Novel tree edit operations for RNA secondary structure comparison

Julien Allali[1] and Marie-France Sagot[2]

[1] Institut Gaspard-Monge, Université de Marne-la-Vallée, Cité Descartes, Champs-sur-Marne, 77454 Marne-la-Vallée Cedex 2, France, allali@univ-mlv.fr

[2] Inria Rhône-Alpes, Université Claude Bernard, Lyon I, 43 Bd du 11 Novembre 1918, 69622 Villeurbanne cedex, France, Marie-France.Sagot@inria.fr, and King's College, London, UK

Abstract. We describe an algorithm for comparing two RNA secondary structures coded in the form of trees that introduces two novel operations, called *node fusion* and *edge fusion*, besides the tree edit operations of deletion, insertion and relabelling classically used in the literature. This allows us to address some serious limitations of the more traditional tree edit operations when the trees represent RNAs and what is searched for is a common structural core of two RNAs. Although the algorithm complexity has an exponential term, this term depends only on the number of successive fusions that may be applied to a same node, not on the total number of fusions. The algorithm remains therefore efficient in practice and is used for illustrative purposes on ribosomal as well as on other types of RNAs.

keywords: tree comparison, edit operation, distance, RNA, secondary structure

1 Introduction

RNAs are one of the fundamental elements of a cell. Their role in regulation has been shown recently to be far more prominent than initially believed (20 December 2002 issue of *Science*, which designated small RNAs with regulatory function as the scientific breakthrough of the year). It is now known, for instance, that there is massive transcription of non-coding RNAs. Yet current mathematical and computer tools remain mostly inadequate to identify, analyse and compare RNAs.

An RNA may be seen as a string over the alphabet of nucleotides (also called bases), {A, C, G, T}. Inside a cell, RNAs do not retain a linear form but instead fold in space. The fold is given by the set of nucleotide bases that pair. The main type of pairing, called canonical, corresponds to bonds of the type $A-U$ and $G-C$. Other rarer types of bonds may be observed, most frequent among them is $G-U$, also called the wobble pair. Figure 1 shows the sequence of a folded RNA. Each box represents a consecutive sequence of bonded pairs, corresponding to a helix in 3D space. The secondary structure of an RNA is the set of helices (or the list of paired bases) making up the RNA. Pseudo-knots, which may be described as a pair of interleaved helices, are in general excluded from the secondary structure of an RNA. RNA secondary structures can thus be represented as planar graphs. An RNA primary structure is its sequence of nucleotides while its tertiary structure corresponds to the geometric form the RNA adopts in space.

Apart from helices, the other main structural elements in an RNA are: 1. hairpin loops which are sequences of unpaired bases closing a helix; 2. internal loops which are sequences of unpaired bases linking two different helices; 3. bulges which are internal loops with unpaired bases on one side only of a helix; 4. multi-loops which are unpaired bases linking at least three helices. Stems are successions of one or more among helices, internal loops and/or bulges.

RNA secondary structure comparison is one of the main basic computational problems raised by the study of RNAs. It is the problem we address in this paper. The motivations are many. RNA structure comparison has been used in at least one approach to RNA structure prediction that takes as initial data a set of unaligned sequences supposed to have a common structural core [1]. For each sequence, a set of structural predictions are made (for instance, all suboptimal

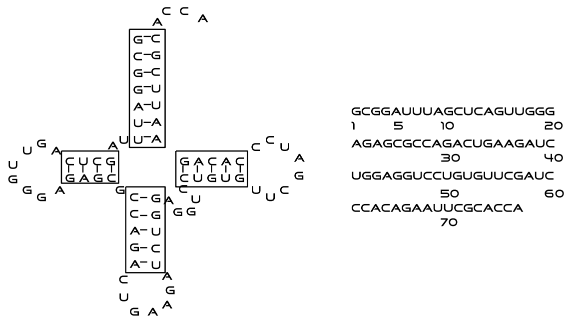


Fig. 1. Primary and secondary structures of a transfer RNA.

structures predicted by an algorithm like Zucker's MFOLD [9], or all suboptimal sets of compatible helices or stems). The common structure is then found by comparing all the structures obtained from the initial set of sequences, and identifying a substructure common to all, or to some of the sequences. RNA structure comparison is also an essential element in the discovery of RNA structural motifs, or profiles, or of more general models that may then be used to search for other RNAs of the same type in newly sequenced genomes. For instance, general models for tRNAs and introns of group I have been derived by hand [3] [5]. It is an open question whether models at least as accurate as these, or perhaps even more accurate, could have been derived in an automatic way. The identification of smaller structural motifs is an equally important topic that requires comparing structures.

As we saw, the comparison of RNA structures may concern *known* RNA structures (that is, structures that were experimentally determined) or *predicted* structures. The objective in both cases is the same: to find the common parts of such structures.

