An edit distance between RNA stem-loops

Valentin Guignon
Programme de Bioinformatique, Université de Montréal

Cedric Chauve
LaCIM, Département d'Informatique, Université du Québec à Montréal

Sylvie Hamel
LBIT, DIRO, Université de Montréal

String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, volume 3772 of *Lecture Notes in Computer Science*, p. 334–345, 2005.

An edit distance between RNA stem-loops

Valentin Guignon[1⋆], Cedric Chauve[2⋆⋆] and Sylvie Hamel[3⋆⋆⋆]

[1] Programme de Bioinformatique, Université de Montréal,
Pav. André-Aisenstadt, CP 6128 succ. Centre-ville, Montréal (QC), H3C 3J7, Canada
[2] LaCIM, Département d'Informatique, Université du Québec à Montréal,
C.P. 8888 succ. Centre-Ville, Montréal (QC), H3C 3P8, Canada
[3] LBIT, DIRO, Université de Montréal,
Pav. André-Aisenstadt, CP 6128 succ. Centre-ville, Montréal (QC), H3C 3J7, Canada
guignonv@yahoo.fr, chauve@lacim.uqam.ca, sylvie.hamel@umontreal.ca

Abstract. We introduce the notion of conservative edit distance and mapping between two RNA stem-loops. We show that unlike the general edit distance between RNA secondary structures, the conservative edit distance can be computed in polynomial time and space, and we describe an algorithm for this problem. We show how this algorithm can be used in the more general problem of complete RNA secondary structures comparison.

1 Introduction

In this paper, we address a classical problem in bioinformatics, the comparison of two RNA secondary structures, and we describe a new algorithm to compute an edit distance and a mapping between two RNA secondary structures. The importance and functional variety of the several types of known RNA molecules, especially non-coding RNAs like transfer RNAs (tRNAs), ribosomal RNAs (rRNAs), untranslated regions (UTRs), small nuclear RNA (snRNAs) for example, is a strong motivation for their study [4], in particular, in the comparative approach that relates combinatorial similarity between molecules to functional similarity. Several algorithms exist to compare RNA secondary structures, most of them based on the encoding of RNA secondary structures by ordered trees, followed by the computation of an edit distance and a mapping between these trees. This approach was first used in [9, 10], and is part of the popular Vienna RNA Package [7]. However, the set of edit operations considered in these works appears to be too limited, as it does not contain some edit operations that correspond to natural evolutionary events for RNA structures, a problem that is discussed in [1]. Recently, Jiang *et al.* introduced a new edit distance model [8] that is more realistic, as it contains a broader set of edit operations, but also less tractable algorithmically, as computing the edit distance between two RNA secondary structures in this model is NP-hard [3].

⋆ Supported by a scholarship of the Génome Québec program "Comparative and integrative bioinformatics".
⋆⋆ Supported by grants from NSERC and FQRNT.
⋆⋆⋆ Supported by grants from NSERC and FQRNT.

In the present work, we consider the set of edit operations defined in [8]. Our main result is that, when the compared RNA structures are simple enough and the set of allowed mappings is restricted, the distance and mapping computation becomes tractable, and even easy. More precisely, we consider the computation of a *conservative mapping* between two *stem-loops*; a precise statement of the problem is given in Section 2, but intuitively, a mapping is said to be conservative if it describes only evolutionary events involving bases that are closely located in terms of secondary structure. We show that such a computation can be done efficiently with a dynamic programming algorithm that is an extension of the classical algorithm computing an edit distance between strings. The motivations for considering such a restricted notion of distance and mapping are the following. First, every RNA secondary structure can be decomposed into a sequence of stem-loops and stem-loop-like substructures, that can be handled by our algorithm. For example, several families of non-coding RNAs, like tRNAs, snoRNAs H/ACA or miRNAs precursors, have a secondary structure mostly composed of a few stem-loops. Second, two closely related, from the evolutionary point of view, RNA structures – not limited to stem-loops – should share several stem-loops that are very similar and whose comparison will be well represented by a conservative mapping. Moreover, if two compared RNA structures share only some similar motifs that contain stem-loops, our approach based on the decomposition in stem-loop-like substructures and their pairwise comparisons can highlight such locally conserved motifs, a problem that received some attention recently [2, 6].

This paper is organized as follows. In Section 2, we state precisely the problem we address, namely the computation of a conservative distance and mapping between stem-loops. In Section 3, we describe and analyze a dynamic programming algorithm that solves this problem. In Section 4, we show how the stem-loops comparison can be used in a very simple way to compare complete RNA secondary structures and we illustrate this approach with the comparison of two RNAse P RNA.

