Introduction to RNA secondary structure comparison

Stefanie Schirmer, Yann Ponty, and Robert Giegerich

Abstract Many methods have been proposed for RNA secondary structure comparison, and new ones are still being developed. In this chapter, we first consider structure representations and discuss their suitability for structure comparison. Then,we take a look the more commonly used methods, restricting ourselves to structures without pseudoknots. For comparing structures from of the same sequence, we study base pair distances. For structures of different sequences (and of different length), we study variants of the tree edit model. We name some of the available tools, and give pointers to the literature. We end with a short review on comparing structures with pseudoknots as an unsolved problem and topic of active research.

Key words: RNA structure comparison, basepair distance, tree edit distance, tree alignment distance, forest alignment

Stefanie Schirmer
Institute for Research in Immunology and Cancer (IRIC), Department of Computer Science and Operations Research, Université de Montréal, Montréal, Québec H3C 3J7, Canada, e-mail: stefanie.schirmer@umontreal.ca

Yann Ponty
CNRS, Laboratoire d'Informatique de l'X (LIX) UMR 7161, INRIA AMIB, Ecole Polytechnique, Palaiseau, France, e-mail: yann.ponty@lix.polytechnique.fr

Robert Giegerich
Faculty of Technology and Center of Biotechnology, Bielefeld University, Bielefeld, Germany, e-mail: robert@techfak.uni-bielefeld.de

1 Introduction

In many chapters of this book, we study the problem of *assigning* secondary structure to RNA sequences. In the present chapter, we consider this problem solved. We are given a set of RNA sequences, each together with one or more secondary structures. We do not care about their origin – they could be predicted by one of the algorithms in the other chapters, they could sampled from the folding space, derived from resolved 3D structures, or even generated as a random test set. The question at hand is how to compare these structures. Our motivation might be to cluster them into families, or to evaluate a new prediction tool against others or against a reference data base.

In principle, all questions we deal with in sequence comparison re-pose themselves with structures: Pairwise (global) alignment, best local alignment, finding the best fit of a small structure in a larger one. Naturally, algorithms for structure comparison are different from those used in sequence comparison, but in a very systematic way. Sequences have a one-dimensional principle of organization: adjacency. One character follows the other. Structures have two dimensions: adjacency and inclusion. Unless we have pseudo-knots, two RNA helices are either adjacent or one includes the other. Where each alignment step with two sequences recurs on a single subproblem of aligning two suffixes of the sequences, each alignment step with structures creates two subproblems – at least. So, even if you think sequence comparison is a worn-out and dull topic algorithmically, you might find structure comparison quite interesting.

In fact, RNA structure comparison has been a creativity parlor for computer scientists. Fancy representations of RNA structures have been suggested and algorithms working thereon have been proposed. In this chapter, we focus on a small number of approaches which have been widely used and which, in our humble opinion, essentially solve the basic problem to satisfaction – for structures without pseudo-knots.

The structure of this chapter is as follows:

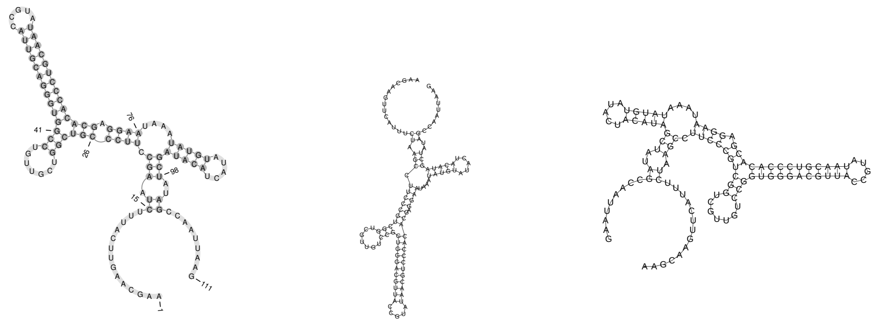
1. We begin with a section on structure representation, as it is the representation on which our algorithms work. Essentially, we argue that representation should be spartanic, and not introduce extra detail that is irrelevant and may confuse the comparison.
2. Then, we focus on the problem of (global) structure comparison. We first discuss the simple scenario, where we compare structures from the folding space of a single sequence. This is the easier problem, as all structures have the same size, and the i-th residue means the same base for each structure.
3. We then study the general problem: comparing arbitrary structures from different sequences of varying length. We present two alternative methods, and report on programs that implement them. At the end of this section, we discuss some variations of these problems and algorithms.
4. In the concluding section of this chapter, we report on the difficulties encountered with the comparison of pseudo-knotted structures, and give hints to the literature.

2 A few words on secondary structure representation

In this chapter, we can safely assume that our reader is familiar with the notions of RNA secondary structure, and has already encountered various forms of structure representation. Here, we discuss representations from the viewpoint of their suitability for structure comparison, both by human experts and by computer programs.

2.1 Aah, squiggle plots!

The most popular representation of RNA secondary structure, and one of the most intuitive, is the so-called *squiggle plot*. It provides a graphical 2D layout of the helices and loops that make up a secondary structure. Even when structures are quite large, these plots give us a good overview of characteristic structural features.



(a) An RNA secondary structure.

(b) Another secondary structure?

(c) Yet another secondary structure?

Fig. 1: Three squiggle plots of the same secondary structure, drawn by different programs. (a) drawn by pseudoviewer [11], (b) drawn by RNAplot [21] with the naview layout option, (c) drawn by RNAplot with the simple radial layout option. Note that not only the layout, but also the orientation changes between clockwise and counter-clockwise.

However, squiggle plots have a shortcoming when it comes to structure comparison. Each structure can be drawn in many different ways. In Figure 1, you see three plots of the same structure, produced by different algorithms. Why are they different? A graphical visualization leaves much room for choice. The layout of helices asks for an aesthetic design, the angles chosen between branching helices in a multiloop are designed to avoid overlapping of substructures. The drawing of the external loop akin to a hairpin loop in a circular fashion, as you see it in Figure 1, is a bit of a questionable convention. There is nothing to bring the 5' and the 3' nucleotide into vicinity. So, structure drawings are good for comparison by human inspection only if they are produced by the same algorithm, of if their layout has been engineered by a human expert to exhibit their similarity. After structures have been aligned, visualizing the alignment with a graphics tool (such as *VARNA* [12]) is quite useful for the human inspector. So, while using such plots as computer *input* for structure comparison does not seem a good idea, many programs produce them as output. For embedding such a program in software pipeline, a second, more computer oriented representation must be offered.

2.2 Dot-bracket representation

A good compromise between human readability and suitability for program input is the *dot-bracket representation*, popularized, for example, by the *ViennaRNA* package [21]. Most commonly, each secondary structure of length n is represented as a sequence of length n that consists of parentheses and dots. Each base pair (i, j) in the structure is represented by a pair of opening and closing parenthesis at the i-th and j-th position. Each unpaired base is represented as a dot. An example is shown below, and we are sure you have seen many of those in the other chapters. Some people use angle or curly brackets rather than the round parentheses, and when different types of brackets are combined, pseudo-knots can be represented as well. One nice feature is that, written below the RNA sequence, the dot-bracket string annotates it with structure. This also works for structural alignments, where sequence and structure strings are padded with gap symbols in the same way. A second advantage of dot-bracket representation is that it does *not* show the concrete bases. Hence, it represents a structure *per se*, and can be associated with any RNA sequence of the same length – suitable bases for matching brackets provided.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$S_1 =$	C	A	G	C	U	C	U	C	A	U	G	U	C	C	C	A	C	A	T	A
$S_2 =$	C	A	G	C	U	C	U	G	U	G	U	C	C	C	A	C	A	C	A	A
$D_{S_1} =$	.	.	.	.	.	.	.	.	(	(	(	(	.	.	.	)	)	)	)	.
$D_{S_2} =$	.	.	.	.	.	.	.	(	(	(	(	.	.	.	)	)	)	)	.	.