In [6], Shapiro suggested to mathematically model RNA secondary structures without pseudoknots by means of trees. The trees are rooted and ordered, which means that the order among the children of a node matters. This order corresponds to the 5'-3' orientation of an RNA sequence. One way to compare two RNA secondary structures is then to apply a number of tree edit operations in one or both of the trees representing the RNAs until isomorphic trees are obtained. The tree edit operations considered are derived from the operations classically applied to sequences: substitution, deletion and insertion. In 1989, Zhang and Shasha proposed [8] a dynamic programming algorithm for comparing two trees. Shapiro and Zhang then showed [7] how to use tree editing to compare RNAs. The latter also proposed various tree models that could be used for representing RNA secondary structures. Each suggested tree offers a more or less detailed view of an RNA structure. Figure 2 (b) to (e) presents a few examples of such possible views for the RNA given in Figure 2 (a). In Figure 2, the nodes of the tree in (b) represent either unpaired bases (leaves) or paired bases (internal nodes). Each node is labelled with, respectively, a base or a pair of bases. A node of the tree in (c) represents a set of successive unpaired bases or of stacked paired ones. The label of a node is an integer indicating, respectively, the number of unpaired bases or the height of the stack of paired ones. The nodes of the tree in (d) represent elements of secondary structure: hairpin loop (H), bulge (B), internal loop (I) or multi-loop (M). The edges correspond to helices. Finally, the tree in (e) contains only the information concerning the skeleton of multi-loops of an RNA. The last representation, though giving a highly simplified view of an RNA, is important nevertheless as it is generally accepted that it is this skeleton which is usually the most constrained part of an RNA. The last two models may be enriched with information concerning, for instance, the number of (unpaired) bases in a loop (hairpin, internal, multi) or bulge, and the number of paired bases in a helix. The first label the nodes of the tree, the second its edges. Other types of information may be added (such as overall composition of the elements of secondary structure). In fact, one could consider working with various representations simultaneously or in an interlocked, multi-level fashion. This goes beyond the scope of this paper which is concerned with comparing RNA secondary structures using one among the many tree representations possible. We shall however comment further this multi-level approach later on.

Concerning the objectives of this paper, they are twofold. The first is to give some indications on why the classical edit operations that have been considered so far in the literature for comparing trees present some limitations when the trees stand for RNA structures. Three cases of such limitations will be illustrated through examples in Section 3. In Section 4, we then introduce two novel operations, so-called *node-fusion* and *edge-fusion*, that enable us to address some of these limitations and then give a dynamic programming algorithm for comparing two RNA structures with these two additional operations. Implementation issues and initial results are presented in Section 5. Before that, we start by introducing some notation and by recalling in the next section the basics about classical tree edit operations and tree mapping.

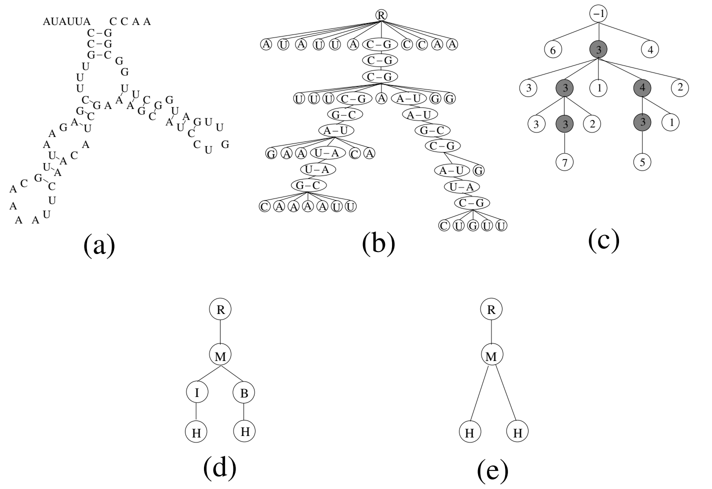


Fig. 2. Example of different tree representations of a same RNA.

2 Tree editing and mapping

Let T be an ordered rooted tree, that is, a tree where the order among the children of a node matters. We define three kinds of operations on T: deletion, insertion and relabelling (corresponding to a substitution in sequence comparison). The operations are shown in Figure 3. The deletion (3 (b)) of a node u removes u from the tree. The children of u become the children of u's father. An insertion (3 (c)) is the symmetric of a deletion. Given a node u, we remove a consecutive (in relation to the order among the children) set $u_1, \ldots, u_p$ of its children, create a new node v, make v a child of u by attaching it at the place where the set was, and, finally, make the set $u_1, \ldots, u_p$ (in the same order) the children of v. The relabelling of a node (3 (d)) consists simply in changing its label.

Given two trees T and T', we define $\mathcal{S} = \{s_1 \ldots s_e\}$ to be a series of edit operations such that if we apply successively the operations in $\mathcal{S}$ to the tree T, we obtain T' (*i.e.*, T and T' become isomorphic). A series of operations like S *realizes the editing* of T into T' and is denoted by $T \xrightarrow{\mathcal{S}} T'$.

We define a function *cost* from the set of possible edit operations (deletion, insertion, relabelling) to the integers (or the reals) such that $cost_s$ is the score of the edit operation s. If $\mathcal{S}$ is a series of edit operations, we define by extension that $cost_{\mathcal{S}}$ is $\sum_{s \in \mathcal{S}} cost_s$. We can define the edit distance between two trees as the series of operations that performs the editing of T into T' and such that its cost is minimal: $distance(T, T') = \{\min(cost_{\mathcal{S}}) | T \xrightarrow{\mathcal{S}} T'\}$.

