a special-case algorithm [2,3] that runs in $O(9^k n^2)$ time, where where n is the sequence length and k is an upper bound on the *cutwidth* of the arc structure, *i.e.*, the maximum number of arcs that cross any part of the sequence.

Theorem 2.1 *The minimum arc-preserving edit distance between two arc-annotated sequences can be computed in $O(9^k n^2)$ time.*

Proof: The required algorithm is given in Table 1. This algorithm keeps track of the optimal subset of preserved arcs by representing all possible subsets of arcs implicitly in $2^k \times 2^k = 4^k$ dynamic-programming tables, and by activating computations in and merging results computed within these tables as necessary over the course of the algorithm's execution.

Table 1: An Algorithm for Computing Arc-Preserving Edit Distance.

1. Partition each of P_1 and P_2 into k sets, where each set contains a chain of arcs that do not cross or nest. Number these chains 0 through $k-1$.

2. For each of (S_1, P_1) and (S_2, P_2), for each subset of the set of chains P_i, create a copy of sequence S_i with the initial endpoints of all arcs in those chains removed and replaced by a space. The set of sequences thus created is $\mathcal{S}_i$, and is generally indexed by h_i, where $h_i = \sum_{j \in subset} 2^j$.

3. For each combination of h_1 and h_2, create a two-dimensional table, $n \times n$, that uses strings $\mathcal{S}_1[h_1]$ and $\mathcal{S}_2[h_2]$. Each table position includes both a value $T^{(h_1,h_2)}[i,j]$, the minimum edit distance so far, and a tree $M^{(h_1,h_2)}[i,j]$ of the initial arc endpoints matched along the computation paths that produce that value. These matches between initial endpoints are tentative assignments that will be checked when the final endpoints of the arcs are encountered.

4. Calculate the longest common arc-preserving subsequence of (S_1, P_1) and (S_2, P_2) by traversing the tables in n^2 steps: Starting with cell $[1,1]$ in table $T^{(0,0)}$, at each step, compute the values of the corresponding cells in all active tables, where a table is *active* if one arc from each of the chains in its associated subsets is being considered for preservation.

5. $\mathbf{return}(T^{(0,0)}[n,n])$

The computations in step (4) of the algorithm are somewhat complex and require more explanation. During the table traversal, the trees associated with cells of these tables are merged and manipulated, requiring that the tree data structure support the following operations:

merge: is applied to a finite list of trees and their corresponding edit distance values, and returns the merge of those trees that have the minimum corresponding values. When the trees are merged, they are copied and also simplified from the root down by uniting identical children of the same parent node.

test: looks for a given arc assignment pair in the tree; returns *true* if it is found, *false* otherwise.

prune: given a tree and an arc assignment pair, removes all paths that do not contain the pair, and then removes the pair itself.

trim: given a tree, an arc number k', and a flag value, removes all nodes in the tree that contain an arc assignment that involves an arc with that number k'. This operation checks either the i or j values, depending on the value of the flag.

extend: given a tree and an arc assignment pair, add the pair to the tree as its new root.

The basic recurrence describing the edit distance calculation is

$$\begin{aligned} T[i,j] &= \min(T[i-1,j]+w_s, T[i,j-1]+w_s, \\ &\qquad T[i-1,j-1]+w(S_1[i],S_2[j])) \end{aligned} \tag{1}$$

This calculation is varied when arc endpoints are encountered as follows:

1. When an initial arc endpoint is encountered, both tables relative to the table being considered that include that arc's chain in its subset are activated and initialized by copying over needed values into the preceding row or column in each table.

2. If a pair of initial endpoints is encountered, one from each sequence, the algorithm attempts to match their arcs. Four tables relative to the table being considered (essentially a pair of instances of (1) corresponding to the two initial endpoints) are activated and initialized as in (1), but that table which has both initial endpoints requires the tree at that position to be *extended* by adding that arc assignment pair.

3. When a final arc endpoint is encountered, the table without the initial endpoint is calculated normally. The other table, where the initial endpoint was allowed to align with a non-space symbol, is calculated with an arc penalty if the final endpoint is also aligned with a non-space symbol, *i.e.*, the third term of the recurrence becomes $T[i-1,j-1]+w(S_1[i],S_2[j]+w_a$. These two tables are then *merged* to find the minimum, and their trees are *trimmed* to remove all assignments that use that arc.