Structures not extending beyond one line can well be read and compared by a human, given some practice. But one or two line breaks already destroy much of the readability. A caveat is the following. While the overall arrangement of helices is fairly easy to see, it is difficult to check if the opening and closing parentheses actually match up well. It requires a little parsing algorithm to check for this property. Do not forget to include this check while writing a program that takes dot-bracket strings as input.

2.3 Basepair set representation

For mathematical purposes, a secondary structure is often defined as a set of base pairs, or more precisely, the set of base paired positions in the primary sequence.

```
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
S1 =  C A G C U C U C A U  G  U  C  C  C  A  C  A  T  A
S2 =  C A G C U C U G U G  U  C  C  C  A  C  A  C  A  A
```

$$B_{S_1} = \{\{9,19\},\{10,18\},\{11,17\},\{12,16\}\}$$
$$B_{S_2} = \{\{8,18\},\{9,17\},\{10,16\},\{11,15\}\}$$

This representation is quite impractical for human inspection, and for computer input and output, the mathematical *set* should be represented by an *sorted list* to make comparison easy.

2.4 Tree representations

The *tree representations* constitute a class of representations that may differ in their level of detail. Generally, a secondary structure is represented by an ordered rooted tree. Trees mathematically encompass the two organization principles of adjacency and inclusion. A *rooted ordered tree* consists of a root node and a forest of subtrees (inclusion). A *forest* is a possibly empty sequence of trees (adjacency). We normally speak of tree representation, but – given that a tree node has a forest of subtrees – speaking of forest representations would be at least as adequate. Note that there are no empty trees, while forests can be empty.

Figure 2 shows one out of many tree representations. This one combines sequence and structure representation. At the leaves of the tree, we find the primary sequence, in 5' to 3' order. A node labeled P denotes a base pair bond between the two bases represented as its left- and rightmost child.

From what we said about squiggle plots, you might expect that we consider trees as unsuitable for communication between programs producing and consuming or comparing structures. But this is not the case, since a trees can be encoded by straightforward formula, allowing for an easy manipulation. For example,
$P(C,P(C,P(G,P(U,P(C,P(G,P(G,U,C,G,U,U,G,U,C),C),G),P(G,P(U,$
$\quad P(G,P(G,P(G,P(A,P(C,P(G,P(U,P(U,P(A,C,C,G,U,A,U)A),A),$
$\qquad C),G),U),C),C),C),A),C),A),C),G),A,G)$
encodes the tree of Figure 2.

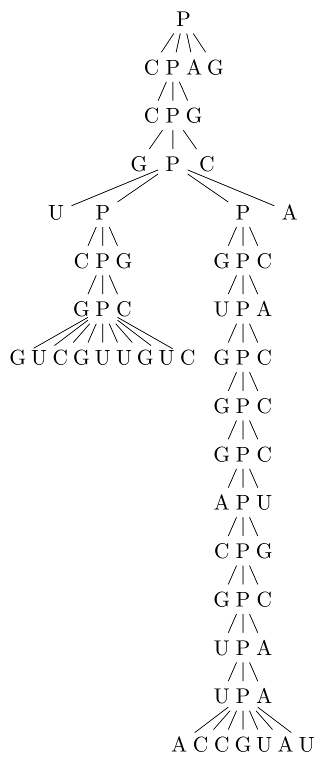
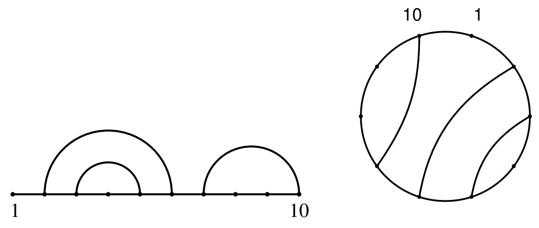


Fig. 2: A tree representation of a y-shaped substructure of the structures plotted in Figure 1, including residues 25 – 73



(a) Arc-annotated sequence (b) Circle representation

Fig. 3: Two equivalent representations, except for their layout ...

2.5 Further representations

2.5.1 Arc-annotated sequences

Consider Figure 3. Depending which lines we draw straight, we have a *circle plot* or an *arc-annotated sequence*. You can also see the latter as a tree, where the leaves are written on a line and the P-nodes have become the arcs. If so, these arcs never cross. But here is the important difference that makes arc-annotated sequences interesting: Once we do allow to connect arbitrary bases by arcs, we can represent more general structures as well. We may allow crossing arcs leading to pseudo-knots, and bases incident to two arcs, modeling triple base interactions. Most of the recent work that considers pseudo-knotted structures uses the arc-annotated sequence representation.

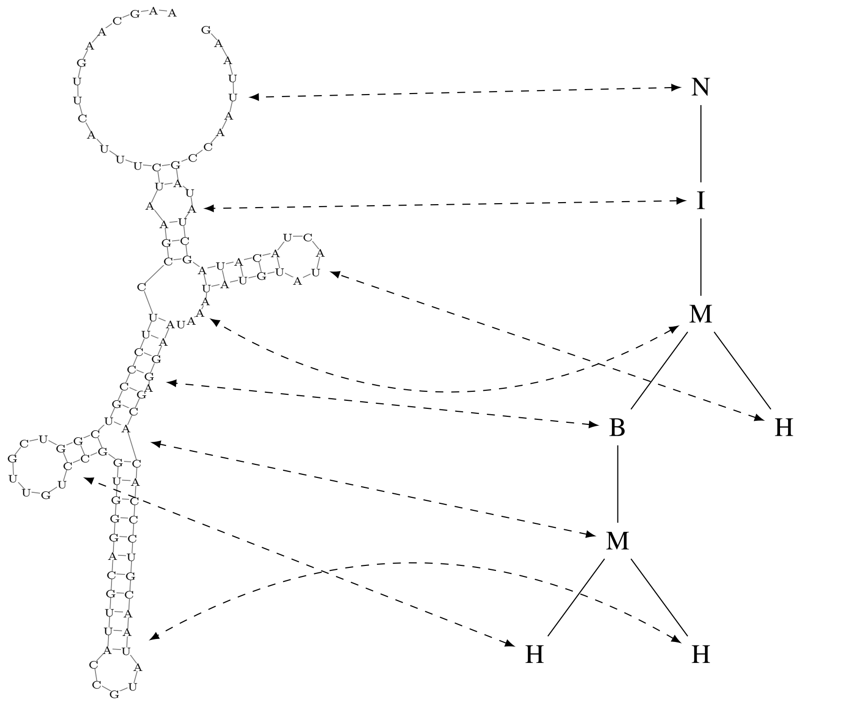
2.5.2 Coarse graining

Sometimes when we ask for structural similarity, we do not care about resolution on the base pair level. Trees nicely lend themselves also to coarse-grained structure representations. For example, a tree representation can also be reduced to a homeomorphic irreducible tree (HIT) representation, in which all sequences of internal nodes which have only one child are contracted into one node. A weight associated with it might reflect the number of nodes that are combined into this one node ([15, 38]). A coarse grained tree representation, as we show it in in Figure 4 , indicates structural elements, such as stems, hairpin loops, internal loops, bulges, multibranch loops (multiloops). A fictitious root (labeled N in Figure 4) node is added sometimes. It does not correspond to a structural element, but ensures the formation of a tree (preventing a forest).

Many similar representations are found in the literature. In [37], stems are not included as nodes, but just loops. In [38], the stems may be included and the stem loop size can be used to influence the score function.

Another coarse grain representation are abstract shapes [16], see also Chapter 11. A Y-shaped structure is denoted `"[[][]]"`, a cloverleaf is `"[[][][]]"`. Shapes can also cover different levels of abstraction. If the presence of bulges is to be indicated, the representation could be `"[[_[]_]_]"`, where the underscores indicate that this simple stem-loop structure has an internal loop and a bulge on the 3’ side. Is is easy to convert an abstract shape string into a tree, and vice versa, so there is not much essential difference between shapes and other coarse grain representations.

Coarse graining can be used to organize structure comparison in a hierarchic fashion, proceeding from more abstract to more concrete representations [4]. Coarse graining is also useful when we deal with large numbers of pairwise structure comparisons. Often, when there is no match on the coarse grain structure, a detailed (and more expensive) comparison of two structures can be skipped. For example, Rfam families have been shown to be well filtered by their spectrum of abstract shapes [22].