2 Edit distance between stem-loops

Tree representation of stem-loops. RNA primary structure is generally represented by a string over the four letters alphabet $\{A, C, G, U\}$. This primary structure folds back onto itself to form the secondary structure, that is a planar structure containing unpaired bases from $\{A, C, G, U\}$ and base pairs, that are ordered pairs of bases, the most common being AU, UA, CG, GC, GU, UG. This secondary structure can be represented by an ordered tree[1], where, for a given RNA secondary structure, each base pair is represented by an internal node labeled by this base pair and each unpaired base by a leaf, labeled by this base (see Figure 1). In such a representation, a multi-loop corresponds to an internal node having several children that are internal nodes. We call *stem-loop* an RNA secondary structure without multi-loop, which implies that the tree representing

[1] Depending on the level of comparison, an RNA can be represented by different trees, see Allali and Sagot [1] for example.

a stem-loop is *linear*: each internal node has at most one child that is also an internal node. Note that linear trees are quite similar to strings, as a string can be seen as a linear tree with only one leaf.

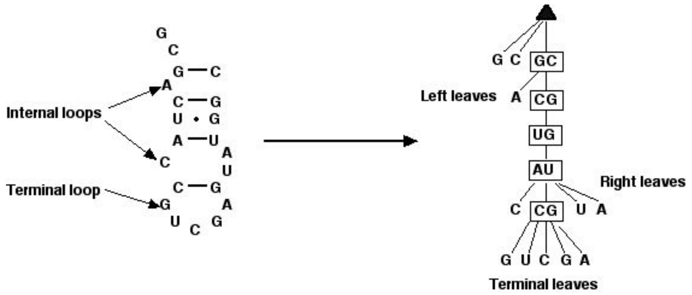

Fig. 1. A stem-loop and its tree representation.

Given a linear tree representing a stem-loop, leaves can be of three kinds: *left leaves*, *right leaves* and *terminal leaves*. The leaves representing unpaired bases of the terminal loop are the terminal leaves, the leaves representing unpaired bases located on the left (resp. right) of the stem-loop are the left leaves (resp. right leaves). In Figure 1, the terminal leaves are G, U, C, G, A, the left leaves are G, C, A, C and the right leaves U, A.

Edit operations. Let T_1 and T_2 be two linear trees representing two stem-loops. Edit operations represent evolutionary events acting on secondary structures. The set of edit operations we describe now is the same that was described in [8][2], augmented of additional constraints on some of these operations. These operations are illustrated in Figure 2.

We define an *edit operation between T_1 and T_2* as a couple (a, b), where a (resp. b) can be an internal node or a leaf of T_1 (resp. of T_2), or a pair of leaves of T_1 (resp. of T_2) that have the same parent, or the symbol $-$ (but both a and b can not be $-$).

If a and b are both internal nodes or both leaves, the operation (a, b) is a *relabeling*: the node a changes its label to become b. If $b = -$, (a, b) is a *deletion*: the node a is removed from T_1. If $a = -$, (a, b) is an *insertion*, the symmetric operation of a deletion. We add the following constraint on the relabeling operation: if a and b are both leaves, respectively of T_1 and T_2, then either one of these two leaves is a terminal leaf, or both of them are left leaves, or both are right leaves. We call these three operations the *simple operations*.

If a is an internal node and b a leaf, (a, b) is called an *altering* – the internal node a is replaced by the leaf b in T_1 –, and if a is a leaf and b an internal node, it is called a *completion*, which is the symmetric operation of an altering. Finally, (a, b) is called an *arc-breaking* if a is an internal node and b is a pair of leaves, and an *arc-creation* if a is a pair of leaves and b is an internal node. These last four operations, that represent evolutionary events that act on base pairs, were introduced in [8]. We call them *complex operations*. The fact that

[2] Note that in [8] RNA secondary structures are represented by *non-crossing arc-annotated sequences*, but such sequences are naturally equivalent to ordered trees.

these operations represent evolutionary events on base pairs that transform a stem-loop into another stem-loop impose some implicit conditions on the nodes of a linear tree that are involved. In particular, an arc-creation can only involve two leaves that have the same parent, and these two leaves can not be both left leaves or right leaves.