4. If a pair of final endpoints is encountered, the algorithm uses *test* to determine if the corresponding pair of initial endpoints are in the tree. If they are, and the minimum value is produced by matching the final endpoints (term $T[i-1,j-1]+w(S_1[i],S_2[j])$, those endpoints are matched and the tree is *pruned.* Otherwise, as the endpoints are not in the tree or they do not produce the minimum value, the tables are computed by adding the appropriate arc penalties to this term, *i.e.*, if the table allowed both initial endpoints to match a non-space symbol, $2w_a$ is added; else, if it allowed only one initial endpoint to match a non-space symbol, w_a is added. Finally, the appropriate trees and tables are *merged* as in (3).

Each of the tree operations except *extend* runs in time proportional to the size of the tree, and *extend* runs in constant time. The algorithm initializes and subsequently modifies at most $4^k n^2$ table entries, where the size of the tree associated with each cell is bounded by the number of active arcs in that cell's table. For any of the n^2 table positions, the trees at that position (over all tables) have a total of

$$\sum_{r=0}^{k}\sum_{s=0}^{k}\binom{k}{r}\binom{k}{s}\binom{r+s}{s} \quad \leq \quad 9^k \tag{2}$$

associated endpoint entries;[2,3] hence, the algorithm as a whole runs in $O(9^k n^2)$ time. ∎

Corollary 2.2 *An optimal alignment of two arc-annotated sequences can be computed in $O(9^k n^2)$ time.*

Proof: The distance computation in Theorem 2.1 can be modified to include pointers for each table cell that point to the cell whose value was used to generate the minimum distance at that point. Cells that include the endpoints of arcs also need to note if the arc was used, and if a pair of arcs were matched then they should also indicate which path in the arc assignment tree was used. An optimal alignment can then be found by tracing back from $T^{(0,0)}[n,n]$ through these pointers. If a cell points to an adjacent (but nondiagonal) cell in the same table, a space is added to the alignment. If a cell points to a cell at the same position in a different table, no space is added. If an arc match is encountered, then the path through the tree (to the matched initial endpoints) is used to determine the path through the intermediate cells. This requires the comparison of assignment trees, which can be done at each intermediate cell in $O(9^k)$ time. Since this comparison operation will always make progress in determining an alignment position, it needs to be done at most $2n$ times. Thus the alignment can be extracted in $O(9^k n)$ time, and the algorithm as a whole runs in $O(9^k n^2)$ time. ∎

2.2 Three Sequence Median with Arc Structure

Theorem 2.3 *The minimum arc-preserving 3-median distance of a triple of arc-annotated sequences can be computed in $O(9^k n^3)$ time.*

Proof: The algorithm in Theorem 2.1 can be modified as follows:

- All three sequences and arcs are processed in the initial steps, and are indexed by i, j, and g.
- The edit distance is computed using 8^k three-dimensional tables of size $n \times n \times n$, in which each cell stores the 3-median edit distance. Moreover, as each tree entry can now be an assignment between a pair of arcs or between three arcs, each cell will now have three trees, each maintaining matching pairs in one pair of the given sequences.
- The edit distance recurrence is

$$T[i,j,g] = \min_{x \in \Sigma} \begin{cases} T[i,j,g-1] + w_s, \\ T[i,j-1,g] + w_s, \\ T[i-1,j,g] + w_s, \\ T[i,j-1,g-1]+ \\ \quad \min(w(x,S_2[j]) + w(x,S_3[g]) + w_s, 2w_s), \\ T[i-1,j,g-1]+ \\ \quad \min(w(x,S_1[i]) + w(x,S_3[g]) + w_s, 2w_s), \\ T[i-1,j-1,g]+ \\ \quad \min(w(x,S_1[i]) + w(x,S_2[j]) + w_s, 2w_s), \\ T[i-1,j-1,g-1] + w(x,S_1[i])+ \\ \quad w(x,S_2[j]) + w(x,S_3[g]) \end{cases} \tag{3}$$

- The calculation of values for final endpoints adds w_a for each time that an arc mismatches, and $2w_a$ if all three final endpoints that are being matched are from non-matching arcs. All three trees are managed in the same way as before.