(a) RNA secondary structure (b) The coarse tree representation

Fig. 4: Coarse grained tree representation, which represents an RNA secondary structure as a tree of structural building blocks such hairpin loops (H), multiloops (M), bulges (B), internal loops (I). Node N does not represent a structural element, it closes the secondary structure and makes sure the representation forms a tree.

2.5.3 Representing sets of structures

Comparing two sets of structures, not element-wise but in their compact representation, is an interesting problem. When considering the folding spaces of two sequences, rather than just the two minimum-free-energy structures, the *CONSTRUCT* tool allows to interactively align two sequences based on their base pair probability dotplots [43]. When it comes to representing sets of structures from different sequences, there is probably no better way to do this than designing a context free grammar (or a tree grammar) that generates exactly the desired set. In this book, this is done in the chapter on stochastic grammars and RNA family models. Another recent step in this direction is an approach that compares two covariance models [46].

3 Comparing of structures from the same sequence

Comparing structures from the folding space of one sequence is an simple but important special case of structure comparison, which we cover first. When we deal with a single sequence, there is no need of inserting gaps to exhibit similarity. The methods described here are also applicable for structures of closely related sequences of the *same* length, a situation which may arise, for example, in the analysis of the impact of SNPs on the folding space of an RNA molecule.

For the following explanation, let B_1 and B_2 denote two alternative structures of an RNA sequence.

3.1 Base pair distance

Base pair sets are a useful representation in the present case, as ”residue i” means the same nucleotide for all compared structures. Methods for comparing these sets can be transferred from set theory. The *symmetric set difference* is a good first approach to evaluate difference of structures:

Definition 1. The naive *base pair distance* d_{BP} is the cardinality of the symmetric difference between the sets of base pairs B_1 and B_2,

$$d_{BP}(B_1,B_2) = |\,(B_1 \backslash B_2) \cup (B_2 \backslash B_1)\,|.$$

where a base pair is given as a pair of positions in the sequence.

Hence, we count the base-pairs present in either of the structures, but not in both. This naive base pair distance is a metric. This metric is very strict. All differences have the same weight. Consider as examples the base pair sets B_1 and B_2 we encountered in Section 2.3. The two structures are considered as different as can be, $D_{bp}(B_1,B_2) = 8$, as *all* base pairs are different. Intuitively, one would say that the two structures are not that different, as both have the same number of base pairs, in the same arrangement, but shifted one position. These concerns are addressed by variations of the base pair distance, such as our next refinement.

3.2 Hausdorff distance

The Hausdorff distance measures the distance between sets of points. It captures the “maximum distance of a point in a set to the nearest point in the other set”. The Hausdorff distance is a classic maximin function, which fulfills metric properties. Applying the Hausdorff distance to RNA secondary structures means that we interpret their base pair positions as coordinates in a 2D plane. A peculiarity: The sets of base pairs must not be empty, and hence this distance measure is not applicable to the empty (unfolded) structure.

Definition 2. The *Haussdorff distance d_H* is defined in three steps. First, the distance between two base pairs $(i, j) \in B_1$ and $(i', j') \in B_2$ is defined as

$$d\left((i,j),(i',j')\right) = \max\left(|i-i'|,|j-j'|\right).$$

We next formulate the distance between a base pair and a set:

$$d_a\left((i,j),B_2\right) = \min_{(i',j')\in B_2} d\left((i,j),(i',j')\right)$$

Then, the asymmetric distance d_a between two sets of base pairs is defined as:

$$d_a\left(B_1,B_2\right) = \max_{(i,j)\in B_1}\ \min_{(i',j')\in B_2} d\left((i,j),(i',j')\right).$$

Establishing symmetry gives the final definition of distance d_H:

$$d_H\left(B_1,B_2\right) = \max\left(d_a(B_1,B_2), d_a(B_2,B_1)\right).$$

For our examples used above, we find $D_H(B_1,B_2) = 1$.

The metric d_h can deal reasonably well with shifted base pairs, but also has its weakness. Differences in isolated base pairs can lead to very small or very high distance values depending on the distance to the next base pair. Inspired by the *Hausdorff metric*, a further refined distance d_Z on RNA secondary structures was developed by Zuker et al. [47] to circumvent the problem with isolated base pairs. The distance d_Z was used to filter very similar foldings from the output of near-optimal structures in the early *Mfold* program.

4 Comparing structures from different sequences

In this section, we turn to methods suitable to compare structures of RNA molecules of different primary sequence and, in most cases, of different length. These methods are more general than the base pair metrics, and they are more expensive to compute. We start with a warning about simple ideas that seem plausible, but do not work well. Thereafter, we turn to two models of comparing structures represented as trees: the tree edit and the tree alignment model.

4.1 Ideas that are too simple ...

What do we expect from a general method of structure comparison? Here are two examples:

- Assume we are comparing two tRNA structures. One is the classical four-stem cloverleaf structure, the other exhibits a fifth stem that is observed with certain tRNAs. A good comparison method should be able to match up the “standard” stems and show the extra stem as an insertion.
- Assume we are comparing 5’ and 3’ UTRs of genes. We expect to find there a large number of small hairpins, about ten base pairs and a short hairpin loop each. If among them there are iron responsive elements – characterized as small hairpins with a bulged-out ’C’ nucleotide, we hope that a good comparison method would allow to match them up automatically.

Hence, a general comparison should allow for sequence insertions and deletions, as well as replacements of bases that preserve the base pairs in the structure. This general problem is tackled by the tree edit models studied in the next two sections. The methods we discuss in the subsequent sections have been shown to master these challenges.

Before we start, let us discuss why we cannot simply use sequence comparison methods. After all, we can represent structures as (dot-bracket) sequences! People familiar with sequence alignment might approach the problem by aligning dot-bracket strings. This might work for *very* similar structures, but in general, strange things will happen.

Consider an example such as

```
((((...))))...((((...))))
((((.................))))
```

Although the shown alignment is certainly an optimal *string* alignment, it is not an alignment that is consistent with structure. Left and right brackets that constitute a base pair in the lower sequence are aligned to left and right brackets in the upper structure – but these do not make up a base pair there! Sequences represent the principle of adjacency, but not that of inclusion. There is no way a sequence alignment algorithm can look back to the right place before aligning two closing brackets. People have tried to overcome this by using a richer sequence alphabet, where `M` denotes a multiloop, `B` a bulge, etc. It does not help. The general message is: No matter how we encode a structure in a string representation, a sequence alignment method will run into cases where it aligns parts of the sequence that do not coincide in structure. We need to make alignment algorithms work on trees.

4.2 The tree edit model

The tree edit model requires finding a series of edit operations that transforms one input tree into the other with minimum overall cost, defined as the accumulated cost of the basic edit operations. We have replacements, insertions and deletions as edit operations, as they are familiar from sequence comparison models, but here they operate on trees.

4.2.1 Tree edit operations

The tree edit operations *Replace*, *Delete*, and *Insert* are shown in Figure 5. Each

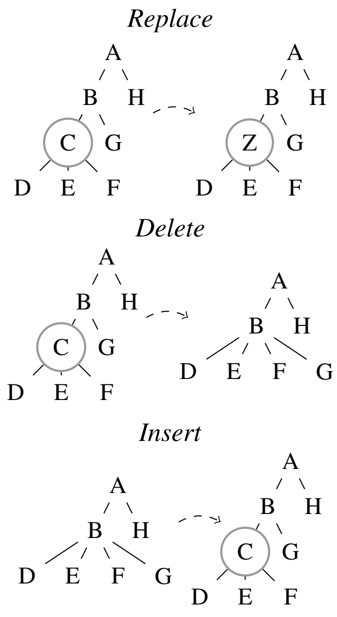


Fig. 5: Tree edit operations: Node replacement, deletion and insertion.

edit operation is associated with a cost, which may depend on the node label that is replaced, deleted or inserted. When one tree is edited into the other, the accumulated sum of edit operation costs makes up the overall cost of the editing. When we assign costs, we seek to minimize them. Alternatively, we might use similarity scores, which we seek to maximize.