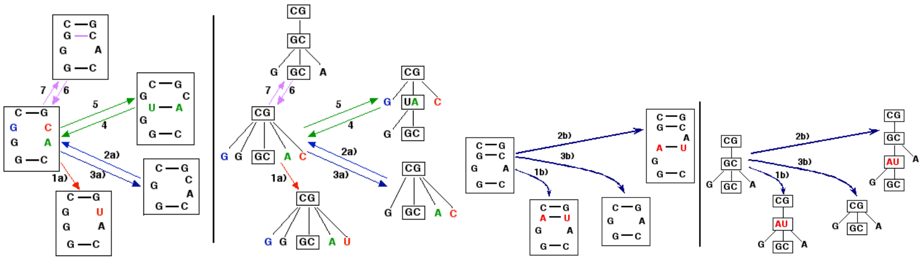

Fig. 2. Illustration of edit operations: relabeling (1a and b), insertion (2a and b), deletion (3a and b), altering and completion (4 and 5), arc-breaking and arc-creation (6 and 7) on stem-loops and trees

Conservative mapping and edit distance. Let T_1 and T_2 be two linear trees and $\mathcal{S} = s_1, \ldots, s_n$ a sequence of edit operations between T_1 and T_2 such that their successive application transforms T_1 into T_2. Such a sequence of edit operations is called a *conservative edit sequence between* T_1 *and* T_2. We shall here notice that the definition of a conservative edit sequence is less general than the general definition of edit sequence, used in [8], as we impose that all the operations of this sequence have to be described on T_1 and T_2. This, for example, forbids that a base that was unpaired by an arc-breaking could later be involved in a arc-creation or a completion. This is why we call conservative such an edit sequence, and we will justify later in this section why we consider such restrictions.

The edit operations of a conservative edit sequence between T_1 and T_2, other than insertions and deletions, naturally induce, by their definition, a *mapping* between nodes of T_1 and T_2, where a leaf of a tree can be mapped to a leaf or an internal node of the other tree, and an internal node can be mapped to an internal node, a leaf, or a pair of leaves. This mapping highlights the bases that are common, up to relabeling, between the stem-loops represented by T_1 and T_2. We call such a mapping a *conservative mapping*.

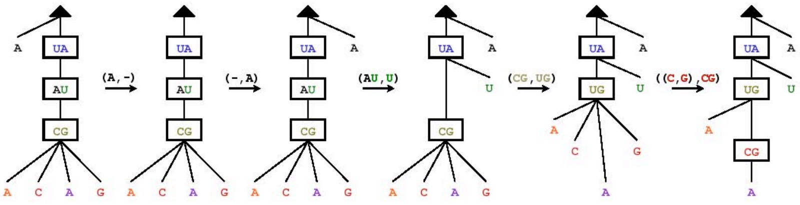

Fig. 3. A conservative edit sequence between two linear trees

Finally, we associate to each edit operation (a, b) a *cost* denoted $\delta(a, b)$. If (a, b) is a relabeling, where the nodes a of T_1 and b of T_2 have the same label,

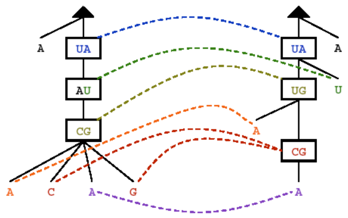

Fig. 4. The conservative mapping corresponding to the edit sequence of Figure 3.

then $\delta(a, b) = 0$. Note that a complex operation (a, b), depending on the label of the nodes in a and b, can also imply a relabeling. Hence, the cost of an operation depends of its nature and of the labels of the involved nodes. The cost associated to an edit sequence between T_1 and T_2 is the sum of the individual cost of each operations in the sequence. The conservative edit distance is the minimal cost of a conservative edit sequence that transforms T_1 into T_2.

We describe in this work an algorithm that computes a conservative mapping and the corresponding edit distance between two linear trees T_1 and T_2.

Discussion on various distances. Several RNA comparison algorithms have been defined based on different subsets of the set of edit operations we defined above. We illustrate now, through a simple example on real data, the influence of the choice of the set of allowed edit operations on the comparison of two structures. Given the two micro-RNAs (miRNAs) precursors of Figure 5, we describe two possible sequences of edit operations that transform the first structure into the second one, based on different sets of edit operations.

Fig. 5. Two miRNAs (mouse and human).

If we consider only the simple operations of relabeling, insertion and deletion, with cost 0.75 (resp. 1.25) for the insertion or deletion of an unpaired base (resp. a base pair) and 0.25 for the relabeling of a base[3], a possible optimal scenario to transform the mouse miRNA precursor into the human one contains 4 operations, for a cost of 3: relabeling (UA, UG), deletion $(U, -)$, insertion $(-, U)$ and insertion $(-, UA)$. If we add to this set of possible edit operations the operations of arc-creation and arc-breaking, each with a cost of 0.5, and the altering and completion operations, with a cost of 1 each for example, then the following scenario, that is conservative, has a better score of 2.25 and seems more

[3] The scores we use here are the same we use in Section 4, where they are discussed.

plausible, from the evolutionary point of view than the previous one: insertion $(-, U)$ (below), relabeling (UA, UG) and completion (U, UA).