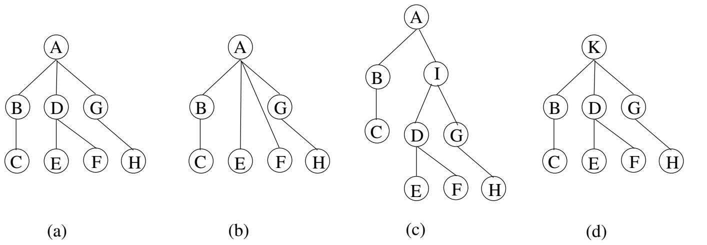
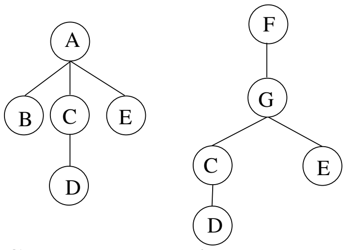


Fig. 3. Edit operations: (a) the original tree T, (b) deletion of node labelled D, (c) insertion of the node labelled I and (d) relabelling of a node in T (the label A of the root is changed into K).



Let an insertion or a deletion cost one and the relabelling of a node cost zero if the label is the same and one otherwise. For the two trees of the figure on the left, the series $relabel(A \rightarrow F).delete(B).insert(G)$ realizes the editing of the left tree into the right one and costs 3. Another possibility is the series $delete(B).relabel(A \rightarrow G).insert(F)$ which also costs 3. The distance between these two trees is 3.

Given a series of operations $\mathcal{S}$, let us consider the nodes of T that are not deleted (in the initial tree or after some relabelling). Such nodes are associated with nodes of T'. The *mapping* $M_{\mathcal{S}}$ relative to $\mathcal{S}$ is the set of couples (u, u') with $u \in T$ and $u' \in T'$ such that u is associated with u' by $\mathcal{S}$.

The operations described above are the "classical tree edit operations" that have been commonly used in the literature for RNA secondary structure comparisons. We now present a few results obtained using such classical operations that will allow us to illustrate a few limitations they may present when used for comparing RNA structures.

3 Limitations of classical tree edit operations for RNA comparison

As suggested in [7], the tree edit operations recalled in the previous section can be used on any type of tree coding of an RNA secondary structure.

Figure 4 shows two RNAsePs extracted from the database [2] (they are found, respectively, in *Thermotoga maritima* and *Streptococcus gordonii*). For the example we discuss now, we code the RNAs using the tree representation indicated in Figure 2 (b) where a node represents a base pair and a leaf an unpaired base. After applying a few edit operations to the trees, we obtain the result indicated in Figure 2, with deleted/inserted bases in grey. We have surrounded a few regions that match in the two trees. Bases in the box at the bottom of the RNA on the left are thus associated with bases in the rightmost box of the RNA on the right. Such matches illustrate one of the main problems with the classical tree edit operations: bases in one RNA may be mapped to identically labelled bases in the other RNA to minimise the total cost, while such bases should not be associated in terms of the elements of secondary structure to which they belong. In fact, such elements are often distant from one another along the common RNA structure. We call this problem the "scattering effect". It is related to the definition of tree edit operations. In the case of this example and of the representation adopted, the problem might have been avoided if structural information had been used. Indeed, the problem appears also because the structural location of an unpaired base is not taken into account. It is therefore possible to match, for instance, an unpaired base from a hairpin loop with an unpaired base from a multi-loop. Using another type of representation, as we shall do, would, however, not be enough to solve all problems as we see next.

Indeed, to compare the same two RNAs, we can also use a more abstract tree representation such as the one given in Figure 2 (d). In this case, the internal nodes represent a multi-loop,

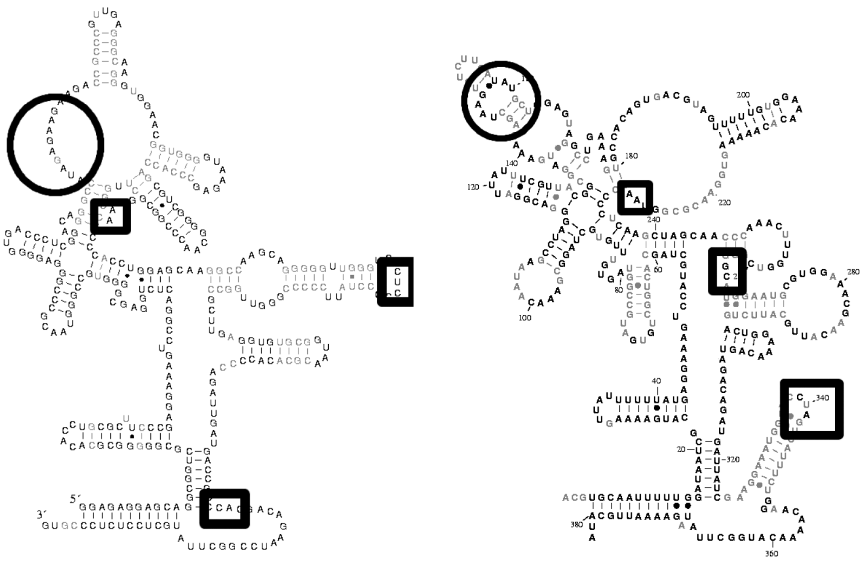


Fig. 4. Result of the matching of the two RNAs in Figure 4 using the model (b) given in Figure 2.

internal-loop or bulge, the leaves code for hairpin loops and edges for helices. The result of the edition of T into T' for some cost function is presented in Figure 5 (we shall come back later to the cost functions used in the case of such more abstract RNA representations; for the sake of this example, we may assume an arbitrary one is used).