As trees with an upper bound of $\binom{r+s}{s}$ will occur at most $\binom{k}{r}\binom{k}{s}2^k$ times for each of the three pairwise sequence comparisons, the number of tree entries associated with any table cell is

$$3 \cdot \sum_{r=0}^{k} \sum_{s=0}^{k} \binom{k}{r} \binom{k}{s} 2^k \binom{r+s}{s} \quad \leq \quad 3 \cdot 18^k \tag{4}$$

which means that the algorithm as a whole runs in $O(18^k n^3)$ time. ∎

This algorithm can be modified as in the previous section to determine the arc-annotated median sequence. Trees must now be compared in sets of three to determine the paths for locations between the initial and final endpoints of matched arcs. The edit distance calculation phase must also store the symbol choice at each position in addition to the pointer to the previous cell. The median sequence itself can then be extracted in the same manner as for the pairwise alignment, and any arcs that were matched become part of the median sequence. If arcs are mismatched, with the accompanying penalty as part of the minimum calculation, then one of the arcs becomes part of the median sequence.

Corollary 2.4 *An optimal arc-preserving median sequence of three arc-annotated sequences can be computed in* $O(18^k n^3)$ *time.*

3 Substring-Annotated Sequences

3.1 Pairwise Alignment with Substring Structure

An edit distance between two substring-annotated sequences can be defined recursively starting from the top-level sequences. Each such sequence can be considered as a sequence of substrings; comparisons are thus also done between substrings instead of just between symbols. To find the edit distance between the top level sequences, the distances between their children are found, and then used in the distance computation. Deletion of subtrees produces a penalty proportional to the number of leaves (length of the substring, l_i). Since the sequences may have different numbers of levels, any edit distance or alignment algorithm should allow levels to be bypassed (with penalties for the removal of the matched subtree's siblings).

Given two substring-hierarchy annotated sequences S_1 and S_2 such that the maximum sequence length is n and the maximum number of substrings per sequence is m, we compute the *substring-preserving edit distance* between S_1 and S_2 using an initial weight matrix $w(x, y)$ over Σ and a space penalty w_s. Since substrings may be labeled, let $w(c_1, c_2)$ be the penalties for changing the type of substring from label c_1 to label c_2. The recursive alignment computation can be efficiently simulated using a dynamically calculated table of substring edit weights under standard pairwise edit distance d_e computed relative to the following recurrence:

$$\begin{aligned} T[i,j] \quad = \quad & min(T[i,j-1] + w_s \cdot l_j, T[i-1,j] + w_s \cdot l_i, \\ & T[i-1,j-1] + w(S_1[i], S_2[j])) \end{aligned} \tag{5}$$

Table 2: An Algorithm for Computing Substring-Preserving Edit Distance.

1. Build the substring trees for both sequences.
2. Number the nodes in each tree in postorder, starting with $|\Sigma|$; let k_A (k_B) be the root number in sequence A (B) and l_i be the number of leaves (symbols) in the subtree rooted at node i.
3. Extract the sequence of child indices $indseqA_i$ ($indseqB_i$) for each node i in the substring tree for A (B).
4. Build a $(k_A + 1) \times (k_B + 1)$ weight table, and initialize submatrix $[0..|\Sigma| - 1] \times [0..|\Sigma| - 1]$ of this table with the symbol edit weights.
5. **for** $i = 0$ **to** k_A
   for $j = 0$ **to** k_B
   $d = d_e(indSeqA_i, indSeqB_j)$;
   $$w(i,j) = \min \begin{cases} d + w(c_i, c_j), \\ \min_{t \in indSeqA_i} \quad w(t,j) + w_s \cdot (l_i - l_t), \\ \min_{t \in indSeqB_j} \quad w(i,t) + w_s \cdot (l_j - l_t) \end{cases}$$
6. **return**($w(k_A, k_B)$)

This weight table is an extension of the original table of symbol edit weights that includes weights for matching a substring to a single symbol.