This example is a good illustration of why we believe that the set of all edit operations we described above should be considered when comparing RNA secondary structures. However, the problem of computing the general edit distance between RNA secondary structures is NP-hard [3]. And even in the case of the comparison of two stem-loops, it is not known if the general edit distance can be computed in polynomial time. As we will see in the next section, in the case of a conservative distance and mapping between stem-loops, the problem can be solved in polynomial time, due to the similarity between this problem and the problem of computing an edit distance between strings.

Finally, one can see that the restrictions that we impose to define a conservative mapping prevent the evolutionary scenarios corresponding to such mappings to create a base pair between bases that are not closely located in the secondary structure. Hence, if considering only conservative mapping is a strong combinatorial restriction, it should not prevent to obtain a pertinent distance and mapping between stem-loops that are close from an evolutionary point of view, which is our goal in this work. Our experiments on real data, miRNAs precursors (not shown) and RNAse P RNA (Section 4) seem to confirm this intuition.

3 A dynamic programming algorithm

We now describe a dynamic programming algorithm that computes the conservative distance between two stem-loops, by using a unique two-dimensional dynamic programming table. Through all this section, we use distance and mapping respectively for conservative distance and conservative mapping. We recall that a depth-first prefix traversal (DFP) of an ordered tree is a traversal of the tree that visits recursively the children of the root from left to right.

Indexing pairs, predecessor and successor of a node. An ordered pair $I = (x, y)$ of nodes of a linear tree T is called an *indexing pair* if it satisfies one the five following conditions: (1) x is an internal node and $y = x$, (2) x is an internal node and y a right leaf of x, (3) y is an internal node and x a left leaf of y, (4) x and y are respectively a left leaf and a right leaf and they have the same parent, or (5) x and y are terminal leaves, and x is located to the left of y.

An indexing pair (x, y) of T defines a subtree of T, denoted by $T_{(x,y)}$, in the following way: $T_{(x,y)}$ is the tree obtained from T by removing all the nodes visited between x and y during a DFP traversal of T (if $x = y$, this corresponds to removing from T all the nodes other than x in the subtree rooted in x).

We define the *predecessor* of a node x, $\mathrm{p}(x)$, as its immediate left sibling if x is not the leftmost child of its parent, and its parent otherwise. Symmetrically, the *successor* of a node x, $\mathrm{s}(x)$, is its immediate right sibling if x is not the rightmost child of its parent, and its parent otherwise. Note that for an internal node x that is the only child of its parent y, $\mathrm{p}(x) = \mathrm{s}(x) = y$. According to the previous definitions, the root r of a tree does not have a predecessor, neither a successor, so we define them formally by $\mathrm{p}(r) = \mathrm{s}(r) = \emptyset$.

Finally, an indexing pair (x, y) is said to be *terminal* if x (resp. y) is a terminal leaf and $y = \mathrm{s}(x)$ (resp. $x = \mathrm{p}(y)$).

A dynamic programming algorithm. We can now define the dynamic programming table that we use to compute the edit distance between two stem-loops. This table, denoted D, is a two-dimensional table indexed by pairs (I, J) such that I is either an indexing pair of T_1 or $I = \emptyset$, and J is either an indexing pair of T_2 or $\emptyset$. The cell $D[I, J]$ of this table contains the edit distance between the two linear trees T_I and T_J.

It follows immediately from the definition of indexing pairs that we can define the edit distance between T_1 and T_2 in terms of $D[I, J]$. Indeed, if we denote by F_1 and F_2 the sets of terminal indexing pairs respectively of T_1 and T_2, we have:

$$d(T_1, T_2) = \min_{(x,y)\in F_1,(u,v)\in F_2} \{D[(x, y), (u, v)]\}. \tag{1}$$

To compute the table D, we use a dynamic programming algorithm, based on the following equations. First, we initialize the table

$$\begin{cases} D[\emptyset, \emptyset] = 0, \\ D[(x, y), \emptyset] = \sum_{a \text{ node of } T_{1(x,y)}} \delta(a, -), \text{ for all indexing pairs } (x, y) \text{ of } T_1, \\ D[\emptyset, (u, v)] = \sum_{b \text{ node of } T_{2(u,v)}} \delta(-, b), \text{ for all indexing pairs } (u, v) \text{ of } T_2. \end{cases} \tag{2}$$

The general case is composed of 4 sub-cases. In the following equations, we denote by (R) an equation corresponding to a relabeling event, (I) an insertion, (D) a deletion, (AC) an arc-creation, (AB) an arc-breaking, (C) a completion and (A) an altering.