The problem we wish to illustrate in this case is shown by the boxes in the figure. Consider the boxes at the bottom. In the left RNA, we have a helix made up of 13 base pairs. In the right RNA, the helix is formed by 7 base pairs followed by an internal loop and another helix of size 5. By definition (see Section 2), the algorithm can only associate one element in the first tree to one element in the second tree. In this case, we would like to associate the helix of the left tree to the two helices of the second tree since it seems clear that the internal loop represents either an inserted element in the second RNA, or the unbonding of one base pair. This, however, is not possible with classical edit operations.

A third type of problem one can meet when using only the three classical edit operations to compare trees standing for RNAs is similar to the previous one but concerns this time a node instead of edges in the same tree representation. Often, an RNA may present a very small helix between two elements (multi-loop, internal-loop, bulge or hairpin-loop) while such helix is absent in the other RNA. In this case, we would therefore have liked to be able to associate one node in a tree representing an RNA with two or more nodes in the tree for the other RNA. Once again, this is not possible with any of the classical tree edit operations. An illustration of this problem is shown in Figure 11 (see Section 5).

We shall use RNA representations that take the elements of the structure of an RNA into account to avoid some of this scattering effect. Furthermore, in addition to considering information of a structural nature, labels are attached, in general to both nodes and edges of the tree representing an RNA. Such labels are numerical values (integers or reals). They represent in most cases the size of the corresponding element, but may also further indicate its composition etc. Such additional information is then incorporated into the cost functions for all three edit operations.

It remains now to deal with the last two problems that are a consequence of the one-to-one associations between nodes and edges enforced by the classical tree edit operations. To that purpose, we introduce two novel tree edit operations, called the *edge fusion* and the *node fusion*.

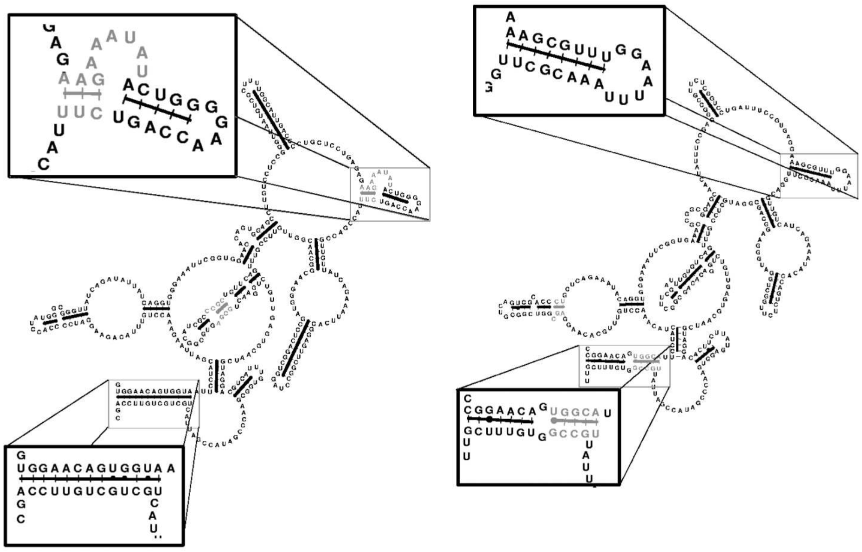


Fig. 5. Result of the matching of the two RNAs using the model (d) given in Figure 2.

4 Introducing novel tree edit operations

4.1 Edge fusion and node fusion

In order to address some of the limitations of the classical tree edit operations that were illustrated in the previous section, we need to introduce two novel operations. These are the *edge fusion* and the *node fusion*. They may be applied to any of the tree representations given in Figure 2(c) to (e).

An example of edge fusion is shown in Figure 6. Let e_u be an edge leading to a node u, c_i a child of u and e_{c_i} the edge between u and c_i. The edge fusion of e_u and e_{c_i} consists in replacing e_{c_i} and e_u with a *new* single edge e. The edge e links the father of u to c_i. Its label then becomes a function of the (numerical) labels of e_u, u and e_{c_i}. For instance, if such labels indicated the size of each element (*e.g.* for a helix, the number of its stacked pairs, and for a loop, the min, max or the average of its unpaired bases on each side of the loop), the label of e could be the sum of the sizes of e_u, u and e_{c_i}. Observe that merging two edges implies deleting all subtrees rooted at the children c_j of u for j different from i. The cost of such deletions is added to the cost of the edge fusion.

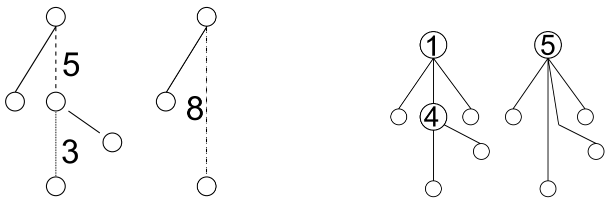


Fig. 6. On the left, an example of edge fusion. On the right, an example of node fusion.