Theorem 3.1 *The minimum substring-preserving edit distance between two substring-annotated strings can be computed in $O((n+m)^2)$ time.*

Proof: The required algorithm is given in Table 2. As each pair of substrings or symbol instances is compared exactly once and each substring comparison result is used three times, the algorithm runs in $O((n+m)^2)$ time. ∎

This edit-distance computation can be converted to an alignment algorithm with the addition of cell pointers and a modified pointer traceback procedure. Traceback needs to be done for each table used to compute the edit distance of two index sequences; moreover, traceback must be done as soon as the edit distance for those index sequences has been computed. The alignment is then stored with the corresponding entry in the edit weight table. After the computation of the overall edit distance is completed, the alignment is extracted from the weight table by recursively substituting for the node indices in the alignment.

Corollary 3.2 *An optimal substring-preserving alignment of two substring-annotated sequences can be computed in* $O((n+m)^2)$ *time.*

Proof: The dynamic programming tables are computed in $O((n+m)^2)$ time, traceback requires $O(n+m)$ time, and the alignment can be extracted from the weight table in $O(n+m)$ time; hence, the algorithm as a whole runs in $O((n+m)^2)$ time. ∎

3.2 Three Sequence Median with Substring Structure

Once again, the pairwise alignment algorithm will be modified to create a 3-median sequence algorithm. No algorithm can efficiently consider all possible symbols at each position because the alphabet has been extended to be a set of hierarchical substrings. Thus the previously calculated substring medians must be used to construct the sequence median. As the substrings can be aligned with a space, the alignment score of pairs of substrings also needs to be stored. The substring edit distance d_e is now calculated relative to triples of substrings using the following recurrence:

$$T[i,j,g] = \min_{x\in\Sigma} \begin{cases} T[i,j,g-1] + w_s \cdot (l_g), \\ T[i,j-1,g] + w_s \cdot (l_j), \\ T[i-1,j,g] + w_s \cdot (l_i), \\ T[i,j-1,g-1] + w(S_2[j], S_3[g]), \\ T[i,j-1,g-1] + w_s \cdot (l_j + l_g), \\ T[i-1,j,g-1] + w(S_1[i], S_3[g]), \\ T[i-1,j,g-1] + w_s \cdot (l_i + l_g), \\ T[i-1,j-1,g] + w(S_1[i], S_2[j]), \\ T[i-1,j-1,g] + w_s \cdot (l_i + l_j), \\ T[i-1,j-1,g-1] + w(S_1[i], S_2[j], S_3[g]) \end{cases} \tag{6}$$

Note that $w(S_1[i], S_2[j])$ is a two-string weight created during the pairwise substring comparisons, and $w(S_1[i], S_2[j], S_3[g])$ is the three-string weight computed in the main part of the algorithm.

Theorem 3.3 *The minimum substring-preserving 3-median distance for a triple of substring-annotated sequences can be computed in* $O((n+m)^3)$ *time.*

Proof: The required algorithm is given in Table 3. Each of $(n+m)^3$ distances between substrings or symbol instances will be calculated once, and used 7 times, so this algorithm runs in $O((n+m)^3)$ time. ∎

If the medians of substring triples are stored with the edit distance entries in the dynamic weight table, they can be extracted recursively in the same way as the alignments are extracted in Corollary 3.2.

Corollary 3.4 *An optimal substring-preserving median sequence of three substring-annotated sequences can be computed in* $O((n+m)^3)$ *time.*

4 Discussion

Future research should both test the algorithms given here on real data and consider algorithms for the generalized n-median and tree alignment problems relative to the annotations examined here. Though it seems reasonable to conjecture that these problems will be NP-complete in general, there may yet be algorithms that are usable in practice under restrictions such as bounded cutwidth. Other types of sequence annotation should also be examined. In terms of representation power and computational efficiency, arc annotation is general but expensive and hierarchical substring annotation is restricted but cheap. Perhaps intermediate types of annotation, *e.g.*, joining substrings with arcs,[3] may strike a better balance of power and efficiency.