Different methods of tree comparison can be based on these edit operations (and there is some confusion about this fact in the literature). It results from some vagueness in the phrase “we edit tree T into tree T' with a sequence of edit operations”.

There is an important difference between insertions and deletions – in spite of the fact that they are mutually inverse operations. If we “delete a node” in T, the resulting tree is uniquely defined. If we “insert a node Y below node X” in T', it is not defined what the resulting tree is. The outcome depends on where we insert Y into the siblings of X, and which siblings of X are chosen to become siblings of the new node Y. Figure 6 shows alternative choices for inserting node C below B. Thus, “a sequence of edit operations transforming tree T into tree T'”, a phrase that we find frequently in the literature, cannot really *produce* T' from T, but can relate T and T' when both are given. And it can do so in different ways.

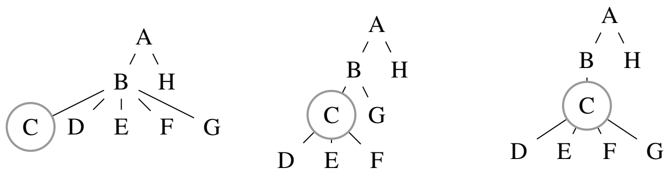


Fig. 6: Inserting node *C* as a child to *B* – three of the possible outcomes.

4.2.2 Edit or align: subtrees versus supertrees

Given our edit operations, different methods of tree comparison can be designed. Two of them have become popular in RNA structure comparison, *tree edit* and *tree alignment*, and we will describe them below. Given trees T_1 and T_2, tree alignment constructs a common supertree, while tree edit constructs a common subtree. Both minimize a cost function associated with the edit operations. Consider Figure 7. The supertree can be transformed, applying *Delete* operations only, into either T_1 or T_2. Both T_1 and T_2 can be transformed, applying *Delete* operations only, into the common subtree.

$$\begin{tikzcd}
& \text{Tree Alignment } (T_1, T_2) & \text{Supertree} \\
T_1 \arrow[ur, leftrightarrow] \arrow[dr, leftrightarrow] & & T_2 \arrow[ul, leftrightarrow] \arrow[dl, leftrightarrow] \\
& \text{Tree Edit } (T_1, T_2) & \text{Subtree}
\end{tikzcd}$$

Fig. 7: Supertree and common subtree, via tree alignment and tree edit.

It remains to be shown that subtree and supertree can be different. This is seen in Figure 8. The supertree must have two E-nodes, as they have different sets of children in T and T'.

We focus on tree edit distance and tree alignment distance here, while further modes of comparing trees are conceivable. Before you venture to design yet another method, be sure to study the article by Rosselló and Valiente [35], where they relate these and other measures under the general notion of tree embeddings.

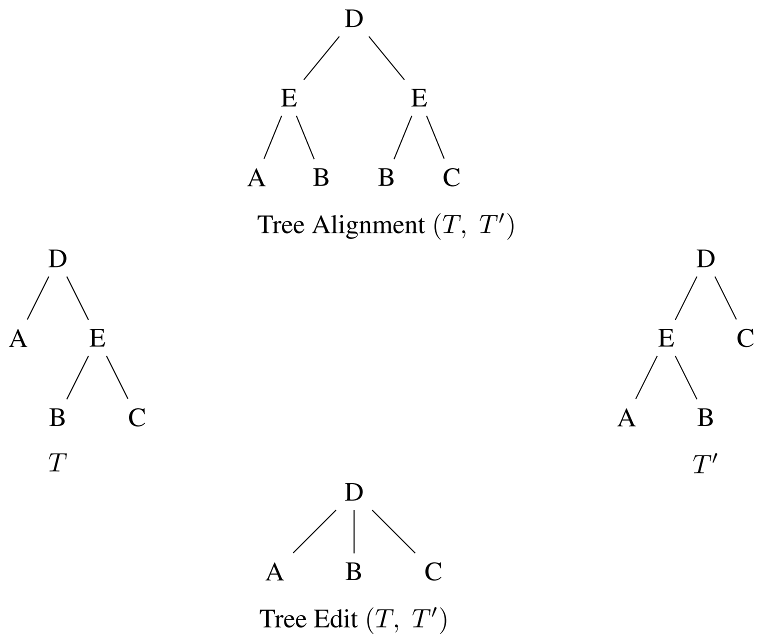


Fig. 8: Example of difference between common supertree and common subtree. Note that the two *B*-nodes in the supertree cannot be joined in the supertree without creating a dag, and the *E*-nodes cannot be joined without making *A* and *C* siblings.

4.3 Tree edit distance

Because node labels generally are not unique, we reference tree nodes by their numbers in preorder. $label(i)$ is the label of node number i. Let us write $i_1 \prec i_2$ when i_1 is an ancestor of i_2. Let w be a cost function defined on pairs of node labels.

Definition 3. A *mapping* M is a partial bijection between nodes of T and T' with the following property:
For any two pairs (i_1, j_1) and (i_2, j_2) in M,

1. $i_1 < i_2$ iff $j_1 < j_2$ (order preservation)
2. $i_1 \prec i_2$ iff $j_1 \prec j_2$ (ancestor preservation).

The *cost* of a mapping M is defined as $\sum_{(i,j)\in M} w(i,j)$

Order preservation is the condition we already have in sequence alignment, while ancestor preservation is the tree-specific property. A mapping can be seen as specifying replacements only, where the replaced nodes constitute the common subtree, while all others are deleted. See Figure 9 for an illustration of a mapping between two trees. Note that it is impossible to extend the mapping the two nodes labeled

F without violating the constraints. Should we decide to include (F,F), the nodes C and E can no longer be mapped. Should we decide to include $(A,E) \in M$, no other node can be mapped, as all nodes are descendents of A in T, while E has no descendent at all in T'.

Definition 4. The *tree edit distance* $d_{TE}(T,T')$ of two trees T and T' is the minimum cost of a mapping between T and T'.

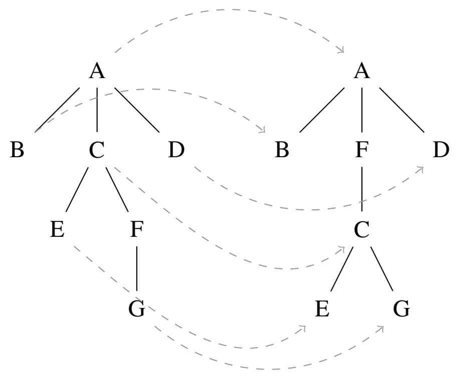


Fig. 9: Mapping, based on [38]. Including nodes F would violate the ancestor preservation requirement.

We can find the mapping of minimal cost by considering all possible edit sequences that touch each node in T and T' exactly once. This can be described by the following recurrences, given in graphical form in Figure 10.

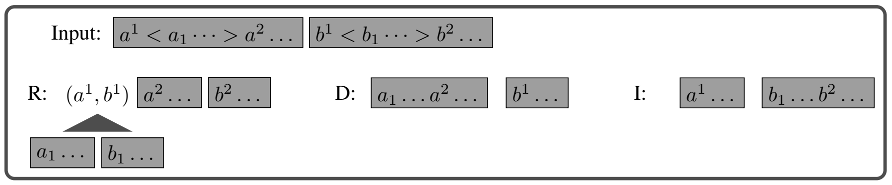

Input: $a^1 < a_1 \cdots > a^2 \ldots$ $b^1 < b_1 \cdots > b^2 \ldots$
R: (a^1, b^1) $a^2 \ldots$ $b^2 \ldots$
$a_1 \ldots$ $b_1 \ldots$
D: $a_1 \ldots a^2 \ldots$ $b^1 \ldots$
I: $a^1 \ldots$ $b_1 \ldots b^2 \ldots$

Fig. 10: Case distinction for the tree edit distance algorithm. On top are the two forests that we align, underneath are three cases corresponding to the edit operations.