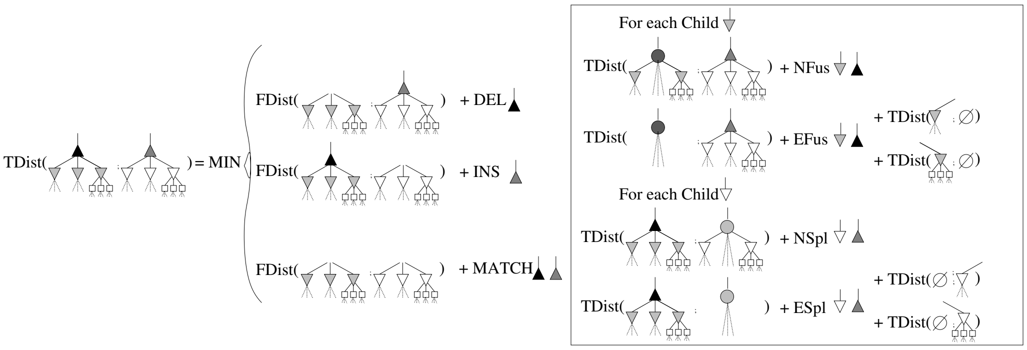


Fig. 7. Zhang and Sasha's dynamic programming algorithm: the tree distance part. The right box corresponds to the additional operations added to take fusion into account.

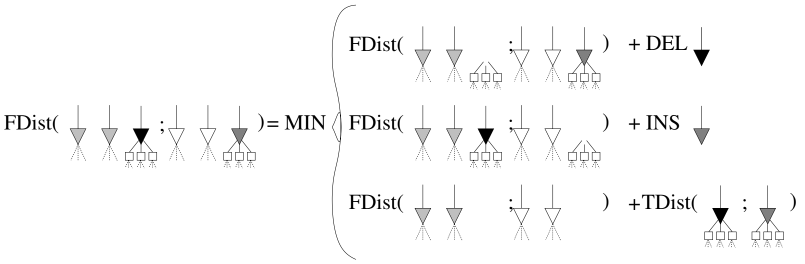


Fig. 8. Zhang and Sasha's dynamic programming algorithm: the forest distance part.

An example of node fusion is given in Figure 6. Let u be a node and c_i one of its children. Performing a node fusion of u and c_i consists in making u the father of all children of c_i and in relabelling u with a value that is a function of the values of the labels of u, c_i and of the edge between them.

Observe that a node fusion may be simulated using the classical edit operations by a deletion followed by a relabelling. However, the difference between a node fusion and a deletion/relabelling is in the cost associated with both operations. We shall come back to this point later.

Obviously, like insertions or deletions, edge fusions and node fusions have of course symmetric conuterparts, which are the *edge split* and the *node split*.

We now present an algorithm to compute the tree edit distance between two trees using the classical tree edit operations plus the two operations just introduced.

4.2 Algorithm

The method we introduce is a dynamic programming algorithm based on the one proposed by Zhang and Shasha. Their algorithm is divided in two parts: they first compute the edit distance between two trees (this part is denoted by $TDist$) and then the distance between two forests (this part is denoted by $FDist$). Figure 7 illustrates in pictorial form the part $TDist$ and Figure 8 the $FDist$ part of the computation.

In order to take our two new operations into account, we need to compute a few more things in the $TDist$ part. Indeed, we must add the possibility for each tree to have a node fusion (inversely, *node split*) between the root and one of its children, or to have an edge fusion (inversely *edge split*) between the root and one of its children. These additional operations are indicated in the right box of Figure 7.

We present now a formal description of the algorithm. Let T be an ordered rooted tree with $|T|$ nodes. We denote by t_i the i^{th} node in a postfix order. For each node t_i, $l(i)$ is the index of the leftmost child of the subtree rooted at t_i. Let $T(i \dots j)$ denote the forest composed by the nodes $t_i \dots t_j$ ($T \equiv T(0 \dots |T|)$). To simplify notation, from now on, when there is no ambiguity, i will refer to the node t_i. In this case, $distance(i_1 \dots i_2, j_1 \dots j_2)$ will be equivalent to $distance(T(i_1 \dots i_2), T'(j_1 \dots j_2))$.

The algorithm of Zhang and Sasha is fully described by the following recurrence formula:

$$\boxed{distance(i_1 \dots i_2, j_1 \dots j_2) =}$$

if $((i_1 == l(i_2))$ and $(j_1 == l(j_2)))$

$$MIN \begin{cases} distance(\ i_1 \dots i_2 - 1\ , j_1 \dots j_2\ ) + cost_{del}(i_2) \\ distance(\ i_1 \dots i_2\ , j_1 \dots j_2 - 1\ ) + cost_{ins}(j_2) \\ distance(\ i_1 \dots i_2 - 1\ , j_1 \dots j_2 - 1\ ) + cost_{match}(i_2, j_2) \end{cases} \tag{1}$$

else

$$MIN \begin{cases} distance(\ i_1 \dots i_2 - 1\ , j_1 \dots j_2)\ ) + cost_{del}(i_2) \\ distance(\ i_1 \dots i_2)\ , j_1 \dots j_2 - 1\ ) + cost_{ins}(j_2) \\ distance(\ i_1 \dots l(i_2) - 1\ , j_1 \dots l(j_2) - 1\ ) \\ \quad +distance(\ l(i_2) \dots i_2\ , l(j_2) \dots j_2\ ) \end{cases} \tag{2}$$

Part (1) of the formula corresponds to Figure 7 while part (2) corresponds to Figure 8. In practice, the algorithm stores in a matrix the score between each subtree of T and T'. The space complexity is therefore $O(|T| * |T'|)$. To reach this complexity, the computation must be done in a certain order (see [8] for further details). The time complexity of the algorithm is $O(|T| * min(leaf(T), height(T)) * |T'| * min(leaf(T'), height(T')))$ where $leaf(T)$ and $height(T)$ represent, respectively, the number of leaves and the height of a tree T.