1. If $x = y$ and $u = v$ (x and u are internal nodes),

$$D[(x, y), (u, v)] = \min \left\{ \begin{array}{ll} D[(\mathrm{p}(x), \mathrm{s}(y)), (\mathrm{p}(u), \mathrm{s}(v))] + \delta(x, u), & \text{(R)} \\ D[(\mathrm{p}(x), \mathrm{s}(y)), (u, v)] + \delta(x, -), & \text{(D)} \\ D[(x, y), (\mathrm{p}(u), \mathrm{s}(v))] + \delta(-, u), & \text{(I)} \end{array} \right\}, \tag{3}$$

where $(\mathrm{p}(x), \mathrm{s}(y)) = \emptyset$ if x is the root of T_1 and $(\mathrm{p}(u), \mathrm{s}(v)) = \emptyset$ if u is the root of T_2.

2. If $x \neq y$ and $u \neq v$,

$$D[(x, y), (u, v)] = \min \left\{ \begin{array}{ll} D[(\mathrm{p}(x), y), (u, v)] + \delta(x, -), & \text{(D)} \\ D[(x, \mathrm{s}(y)), (u, v)] + \delta(y, -), & \text{(D)} \\ D[(x, y), (\mathrm{p}(u), v)] + \delta(-, u), & \text{(I)} \\ D[(x, y), (u, \mathrm{s}(v))] + \delta(-, v), & \text{(I)} \\ D[(\mathrm{p}(x), y), (\mathrm{p}(u), v)] + \delta(x, u), & \text{(R)} \\ D[(x, \mathrm{s}(y)), (u, \mathrm{s}(v))] + \delta(y, v), & \text{(R)} \end{array} \right\}. \tag{4}$$

Note that in the above equation, some of the 6 terms of the form $D[I, J] + \delta(\ldots)$ can be undefined. This can happen if I and/or J is neither $\emptyset$, nor an indexing pair: for example if x is an internal node and y a right leaf of x, then $(\mathrm{p}(x), y)$ is not an indexing pair of nodes of T_1. In such a case, the function min will not take into account these undefined terms.

3. If $x = y$, and $u \neq v$

$$D[(x,y),(u,v)] = \min \left\{ \begin{array}{ll} D[(\mathrm{p}(x),\mathrm{s}(y)),(u,v)] + \delta(x,-), & \text{(D)} \\ D[(x,y),(\mathrm{p}(u),v)] + \delta(-,u), & \text{(I)} \\ D[(x,y),(u,\mathrm{s}(v))] + \delta(-,v), & \text{(I)} \\ D[(\mathrm{p}(x),\mathrm{s}(y)),(\mathrm{p}(u),v)] + \delta(x,u), & \text{(A)} \\ D[(\mathrm{p}(x),\mathrm{s}(y)),(u,\mathrm{s}(v))] + \delta(x,v), & \text{(A)} \\ D[(\mathrm{p}(x),\mathrm{s}(y)),(\mathrm{p}(u),\mathrm{s}(v))] + \delta(x,(u,v)) & \text{(AB)} \end{array} \right\}, \quad (5)$$

where the same remark as in sub-case 2, about possibly undefined terms, applies.
4. If $x \neq y$ and $u = v$,

$$D[(x,y),(u,v)] = \min \left\{ \begin{array}{ll} D[(x,y),(\mathrm{p}(u),\mathrm{s}(v))] + \delta(-,u), & \text{(I)} \\ D[(\mathrm{p}(x),y),(u,v)] + \delta(x,-), & \text{(D)} \\ D[(x,\mathrm{s}(y)),(u,v)] + \delta(y,-), & \text{(D)} \\ D[(\mathrm{p}(x),y),(\mathrm{p}(u),\mathrm{s}(v))] + \delta(x,u), & \text{(C)} \\ D[(x,\mathrm{s}(y)),(\mathrm{p}(u),\mathrm{s}(v))] + \delta(y,u), & \text{(C)} \\ D[(\mathrm{p}(x),\mathrm{s}(y)),(\mathrm{p}(u),\mathrm{s}(v))] + \delta((x,y),u) & \text{(AC)} \end{array} \right\}, \quad (6)$$

where again the same remark as in sub-case 2, about possibly undefined terms, applies.

We now describe the algorithm to fill all the cells of the table D. An indexing pair (u,v) of a tree T is said to be *ancestral* for the indexing pair (w,z) of T if $(u,v) = \emptyset$ or u (resp. v) is not visited after w (resp. before z) during a DFP traversal of T. It follows from this definition and from the equations above that, in order to compute the table D, we have to enumerate all the couples (I,J) of indexing pairs of T_1 and T_2 in a way that preserves the ancestral order for I and J: $D[I,J]$ will be computed after all the cells $D[I',J']$ where I' is ancestral for I and J' is ancestral for J. Such an enumeration scheme is easy to design for a given tree, based on parallel depth-first prefix and postfix traversals of this tree, and can be performed in time that is linear in the number of indexing pairs for this tree. Given D, a mapping is a path in this table computed with the classical *backtracking* method used to compute the alignment of two strings.