The figure is to be read as follows: Two adjacent boxes represent the forests to be mapped. The input forests are replaced by (alternatively) a *Replace*, *Delete* or *Insert* operation plus the related subproblems that arise. Since we only score the replacements (because they constitute the mapping), case *R* is seen as producing

(a^1, b^1) as a contribution to the mapping and $w(a^1, b^1)$ as a contribution to its score, while the D and I cases contribute no local score, but only the (optimal) score of the respective subproblem. All mappings are scored as they are constructed, and the minimal cost is chosen. In an implementation, this requires dynamic programming recurrences that store intermediate results in tables addressed by the subproblem solved. This is technically intricate, and we refer the reader to [40, 45, 38, 36] for these details.

4.4 Tree alignment distance

While tree edit says nothing about the nodes deleted, the tree alignment method has the advantage that it puts all nodes into a relationship in the common supertree. This gives rise to a representation of tree alignments akin to sequence alignments, either in graphical form as in Figure 11 or as (aligned) sequences annotated with (aligned) dot-bracket structure.

4.4.1 Tree alignments

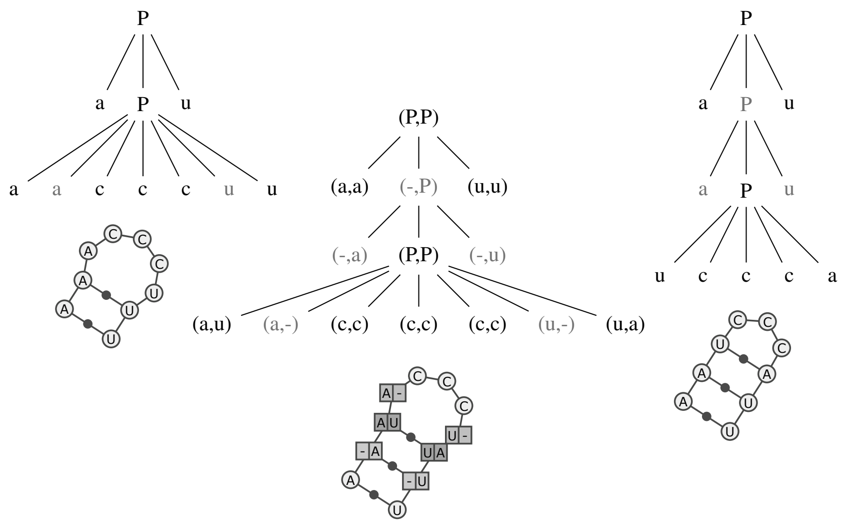


Fig. 11: Graphical representation of a tree alignment. On the left and right are two RNA hairpin structures represented as trees, and in the center is one of their possible tree alignments. Below of each tree is the corresponding squiggle representation (produced with *VARNA* [12]).

Tree alignments are defined formally akin to sequence alignments. If you see a sequence alignments as a *sequence of columns*, where the columns may hold alphabet characters and a gap symbol, the following definition is obvious to you:

Definition 5. Given ordered labeled trees over some alphabet $\mathscr{A}$, an *alignment tree* is a tree over the pair alphabet $\mathscr{A}_{pair} = \mathscr{A} \cup \{-\} \times \mathscr{A} \cup \{-\} \setminus \{(-,-)\}$.

In an alignment tree, the node labels represent edit operations, where nodes of type (a,b) are replacements, $(a,-)$ are deletions and $(-,a)$ are insertions.

Given an alignment tree A, it is clear which are the two trees aligned in it. Its projections to the first and second component, $A|_1$ and $A|_2$, are trees on the alphabet $\mathscr{A} \cup \{-\}$ of RNA with gaps. These trees may be contracted by π, where $\pi(T)$ is the tree that results from deleting all nodes with the gap symbol from a tree T. The result of this contraction are the two aligned trees over $\mathscr{A} \cup \{-\}$. This observation gives us an elegant definition for a tree alignment:

Definition 6. A tree A over $\mathscr{A}_{pair}$ is an *alignment of trees* T, T' over $\mathscr{A}$ iff $T = \pi(A|_1)$ and $T' = \pi(A|_2)$.

The score of a tree alignment is $w(A) = \sum_{(a,b)\in A} w(a,b)$, where w is the cost function on edit operations as before.

Finally, we are ready to define the tree alignment distance:

Definition 7. The *tree alignment distance* $d_{TA}(T,T')$ between two trees is the minimum cost over all possible alignments of the two trees.

$$d_{TA}(T,T') = min\{w(A)\,|\,A \text{ is an alignment of } T \text{ and } T'\}$$

It is interesting to observe that the tree alignment distance is not a metric, as it does not satisfy the triangle inequality. Figure 12 uses unit cost and demonstrates a counter-example. The triangle inequality is violated for d_{TA} (while it generally holds for d_{TE} as long as $w(Replace(a,b))$ is a metric).

4.4.2 Computing optimal tree alignments

To solve the tree alignment problem by finding the alignment with the optimal score, all possible candidate alignments have to be considered in a dynamic programming algorithm. The recurrences are similar to those of tree edit distance, but with important differences that take care that actually a supertree is constructed. Again, we give the recurrences in graphical form, see Figure 13. Again, read these recurrences as the alignment tree spreading out recursively in two dimensions on a sheet of paper. Here, each edit operation chosen leads to a node recorded for the emerging supertree. The *Replace* case leads to two recursive alignment computations for each edit operation. *Delete* and *Insert*, however, lead to splits of certain subforests in all possible ways. To compute the minimal cost of a tree alignment, the recurrences are executed in reverse, scores added as they arise, and optimal scores of

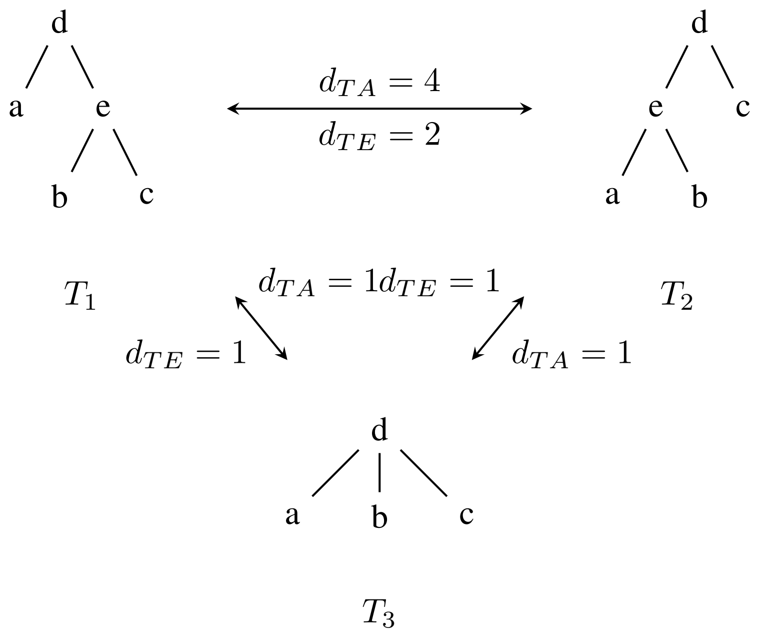


Fig. 12: Counter-example to show that the tree alignment distance does not satisfy the triangle inequality. Under unit cost, $d_{TA}(T_1,T_2) > d_{TA}(T_1,T_3) + d_{TA}(T_2,T_3)$.