Follows now the formula to compute the edit score allowing for both node and edge fusions.

$$\boxed{distance(\{i_1, \dots, i_k\}, path, \{j_1, \dots, j_{k'}\}, path') =}$$

if $((i_1 \geq l(i_k))$ and $(j_1 \geq l(j_{k'})))$

$$MIN \begin{cases} distance(\ \{i_1 \dots i_{k-1}\}\ , \emptyset\ , \{j_1 \dots j_{k'}\}\ , path'\ ) + cost_{del}(i_k) \\ distance(\ \{i_1 \dots i_k\}\ , path\ , \{j_1 \dots j_{k'-1}\}\ , \emptyset\ ) + cost_{ins}(j_{k'}) \\ distance(\ \{i_1 \dots i_{k-1}\}\ , \emptyset\ , \{j_1 \dots j_{k'-1}\}\ , \emptyset\ ) + cost_{match}(i_k, j_{k'}) \\ for\ each\ child\ i_c\ of\ i_k\ in\ \{i_1, \dots, i_k\},\ set\ i_l = l(i_c) \\ \quad distance(\{i_1 \dots i_{c-1}, i_{c+1} \dots i_k\}, path.(u, i_c), \{j_1 \dots j_{k'}\}, path') \\ \quad\quad +cost_{node\_fusion}(i_c, i_k)\ \textbf{note: } i_k \text{ data are changed} \\ \quad distance(\{i_l \dots i_{c-1}, i_k\}, path.(e, i_c), \{j_1 \dots j_{k'}\}, path') \\ \quad\quad +cost_{edge\_fusion}(i_c, i_k) + distance(\{i_1 \dots i_{l-1}\}, \emptyset, \emptyset, \emptyset) \\ \quad\quad +distance(\{i_{c+1} \dots i_k - 1, \emptyset, \emptyset, \emptyset)\ \textbf{note: } i_k \text{ data are changed} \\ for\ each\ child\ j_{c'}\ of\ j_{k'}\ in\ \{j_1, \dots, j_{k'}\},\ set\ j_{l'} = l(j_{c'}) \\ \quad distance(\{i_1 \dots i_k\}, path, \{j_1 \dots j_{c'-1}, j_{c'+1} \dots j_{k'}, path'.(u, j_{c'})) \\ \quad\quad +cost_{node\_split}(j_{c'}, j_{k'})\ \textbf{note: } j_{k'} \text{ data are changed} \\ \quad distance(\{i_1 \dots i_k\}, path, \{j_{l'} \dots j_{c'}, j_{k'}, path'.(e, j_{c'})) \\ \quad\quad +cost_{edge\_split}(j_{c'}, j_{k'}) + distance(\emptyset, \emptyset, \{j_1 \dots j_{l'-1}\}, \emptyset) \\ \quad\quad +distance(\emptyset, \emptyset, j_{c'+1} \dots j_{k'-1}, \emptyset)\ \textbf{note: } j_{k'} \text{ data are changed} \end{cases} \tag{3}$$

else $set\ i_l = l(i_k)\ and\ j_{l'} = l(j_{k'})$

$$MIN \begin{cases} distance(\ \{i_1 \dots i_{k-1}\}\ , \emptyset\ , \{j_1 \dots j_{k'}\}\ , path'\ ) + del(i_k) \\ distance(\ \{i_1 \dots i_k\}\ , path\ , \{j_1 \dots j_{k'-1}\}\ , \emptyset\ ) + ins(j_{k'}) \\ distance(\ \{i_1 \dots i_{l-1}\}\ , \emptyset\ , \{j_1 \dots j_{l'-1}\}\ , \emptyset\ ) \\ \quad +distance(\ \{i_l \dots i_k\}\ , path\ , \{j_{l'} \dots j_{k'}\}\ , path'\ ) \end{cases} \tag{4}$$

Given two nodes u and v such that v is a child of u, $node\_fusion(u, v)$ is the fusion of node v with u and $edge\_fusion(u, v)$ is the edge fusion between the edges leading to, respectively, nodes u and v. The symmetric operations are denoted by, respectively, $node\_split(u, v)$ and $edge\_split(u, v)$.

The distance computation takes two new parameters $path$ and $path'$. These are sets of pairs (e or u, v) which indicate, for node i_k (resp. j_k), the series of fusions that were done. Thus a pair (e, v) indicates that an edge fusion has been perfomed between i_k and v, while for (u, v) a node v has been merged with node i_k. The notation $path.(e, v)$ indicates that the operation (e, v) has been performed in relation to node i_k and the information is thus concatenated to the set $path$ of pairs currently linked with i_k.

Observe that the computation of the forest distance does not change in relation to the original algorithm. We therefore need a matrix to store all edit scores between $(i, path)$ and $(j, path')$.