Complexity analysis. The space complexity of this algorithm is given by the size of the table D, i.e the number of couples (I,J) where I is an indexing pair of T_1 and J an indexing pair of T_2. Let $\mathrm{ind}(T_1)$ and $\mathrm{ind}(T_2)$ denote respectively the number of indexing pairs of T_1 and T_2: the table D contains $\Theta(\mathrm{ind}(T_1) \times \mathrm{ind}(T_2))$ cells.

As the enumeration of all indexing pairs of the trees T_1 and T_2 respecting the ancestral relation can be performed in time linear in the number of such pairs for each tree, the initialization of the table (equation (2)) can be computed in $O(\mathrm{ind}(T_1) \times \mathrm{ind}(T_2))$ time. Moreover, filling one cell of the table, using the dynamic programming equations (3), (4), (5) and (6) can be done in constant time, since testing if a pair of nodes is indexing takes a constant time. Note also that the predecessor and successor of every node of a tree can easily be computed, prior to the computation of the table D, in linear time during a DFP traversal of this tree. Finally, once D has been filled, computing the edit distance using equation (1) can be done by visiting the cells indexed by pairs of terminal indexing pairs, and so in $O(\mathrm{ind}(T_1) \times \mathrm{ind}(T_2))$ time. This leads to the result that the time complexity for computing the conservative edit distance between T_1 and T_2 is $\Theta(\mathrm{ind}(T_1) \times \mathrm{ind}(T_2))$ time. It follows from the similarity between our algorithm and the string edit distance algorithm that computing a mapping from D asks for the same time, that is $\Theta(\mathrm{ind}(T_1) \times \mathrm{ind}(T_2))$.

Let n_1 be the number of nodes of T_1, m_1 be the number of internal nodes of T_1, $\{x_1, \ldots, x_{m_1}\}$ these internal nodes, ℓ_i and r_i the number of left and right leaves of x_i, for $i = 1, \ldots, m-1$, and t_1 the number of terminal leaves. The number $\mathrm{ind}(T_1)$ of indexing pairs in T_1 is exactly

$$m_1 + (t_1(t_1-1)/2) + 2t_1 + \sum_{i=1}^{m_1-1} ((\ell_i+1) \times (r_i+1) - 1), \tag{7}$$

where these four terms correspond respectively to the number of indexing pairs formed by two occurrences of the same internal node, those formed by two terminal leaves, those formed by a terminal leaf and x_{m_1} and, finally, those formed with at least one non terminal leaf.

Hence, $\mathrm{ind}(T_1) \in O(n_1^2)$, and, if we denote by n_2 is the number of nodes in T_2, the overall distance and mapping algorithm has a worst-case time complexity in $O(n_1^2 \times n_2^2)$. However, it is interesting to remark that, if T_1 is a tree representing a stem-loop with few unpaired bases, or small loops (internal loops and the terminal loop), then $\mathrm{ind}(T_1)$ is closer to n_1 than to n_1^2. Hence, when comparing stem-loops with such characteristics in terms of unpaired bases and loops, the algorithm asks for a time that is, in practice, in only quadratic.

4 Comparison of complete secondary structures

In this section, we describe a simple method that allows the comparison of two complete RNA secondary structures R_1 and R_2, based on the stem-loops comparison algorithm of the previous section. This method has three phases: (1) decomposition of the two RNA structures into two sequences of stem-loops substructures, (2) independent pairwise comparisons between the stem-loops of R_1 and the stem-loops of R_2, and (3) finally, an alignment of these two sequences of stem-loops using the distances computed during the phase (2).

Given an RNA secondary structure R, if one removes all the unpaired bases belonging to multi-loops, one obtains a set of substructures with no multi-loops. Even if these substructures are not all stem-loops under the classical definition of the term, due to the fact that some of them do not have a terminal loop, we call them stem-loops, as the algorithm we described in Section 3 does not need any major modification to handle stem-loops that do not have a terminal loop. This set of substructures is naturally ordered by the sequence of bases that forms the primary structure of R, as illustrated in Figure 6.