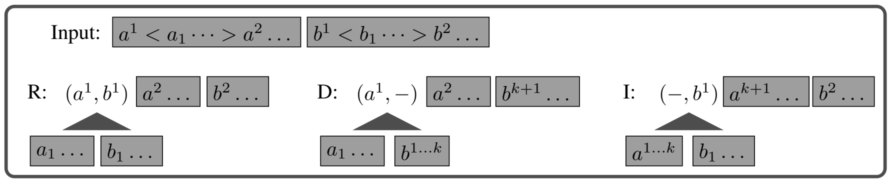

Input: $a^1 < a_1 \cdots > a^2 \ldots$ $b^1 < b_1 \cdots > b^2 \ldots$
R: (a^1, b^1) $a^2 \ldots$ $b^2 \ldots$ $a_1 \ldots$ $b_1 \ldots$
D: $(a^1, -)$ $a^2 \ldots$ $b^{k+1} \ldots$ $a_1 \ldots$ $b^{1 \ldots k}$
I: $(-, b^1)$ $a^{k+1} \ldots$ $b^2 \ldots$ $a^{1 \ldots k}$ $b_1 \ldots$

Fig. 13: Case distinction for the forest alignment distance algorithm. Again, $a^1 < a_1 \ldots > a^2$ denotes a forest whose first tree has root label a^1 and subtrees $a_1 \ldots$. Boxes indicate the forests that have to be aligned as (sub)problems. Edit operations are generated as nodes of the emerging alignment forest. On top are the two forests that we align, underneath are three cases corresponding to the edit operations.

sub-alignments are recorded in the typical dynamic programming fashion as we proceed from smaller to larger alignments. Efficiently storing and accessing these intermediate results requires some care – see [36] for the details.

4.5 Specifics of RNA structure comparison under the tree edit model

4.5.1 Refined edit operations for RNA structure trees

The two previous sections describe tree comparison models in general. Let us now see what tree editing means when comparing, in particular, RNA secondary structures. Following [23], we distinguish RNA-specific situations of interest, such as loosing a single base, a base pair, etc. In Figure 14 we see how these operations are expressed as tree edits in our particular representation of tree alignments. In our

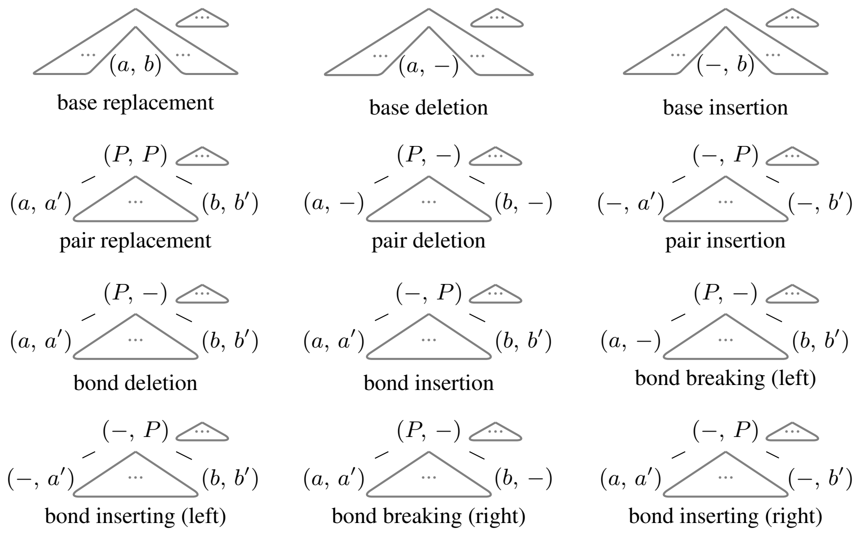


Fig. 14: RNA-specific edit operations for tree alignment

terminology, *Replace* includes the case where a base or base pair is preserved. The replacement scoring function will, of course, make the important difference between matches and mismatches.

As we learn from Figure 14, the general tree edit model accommodates all these situations, it is not specific enough to allow for the scoring one wants to apply with RNA structures. The base pair deletion, for example, is seen and scored under the general model as three *Delete* operations, one for the *P*-node and two for the bases. In scoring for RNA structure, we want to be able to assign a specific score to this situation, different from the sum of three independent deletions. Similarly, bond breaking is a consequence of loosing one base of a base pair, so we might not want to score this as two independent events. Therefore, should you plan to implement

RNA structure comparison based on a tree edit model, you must refine the general case to precisely those operations which you want to score.

One might take this even further – giving special scores to functionally relevant features that are expected in a certain ncRNA family at hand. It seems plausible that one should just have to specify these features as extra edit operations, supply a score, and some tool would generate the algorithm from such a specification. But at present, this remains a topic for research. Fortunately, good implementations of the models presented above are available.

4.5.2 Tools for structure comparison

The tree edit distance is implemented in the program *RNAdistance*, which is distributed with the *ViennaRNA* package [26, 21]. The first tool available for tree alignment was *RNAforester* [19, 20]. It is available online and for download at http://bibiserv.cebitec.uni-bielefeld.de/rnaforester. This program also computes multiple structure alignments by a progressive strategy, and has a local similarity mode to find most similar substructures in two larger structures. *RNAforester* has been used, for example, in a large scale study [33] to separate microRNA precursors from other hairpins by structural clustering. Tree alignment is also implemented in *Gardenia* [8] and available online and for download at http://bioinfo.lifl.fr/RNA/gardenia. It is a bit faster than *RNAforester* because it does not explicitly encode the *P*-nodes, thus keeping the trees smaller. The *RNAforester* tool has recently been enhanced to include an affine gap model, and to speed up alignments of structures that are already known to be similar by the use of anchorings [36]. At the time of this writing, the beta version is available at http://bibiserv.cebitec.uni-bielefeld.de/rnaforester2.

There are more programs on the market, more to come, and there is even a test site to evaluate such programs, made available by the BRASERO project [6] at http://brasero.labri.fr/.

4.6 Variations on tree edit and tree alignment distance

4.6.1 Classical work on tree edit distance

The tree comparison problem was first studied by Tai in [40], and further developed by Zhang and Shasha [45]. Both articles define the distance between two trees as the weighted number of edit operations that transform one tree into the other. Zhang and Shasha compute the distance with a dynamic programming algorithm in $O(|T_1| \cdot |T_2| \cdot depth(T_1) \cdot depth(T_2))$, which is an improvement to [40].

4.6.2 An improvement by path decomposition

For Klein [25], the edit operations are label modification and edge contraction. The trees are represented as Euler strings (parenthesized strings) and the algorithm deals with a string alignment problem. For comparing substrings of Euler strings, a simplified and less efficient variant of the Zhang-Shasha algorithm is presented. From this starting point, a faster dynamic programming algorithm is developed by using a decomposition of the tree into paths. The algorithm has a worst case bound of $O(n^3 \log n)$ for trees of size n, which is better than the Zhang-Shasha algorithm for rooted ordered trees. For some cases, Zhang-Shasha may be still faster, due to its different path decomposition strategy.

4.6.3 A linear tree edit distance algorithm for similar ordered trees

In some situations, we only want to know the exact distance if the two trees are similar, i.e. their distance lies below a certain threshold. In this situation, a linear time algorithm is possible. Touzet in [42] improves the Zhang-Shasha-Algorithm by pruning the search space, if a bounding number of errors k is given. For the search space, she represents the possible edit operations to transform one tree into the other as an edit graph, where the vertices are the Cartesian product of the two trees node indices in postorder traversal. The incident arcs represent deletion, insertion and replacement edit operations.

To prune the search space, such that only relevant vertices and arcs remain, she uses three optimization strategies. The first strategy is similar to the k band alignment algorithm for strings, as it computes a band of the edit graph only, constrained by the number of allowed errors k. The other strategies are a boundary for the maximal number of errors, which allows pruning of non-relevant vertices outside that bound, and a strategy for pruning of vertices below a certain depth, based on the number of errors between the compared subtrees. The resulting algorithm constructs an optimal mapping between trees A and B in $O(n \cdot k^3)$ time if the number of errors is bounded by k. For the trivial bound of $k = |A| + |B|$, the algorithm has the same complexity as the Zhang-Shasha algorithm ($O(n^4)$), and the average complexity would be worse compared to Zhang-Shasha's, as some optimizations have been left out.

4.6.4 Alignment hierarchy and Decomposition strategies

As a theoretical unifying framework, Blin et al. propose the alignment hierarchy [8]. In [13, 14], Dulucq and Touzet generalize several tree edit distance algorithms by describing them as decomposition strategies. The main algorithms they study are the ones of Klein [25] and Zhang-Shasha [44]. The central idea is the concept of cover strategies. Dulucq and Touzet introduce a general framework of cover strategies, analyze the complexity of cover strategies and develop a new tree edit distance algorithm, optimal in the cover strategy framework.