Let d be the maximum degree of the two trees. Then each node has $O((2d)^l)$ different ways of participating in l **consecutive** fusions (that is, in fusions with successive nodes along a same branch). If we limit the number of consecutive fusions to l, the total memory complexity of our algorithm is $O(n(2d)^l)$. By using the same arguments as in [8], we prove that the time complexity of the new algorithm to be $O((2d)^l * |T| * min(leaf(T), height(T)) * |T'| * min(leaf(T'), height(T')))$.

This complexity suggests that the fusion operations may be used only for reasonable trees (typically, less than 100 nodes) and small values of l (typically, less then 4). It is however important to observe that the overall number of fusions one may perform can be much greater than l without affecting the worst-case complexity of the algorithm. Indeed, any number of fusions can be made while still retaining the bound of $O((2d)^l * |T| * min(leaf(T), height(T)) * |T'| * min(leaf(T'), height(T')))$ so long as one does not realize more than l consecutive fusions for each node.

In general also, most interesting tree representations of an RNA are of small enough size as will be shown next, together with some initial results obtained in practice.

5 Implementation and results

The algorithm presented in the previous section has been coded using C++. An online version is available at http://www-igm.univ-mlv.fr/~allali/migal/.

We recall that RNAs are relatively small molecules with sizes limited to a few kilobases. For instance, the small ribosomal subunit of *Sulfolobus acidocaldarius* (D14876) is made up of 1147 bases. Using the representation shown in Figure 2 (b), the tree obtained contains 440 internal nodes and 567 leaves, that is 1007 nodes overall. Using the representation 2 (d), the tree is composed of 78 nodes. Finally, the tree obtained using the representation given in Figure 2 (e) contains only 48 nodes. We therefore see that even for large RNAs, any of the known abstract tree-representations (that is, representations which take the elements of the secondary structure of an RNA into account) that we can use leads to a tree of manageable size for our algorithm. In fact, for small values of l (2 or 3), the tree comparison takes reasonable time (a few minutes) and memory (less than 1Gb).

As we already mentioned, a fusion (resp. split) can be viewed as an alternative to a deletion (resp. insertion) followed by a relabelling. Therefore, the cost function for a fusion must be chosen carefully.

Let us assume that the cost function returns a real value between zero and one. If we want to compute the cost of a fusion between two nodes u and v, the aim is to give to such fusion a cost slightly greater than the cost of deleting v and relabelling u; that is, we wish to have $cost_{node\_fusion}(u, v) = min(del(v) + t, 1)$. The parameter t is a *tuning parameter* for the fusion. Suppose that the new node w resulting from the fusion of u and v matches with another node z. The cost of this match is $cost_{match}(w, z)$. If we do not allow for node fusions, the algorithm will first match u with z, then will delete v. If we compare the two possibilities, on one hand we have a total cost of $cost_{node\_fusion}(u, v) + cost_{match}(w, z)$ for the fusion, that is $del(v) + t + cost_{match}(w, z)$, on the other hand a cost of $del(v) + cost_{match}(u, z)$. Thus t represents the gain that must be obtained by $cost_{match}(w, z)$ with regard to $cost_{match}(u, z)$, that is by a match without fusion. This is illustrated in Figure 9.

The tuning parameter t is thus an important parameter that allows us to control fusions. Always considering a cost function that produces real values between 0 and 1, if t is equal to 0.1,

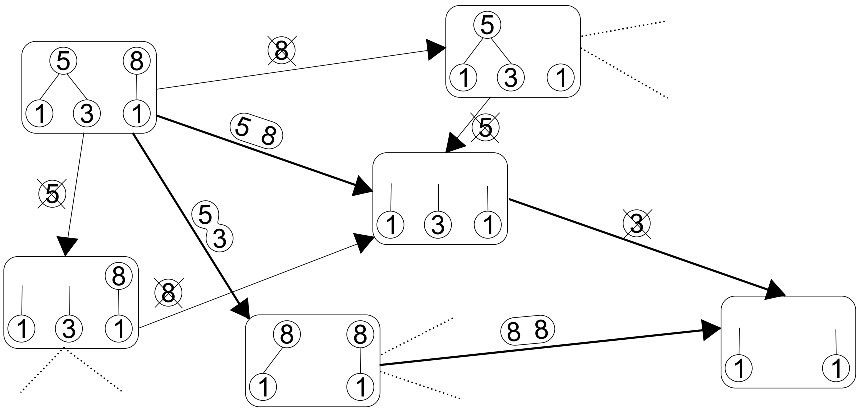


Fig. 9. Illustration of the gain that must be obtained using a fusion instead of a deletion/relabelling.

a fusion will be performed only if it improves the score by 0.1. In practice, we use values of t between 0 and 0.2.

We have applied the new algorithm on the two RNAs shown in Figure 5 (these are eukaryotic nuclear P RNAs from *Saccharomyces uvarum* and *Saccharomyces kluveri*) and coded using the same type of representation as in Figure 2 (d). We have limited the number of consecutive fusions to one ($l = 1$). The computation of the edit distance between the two trees taking node and edge fusions into account besides deletions, insertions and relabelling has taken less than a second. The total cost allowing for fusions is 6.18 with $t = 0.05$ against 7.42 without fusions. As indicated in Figure 10, the last two problems discussed in Section 3 disappear thanks to some edge fusions (represented by the boxes).

An example of node fusions required when comparing two "real" RNAs is given in Figure 11. The RNAs are coded using the same type of representation as in Figure 2 (d). The figure shows part of the mapping obtained between the small sub-units of two ribosomal RNAs retrieved from [4] (from *Bacillaria paxillifer* and *Calicophoron calicophorum*). The node fusion has been circled.