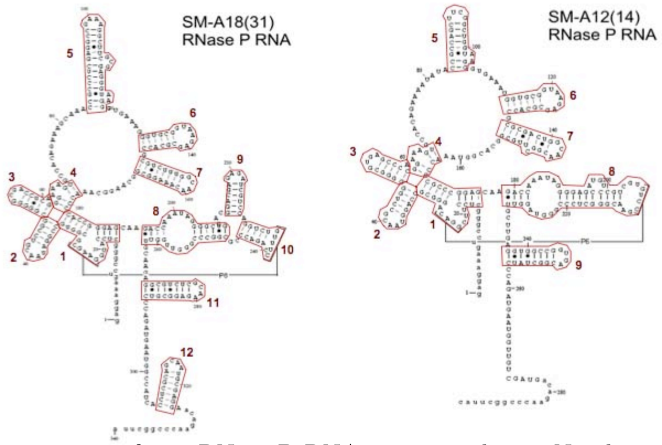

Fig. 6. Decomposition of two RNase P RNA into stem loops. Numbers indicate the order on each of the two sets of stem-loops.

Now, let $R_1^1, \ldots, R_1^k$ and $R_2^1, \ldots, R_2^\ell$ be the two sequences of stem-loops given by the decompositions of two complete RNA secondary structures R_1 and R_2. We use a table P, indexed by pairs of integers belonging to $\{0, \ldots, k\} \times \{0, \ldots, \ell\}$ where $P[i, j]$ is the distance between R_1^i and R_2^j – with $R_1^0 = R_2^0$ being the empty stem-loop –, computed using the algorithm of Section 3. The table of Figure 7 corresponds to the pairwise comparisons of the stem-loops of Figure 6, with the following costs: 0.25 for the relabeling of a single base, 0.4 for the relabeling of the two bases of a base pair, 0.75 (resp. 1.25) for the deletion and the insertion of a leaf (resp. an internal node), 0.5 for an arc-breaking and an arc-creation, and 1 for a completion and an altering. These costs were chosen in such a way that no edit operation can be replaced, for a smaller cost, by a sequence of other edit operations. Moreover, the results we present below did not differ when alternative cost schemes, that had the same property, were used.

Stem-loops Distances
SM-A18(31) RNase P RNA

SM-A12(14) RNase P RNA

	-	1	2	3	4	5	6	7	8	9
-	0	26	15	15	19	22	19	24	52	20
1	27	2,8	13,45	16,25	13,65	15,6	13,05	16,25	25,7	14,15
2	15	12,95	0,8	4,2	10,1	7,9	5,35	8,65	33	6,95
3	15	14,05	3,65	1,3	9,7	7,4	6,45	8,3	32,75	5,8
4	19	11,85	9,85	9,45	0,25	12,95	9,7	12,6	32,45	10,55
5	37	21,55	18,8	18,65	19,95	13	16,35	13,4	25,8	16,1
6	19	13,3	4,95	5,9	9,7	7,5	0	8,25	30,4	4,8
7	23	15,25	9,2	7,55	12,2	7,15	7,7	2,7	29,4	5,1
8	33	11,95	17,8	19	15,2	15,7	15,5	16,85	18,4	16,95
9	16	13,6	3,7	3,2	9,15	7,45	4,55	7,8	32,65	4,55
10	16	9,95	4,6	5,7	8,25	8,95	4,5	9,95	31,4	6,4
11	20	13,4	6,85	6,05	10,55	5,65	5,05	5,05	30,55	2,35
12	20	13,3	6,05	6,45	11,35	5,4	5,25	5,75	30,7	3,3

Fig. 7. Pairwise distances between the stem-loops of Figure 6.

Finally, we apply the classical string global alignment algorithm (see [5] for example) to these two sequences of stem-loops, using the table P to define the cost of the insertion or deletion of a given stem-loop, and the score of a matching between two stem-loops. The resulting dynamic programming table is given in Figure 8, where marked cells describe the alignment of stem-loops obtained by backtracking. To obtain from this table a mapping, one can use the classical backtracking method, both in the table of the alignment of the sequences of stem-loops and in the tables of the pairwise alignments of stem-loops.

Global Alignment

SM-A18(31) RNase P RNA (rows); SM-A12(14) RNase P RNA (columns)

	-	1	2	3	4	5	6	7	8	9
-	0	26	41	56	75	97	116	140	192	212
1	27	2,8	17,8	32,8	51,8	73,8	92,8	116,8	165,7	185,7
2	42	17,8	3,6	18,6	37,6	59,6	78,6	101,5	149,8	169,8
3	57	32,8	18,6	4,9	23,9	45	64	86,9	134,2	154,2
4	76	51,8	37,6	23,9	5,15	27,15	46,15	70,15	119,4	139,4
5	113	88,8	70,6	56,25	42,15	18,15	37,15	59,55	95,95	116
6	132	107,8	89,6	75,25	61,15	37,15	18,15	42,15	89,95	100,8
7	155	130,8	112,6	97,15	84,15	60,15	41,15	20,85	71,55	91,55
8	188	163,8	145,6	130,2	112,4	93,15	74,15	53,85	39,25	59,25
9	204	179,8	161,6	146,2	128,4	109,2	90,15	69,85	55,25	43,8
10	220	195,8	177,6	162,2	144,4	125,2	106,2	85,85	71,25	59,8
11	240	215,8	197,6	182,2	164,4	145,2	126,2	105,9	91,25	73,6
12	260	235,8	217,6	202,2	184,4	165,2	146,2	125,9	111,3	93,6