References

1. B. DasGupta *et al* in *RECOMB'97* (ACM Press, New York, 1997).
2. P.A. Evans in *CPM 1999* (Springer-Verlag, Berlin, 1999).
3. P.A. Evans, *Algorithms and Complexity for Annotated Sequence Analysis*. Ph.D. thesis (University of Victoria, 1999).
4. C. de la Higuera and F. Casacuberta, *Theor. Comp. Sci.*, **230**, 39 (2000).
5. T. Jiang *et al* in *CPM 2000* (Springer-Verlag, Berlin, 2000).
6. R.B. Lyngsø and C.N.S. Pedersen in *RECOMB'2000* (ACM Press, New York, 2000).
7. V. Moulton *et al*, *J. Comp. Biol.*, **7(1/2)**, 277 (2000).
8. M. Murata *et al*, *PNAS USA*, **82**, 3073 (1985).
9. S. Needleman and C. Wunsch, *J. Mol. Biol.*, **48**, 443 (1970).
10. Y. Van de Peer *et al*, *Nucleic Acids Res.*, **28(1)**, 175 (2000).
11. P. De Rijk *et al*, *Nucleic Acids Res.*, **28(1)**, 177 (2000).
12. D. Sankoff, *SIAM J. Appl. Math.*, **45(5)**, 810 (1985).
13. D. Sankoff and R.J. Cedergren in *Time Warps, String Edits, and Macromolecules*, ed. D. Sankoff and J.B. Kruskal (Addison-Wesley, Reading, MA, 1983).
14. D. Sankoff *et al*, *J. Mol. Evol.*, **7**, 133 (1976).
15. L. Wang and T. Jiang, *J. Comp. Biol.*, **1(4)**, 337 (1994).
16. K. Zhang and D. Shasha. *SIAM J. Comp.*, **18**, 1245 (1989).

Table 3: An Algorithm for Computing Substring-Preserving 3-Median Edit Distance.

1. Build the substring trees for all three sequences.
2. Number the nodes in each tree in postorder, starting with $|\Sigma|$; let k_A (k_B) [k_C] be the root number in sequence A (B) [C] and l_i be the number of leaves (symbols) in the subtree rooted at node i.
3. Extract the sequence of child indices $indseqA_i$ ($indseqB_i$) [$indseqC_i$] for each node i in the substring tree for A (B) [C].
4. Run the pairwise substring-preserving edit distance algorithm for each of the sequence-pairs (A, B), (A, C) and (B, C).
5. Build a $(k_A + 1) \times (k_B + 1) \times (k_C + 1)$ weight table, and initialize the $[0..|\Sigma| - 1] \times [0..|\Sigma| - 1] \times [0..|\Sigma| - 1]$ submatrix of this table with the symbol edit weights.
6. **for** $i = 0$ **to** k_A
   for $j = 0$ **to** k_B
   for $g = 0$ **to** k_C
   $d = d_e(indSeqA_i, indSeqB_j, indSeqC_g)$;

$$w(i,j,g) = \min_{c \in C} \begin{cases} d + w(c, c_i) + w(c, c_j) + w(c, c_g), \\ \min_{t \in indSeqA_i} \\ \quad w(t, j, g) + w_s \cdot (l_i - l_t), \\ \min_{t \in indSeqB_j} \\ \quad w(i, t, g) + w_s \cdot (l_j - l_t), \\ \min_{t \in indSeqC_g} \\ \quad w(i, j, t) + w_s \cdot (l_g - l_t), \\ \min_{t_1 \in indSeqA_i, t_2 \in indSeqB_j} \\ \quad w(t_1, t_2, g) + w_s \cdot \\ \quad ((l_i - l_{t_1}) + (l_j - l_{t_2})), \\ \min_{t_1 \in indSeqA_i, t_2 \in indSeqC_g} \\ \quad w(t_1, j, t_2) + w_s \cdot \\ \quad ((l_i - l_{t_1}) + (l_g - l_{t_2})), \\ \min_{t_1 \in indSeqB_j, t_2 \in indSeqC_g} \\ \quad w(i, t_1, t_2) + w_s \cdot \\ \quad ((l_j - l_{t_1}) + (l_g - l_{t_2})) \end{cases}$$

7. **return**$(w(k_1, k_2, k_3))$