6 Further work and conclusion

We have proposed an algorithm that addresses two main limitations of the classical tree edit operations for comparing RNA secondary structures. Its complexity is high in theory if many fusions are applied in succession to any given (the same) node, but the total number of fusions that may be performed is not limited. In practice, the algorithm is fast enough for most situations one can meet in practice.

To provide a more complete solution to the problem of the scattering effect, we propose the following scheme. Given an RNA, we build four tree representations for it instead of just one. Each tree corresponds to a different level of abstraction of the secondary structure of the RNA. The higher level represents the multi-loop skeleton while the lower level represents the sequence of paired and unpaired bases. For each tree except the one at the lowest level, there is a one-to-many relation between nodes, that is, a node of the tree at a given level is related with at least one node in the tree at the levels below. Two elements at the lowest level of abstraction, that is, two bases or base pairs can be matched only if they are part of a same structural element, the correspondence between such elements having being identified at a previous step by using the algorithm for comparing two structural tree representations introduced in this paper. Such approach thus allows us to address the problem of the scattering effect in an efficient way. A full description and practical consequences of such 4-level comparison method will be the subject of another paper.

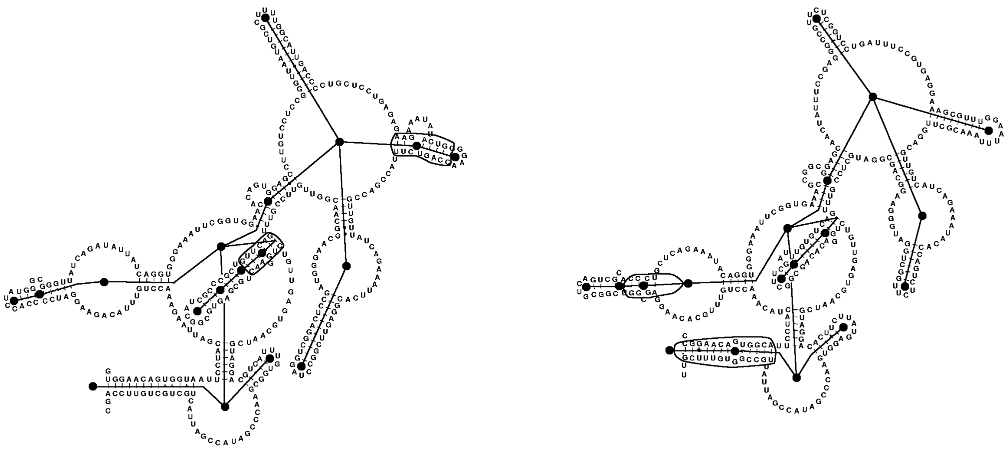


Fig. 10. Result of the editing between the two RNAs shown in the Figure 4 allowing for node and edge fusions. The boxes represent edge fusions.

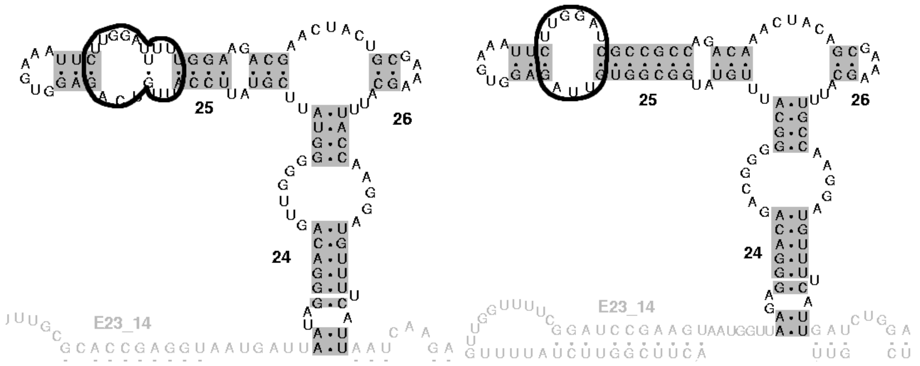


Fig. 11. Part of a mapping between two rRNA small sub-units. The node fusion is circled.

References

1. D. Bouthinon and H. Soldano. A new method to predict the consensus secondary structure of a set of unaligned RNA sequences. *Bioinformatics*, 15(10):785–798, 1999.
2. James W. Brown. The ribonuclease p database. *Nucleic Acids Research*, 24(1):314, 1999.
3. N. el Mabrouk and F. Lisacek. Very fast identification of RNA motifs in genomic DNA. application to tRNA search in the yeast genome. *J. Mol. Biol.*, 264(1):46–55, 1996.
4. T. Winkelmans J. Wuyts, Y. Van de Peer and R. De Wachter. The european database on small subunit ribosomal rna. *Nucleic Acids Research*, 30(1):183–185, 2002.
5. F. Lisacek, Y. Diaz, and F. Michel. Automatic identification of group I intron cores in genomic DNA sequences. *J. Mol. Biol.*, 235(4):1206–1217, 1994.
6. B. Shapiro. An algorithm for multiple rna secondary structures. *Comput. Appl. Biosci.*, 4(3):387–393, 1988.
7. B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6(4):309–318, 1990.
8. 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.
9. M. Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.*, 31(13):3406–3415, 2003.