4.6.5 Tree Edit Distance With Gaps

Touzet studies edit distance with convex gap weights in [41], and proves that there is no polynomial algorithm for the tree edit distance problem with convex gap weights, unless $P = NP$. She restricts the definition of gaps to complete subtrees (such that all descendants belong to the subtree), and presents a quadratic algorithm for the according tree edit distance.

4.6.6 Seeded tree edit distance

The method of seeded tree alignment [27] is an interesting combination of mapping and alignment. In spite of its name, it constructs mappings rather than alignments.

Seed mappings, a set of node pairs which have to map onto each other, are used to constrain the mappings. The seed mappings preserve the lowest common ancestor relationship, and in this way select a specific common super-tree structure. This makes the mappings compatible with, while still more abstract than, tree alignments. This has been generalized to seed sets determined by exact matching, where the seeds in the set need not be compatible [18]. This leads to the problem of optimally "chaining" compatible seeds, for which an in $O(m^2 log(m))$ time and $O(m^2)$ space algorithm is presented in [2], where m is the number of seeds.

4.6.7 Classical tree alignment

The tree alignment algorithm was introduced as an alternative to the tree edit algorithm by Jiang et al. in [24] for pairwise and multiple alignment. Comparing tree alignment to tree edit distance, Jiang states that the tree alignment corresponds to a restricted tree edit distance, "in which all the insertions precede all deletions" [24]. This is an operational way to express the difference between a common supertree and a common subtree. If all insertions are done first, the common supertree is produced at the point before the first deletion. If deletions come first, their end marks the common subtree.

Jiang's algorithm for ordered trees has time complexity $O(|T_1| \cdot |T_2| \cdot (deg(T_1) + deg(T_2))^2)$, where $deg(T_i)$ is the degree of T_i, so the algorithm is faster than all known ones for the tree edit distance, if the degrees are smaller than the depths of the trees.

4.6.8 Average case complexity of tree alignment

Practical experience with tree alignments suggested that the algorithm performs better in practice than the asymptotics given above suggest. Note that in the worst case, when $deg(T) \approx |T|$, a runtime of $O(n^4)$ is implied. This is puzzling, as when the degree is almost equal to the tree size, the tree is essentially a sequence (i.e. a root node with $n-1$ leaves). In fact in [17], Herrbach et al. could show that tree alignment runs in $O(mn)$ *on average*, where m and n are the sizes of the trees. In the proof, they also show that several characteristics of trees, such as the number of closed subforests or prefix/suffix subforests of a tree are in $O(n)$ on average. These results are relevant for the average-case analysis of other tree-based algorithms, too. In the proof, they also show that several characteristics of trees, such as the number of closed subforests or prefix/suffix subforests of a tree are in $O(n)$ on average. These results are also relevant for the average-case analysis of other tree-based algorithms.

4.6.9 Local similarity in RNA secondary structure and progressive multiple alignment

Hoechsmann et al. extended Jiang's tree alignment distance to compute also pairwise local alignments and multiple global alignments [19, 20]. Local alignment maximizes a similarity score (rather than minimizing distance) and is the tree-based analog of the Smith-Waterman algorithm [39] for local sequence similarity. In terms of forests, this means that all pairs of closed subforests (= all substructures) have to be compared to each other as subproblems, which raises worst-case runtime complexity to $O(|T_1| \cdot |T_2| \cdot deg(T_1) \cdot deg(T_2) \cdot (deg(T_1) + deg(T_2)))$. The multiple structure alignment uses the profile alignment method known from sequence alignment. After computing pairwise all-against-all alignments, profiles are created and successively aligned, starting with nearest neighbors.

4.6.10 Tree alignment with affine gaps and anchors

Two improvements to the tree alignment algorithm are presented in [36], motivated by its application to RNA structure alignment, where tree alignments tend to appear scattered with small gaps. The first improvement is the introduction of an affine gap cost model. In this model, the costs of opening a gap can be set higher than the costs of extending it, thus favouring few large gaps over many small ones. This appears biologically more plausible, as each a gap indicates an evolutionary event. The algorithm essentially uses seven copies of the recurrence given above, where the sub-alignments can be in different combinations of no-gap, parent-gap and sibling-gap mode. The second improvement is a speed-up of the alignment when certain nodes in the forest are pre-aligned by a so-called anchoring. While the affine gap model slows down the tree alignment by a constant factor ≈ 7, the anchoring provides a linear speedup depending on the number of anchors.

4.6.11 Alignment under an extended set of edit operations

Allali and Sagot [3] extend the set of operations supported by the Zhang-Shasha algorithm [45], adding node fusion and splitting operations. These operations are particularly relevant in the context of a coarse-grain representation (one node per helix), where they allow to match an helix with two consecutive helices, e.g. separated by a bulge. This improved expressivity comes at a cost, and the resulting algorithm has complexity in $O(4^l \cdot d_1^{l+1} \cdot d_2^{l+1} \cdot n_1 \cdot n_2)$, where d_x is the max degree of a node, n_x is the sequence length and l is the maximal number of consecutive nodes that are fusioned. The increase in complexity is compensated by a hierarchical approach [5], which initially aligns RNAs at a coarse grain level using the extended set of operations. Then the algorithm *zooms in*, using the higher level alignment as a set of constraints for a refined alignment, and using a classic set of operations.

5 Comparing structures with pseudoknots: The next frontier

In this chapter, we have focused on the methods most commonly used for RNA secondary structure comparison. We have restricted ourselves to plain – non-crossing – secondary structures, leaving aside any elements of 3D structure such as base triplets, or special tertiary motifs such as kink turns and E loops (cf. Chapter 18). In particular, we have excluded pseudo-knots, defined here broadly as any crossing interactions, from our consideration – not because they are unimportant, but because they are the topic of ongoing research. At this point, we see no generally accepted method of comparison for pseudo-knotted structures, and no widely used tool for this purpose.

There are three problems that impede progress in this field: algorithms tend to be sophisticated and computationally expensive, topologically-feasible conformations are hard t o characterize, and reliable data is relatively scarce - a situation that equally affects the development of pseudo-knot prediction methods.

5.1 An algorithmic challenge

From an algorithmic perspective, the problem of comparing pseudo-knots is usually abstracted as the comparison of arc-annotated sequences (cf. Section 2.5.1) featuring crossing interactions. This problem has been thoroughly studied through the lens of computational complexity theory, a branch of computer science which aims at characterizing the inherent difficulty of problems. As recently surveyed by Touzet and Blin [9], the results turned out to be quite discouraging. For instance, exactly solving the problem was shown to be NP-hard (i.e. very probably intractable) under any reasonable model for superstructures. It was also proven difficult to approximate accurately and efficiently (MAX-SNP hard) under any realistic sets of operations on RNA (e.g. arc-breaking, arc-altering...). Therefore, there is little hope for general algorithms that would align arbitrary arc-annotated sequences both exactly and efficiently.

5.2 Topology to the rescue

However, while any RNA can be modeled as an arc-annotated sequence, it is noteworthy that not every arc-annotated sequence is a realistic candidate for an RNA conformation. Indeed, some arc-annotated sequences may induce such an intricate structure that reconstructing them in 3D would unavoidably lead to clashes and other highly unstable features!

In a perfect world, the algorithmic difficulty of the problem would arise from such unfeasible instances, and one could devise algorithms whose runtimes would be reasonable for real RNAs. More realistically, the idea that RNA structures may be more constrained than arc-annotated sequences has led to three categories of approaches, each exploiting in some way the restricted topology of real structures. Classes of pseudo-knots have been characterized topologically in [10, 32], where it is shown how to proceed from a chosen pseudo-knot architecture to a folding algorithm.

5.3 From folding pseudo-knots to their alignment

A first idea, due to Möhl et al. [28], considers restricted – computationally easy, yet sufficiently expressive – classes of pseudo-knots introduced in the context of RNA folding (Chapter 12). The main rationale is that the scheme underlying a folding algorithm can be transformed, without too much pain, into a dynamic-programming algorithm for the alignment. Moreover, membership to various classes of pseudo-knots can be efficiently tested, as shown by Rastegari and Condon [31]. This suggests a *meta-algorithm*, which starts by determining the class of each structure, and then selects a suitable dynamic programming algorithm.