Fig. 8. Alignment of the two sequences of stem-loops of Figure 6 using the table of Figure 7: marked cells indicate an optimal stem-loops alignment.

As it appears on Figure 8, this algorithm, applied on the two quite similar RNAse P RNAs of Figure 6 gives a good result, even if the stem-loops 8, 9 and 10 of the RNAse P RNA SM-A18(31) show that this method is sensitive to the insertion of a stem-loop into another stem-loop. However, the comparisons of the other stem-loops that were very similar compensated this problem.

If n_1 is the number of bases of R_1 and n_2 the number of bases of R_2, it follows immediately from the complexity of comparing two stem-loops that the comparison of R_1 and R_2 is performed in $O(n_1^2 \times n_2^2)$ in the worst-case time. However, the low number of unpaired bases in the two sets of stem-loops of our example makes that the effective time complexity was only quadratic.

It is also interesting to notice that all the different variants of the alignment of strings can be used with our method. For example, if one wants to discover clusters of close stem-loops that are similar in R_1 and R_2, that is local motifs, one just has to use the algorithm for local alignment of strings instead of the global string alignment algorithm.

5 Conclusion

We described in this paper an efficient algorithm for the comparison of stem-loops, intended to give good results for stem-loops that are evolutionary close. By imposing some restrictions on the set of possible mappings that are considered, we were able to use the complete set of edit operations defined in [8]. Moreover, we sketched a method that allows to use the stem-loops comparison algorithm as a basis for the comparison of complete RNA secondary structures. Experimental results suggest that this approach gives interesting results.

This work raises several interesting algorithmical questions. First, it would be interesting to see at which point the fact to consider stem-loops makes easier the edit distance computation: is computing the general edit distance of [8] between stem-loops NP-hard ? And if this is the case, are there definitions of some mapping, less restrictive than conservative mappings, that allow a polynomial time computation. From a preliminary work on this question, it seems that it is possible to relax the locality of interactions between bases imposed in a conservative mapping and that one can consider interactions between bases that do not belong to the same internal loop. However, this makes the computation more time-consuming, at least in practice.

It would also be important to understand more deeply the influence of the cost scheme of edit operations on the final result, as it was done in string algorithms. It is for example possible that some cost schemes allow to compute in polynomial time the general edit distance.

The most interesting question concerns the way to use the comparison of stem-loops in the comparison of complete secondary structures. We used here a simple method based on the alignment of strings, that has the good property to be efficient in terms of computing time. In [1], Allali and Sagot introduced the notion of *multilevel RNA structure comparison*, that considers several levels of representation of an RNA structure into trees. The method we described in Section 4 follows this principle in fact, but does not consider the high level architecture of this structure as it considers the stem-loops in a simple sequence. It then would be interesting to combine our algorithm with the multi-level approach of [1].

References

1. J. Allali and M.-F. Sagot. A new distance for high level RNA secondary structure comparison. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):4–14. 2005.
2. R. Backofen and S. Will. Local sequence-structure motifs in RNA. *J. Bioinform. Comput. Biol.*, 2(4):681–698. 2004.
3. G. Blin, G. Fertin and C. Sinoquet. RNA sequences and the EDIT(NESTED, NESTED) problem. Report RR-IRIN-03.07, IRIN (Nantes, France). 2003.
4. J. Couzin. Breakthrough of the year: small RNAs make big splash. *Science*, 298:2296–2297. 2002.
5. D. Gusfield *Algorithms on strings, trees and sequences.* Cambridge University Press. 1997.
6. M. Höchsmann, T. Töller, R. Giegerich and S. Kurtz. Local similarity in RNA secondary structures. In *2nd IEEE Computer Society Bioinformatics Conference (CSB 2003)*, pages 159–169, IEEE Computer Society. 2003.
7. I.L Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Res.*, 31(13):3429–3431. 2003.
8. T. Jiang, G. Lin, B. Ma and K. Zhang. A general edit distance between RNA structures. *J. Comput. Biol.*, 9(2):371–388. 2002.
9. B.A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6(4):309–318. 1988.
10. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262. 1989.