The practicality of the resulting method largely depends on the class of each compared structure. For instance, two pseudo-knotted structures of the Rivas and Eddy type [34] lead to a polynomial, yet prohibitive, complexity in $O(n \cdot m^6)$, where n and m are the length of the longer and shorter sequence respectively. Simpler classes of structures, on the other hand, may be aligned using as little as $O(n \cdot m^4)$ time and, more importantly, only $O(n \cdot m^2)$ memory.

5.4 Parameterized DP-based algorithms

Another difficulty related to the previous approach resides in the necessity, for each RNA, to belong to one of the classes studied for the folding problem. While existing classes already achieve a fair level of generality, a single interaction in the wrong place may be sufficient to make the whole approach inapplicable. This is especially problematic for tertiary motifs, which are increasingly considered as central to the notion of structural homology, whereas pseudo-knotted classes of RNAs are usually defined in term of helices only.

This need for more robust approaches has motivated further algorithmic works, based on the concept of parameterized algorithms. Such an algorithm provides a general solution to an NP-Hard problem, which can be thought as automatically adapting its complexity to the difficulty of the problem. A parameter is introduced, and one aims at finding an algorithm whose complexity is usually exponential on the value of the parameter, but remains polynomial on the length of the instance (here, the cumulated length of both RNAs). Although the parameter may adopt arbitrary large values, its value on real-life instances is hopefully small, and the resulting method may be of practical interest. Another asset of this approach is that the time taken by the algorithm can usually be anticipated, leaving to the user to decide whether to embark in a heavy computation, or to consider alternative options (e.g. simplify the structure, prealign the sequence by adding further constrains...).

Möhl et al. [29] describe such an algorithm, based on a parameter k called the crossing number. The final complexity of the algorithm takes $O(n^4 \cdot s^{16k})$, where n is the length of the longest sequence, and s is the number of base-pairs involved in a crossing stem. In practice, the value of k seems typically equal to 0 or 1, and s typically grows much slower than the sequence length n, allowing for the alignment of RNAs with pseudo-knots about 400 nucleotide long in a matter of hours.

5.5 *Alternative problem encodings*

A last category of exact methods reformulates the alignment of RNAs with pseudo-knots as a mathematical optimization problem. More precisely, the possible ways to align sequences are encoded as variables (e.g. modeling the position), coupled with a system of (in)equations that describes the constraints weighing on a solution. Generic solvers can then be used to maximize an objective function (i.e. the score of the alignment) under the constraints described by the system of equations. While this allows, in principle, to solve any optimization problem, the practicality of the resulting tool greatly depends on the quality of the encoding, and considerable craftsmanship is usually required to achieve good performance.

The LARA software [7] uses integer programming , a specialized version of this paradigm, to solve the RNA alignment problem in the presence of pseudo-knots. Its first encodes the problem as a system of equations, and then simplifies it using the Lagrangian Relaxation. This general technique penalizes the violation of certain constraints (here, the symmetry of the matching) instead of enforcing it strictly, making the system easier to solve. A near-optimal alignment is then reconstructed from the values for the variables giving the best score by iterating the solving of the equation system using different penalties. Quite interestingly, this formalism allows to express multiple base-pairs per position, or even the probability matrices described in Chapter 4. A standalone implementation is available, and can be interfaced with T-Coffee [30] to venture into the realm of multiple sequence alignment with pseudo-knots.

Finally, let us mention a heuristic approach, based on geometric hashing, implemented in the HARP [1] webserver at http://bioinfo3d.cs.tau.ac.il/HARP/. Here, the graph of interacting helices is decomposed into elementary triangles, used as building blocks to reconstruct a mapping using a variant of the maximal-weighted matching algorithm. This matching is then extended in a greedy fashion into an *alignment* (for lack of a better word). Despite its non-exact nature and its theoretical $O(n^7)$ worst-case complexity, the approach seems to allow for a decent alignment of large sub-unit ribosomal RNAs.

5.6 Concluding remark

As can be seen in this short review, few algorithms have been developed for the structural alignment of RNAs featuring pseudo-knots, and even fewer implementations went further than the proof-of-concept stage. Arguably the main open problem is the support of non-canonical base-pairs and tertiary motifs. While such structural features are pervasive and conserved throughout evolution, they tend to be poorly handled by exact approaches, at least for two reasons. Firstly, dynamic programming schemes are usually defined at the helix level, and tertiary interactions may be isolated, breaking their scheme. Secondly, considering such interactions leads to multiple partners for a given base, leading to an increase of the theoretical difficulty. Future algorithms may offer more flexibility towards such an inclusion. In the meantime, aligning pseudo-knots requires huge computational resources, and at least equal amount of patience.

Fig. 15: Before endodontic therapy: filling exhibits pressure on pulp chamber

Acknowledgement Thanks go to Dr. Ralph Kandalla for a careful endodontic therapy of the first author, see also Figure 15. Without his expertise, this chapter would not have been completed in time. Thanks also go to Sonja Klingberg for a careful reading of the manuscript.

References

[1] Mira Abraham and Haim J. Wolfson. Inexact graph matching by "geodesic hashing" for the alignment of pseudoknoted RNA secondary structures. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2011*, pages 45–57, Czech Technical University in Prague, Czech Republic, 2011.

[2] Julien Allali, Cedric Chauve, Pascal Ferraro, and Anne-Laure Gaillard. Efficient chaining of seeds in ordered trees. *Journal of Discrete Algorithms*, 14:107–118, 2012.

[3] Julien Allali and Marie-France Sagot. Novel tree edit operations for RNA secondary structure comparison. *Algorithms in Bioinformatics*, pages 412–425, 2004.

[4] Julien Allali and Marie-France Sagot. A multiple graph layers model with application to RNA secondary structures comparison. In *String Processing and Information Retrieval*, pages 348–359. Springer, 2005.

[5] Julien Allali and Marie-France Sagot. A multiple layer model to compare RNA secondary structures. *Software - Practice and Experience (SPE)*, 38(8):775–792, 2008.

[6] Julien Allali, Cédric Saule, Cedric Chauve, Yves d'Aubenton Carafa, Alain Denise, Christine Drevet, Pascal Ferraro, Daniel Gautheret, Claire Herrbach, Fabrice Leclerc, Antoine de Monte, Aida Ouangraoua, Marie-France Sagot, Michel Termier, Claude Thermes, and Hélène Touzet. Brasero: A resource for benchmarking rna secondary structure comparison algorithms. *Advances in Bioinformatics*, 2012, 2012.

[7] Markus Bauer and Gunnar W. Klau. Structural Alignment of Two RNA Sequences with Lagrangian Relaxation. In Rudolf Fleischer and Gerhard Trippen, editors, *Proceedings of the 15th International Symposium ISAAC 2004*, volume 3341 of *Lecture Notes in Computer Science*, pages 113–123. Springer, 2004.

[8] Guillaume Blin, Alain Denise, Serge Dulucq, Claire Herrbach, and Hélène Touzet. Alignments of RNA structures. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 7(2):309–322, 2010.

[9] Guillaume Blin and Hélène Touzet. How to compare arc-annotated sequences: The alignment hierarchy. In *SPIRE*, pages 291–303, 2006.

[10] Michaël Bon and Henri Orland. Tt2ne: a novel algorithm to predict rna secondary structures with pseudoknots. *Nucleic Acids Research*, 39(14):e93, 2011.

[11] Yanga Byun and Kyungsook Han. PseudoViewer3: generating planar drawings of large-scale RNA structures with pseudoknots. *Bioinformatics*, 25(11):1435–1437, 2009.

[12] Kevin Darty, Alain Denise, and Yann Ponty. VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, 25(15):1974–1975, 2009.

[13] Serge Dulucq and Hélène Touzet. Analysis of tree edit distance algorithms. In *CPM '03: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, pages 83–95, 2003.

[14] Serge Dulucq and Hélène Touzet. Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms*, 3(2-4):448–471, 2005.

[15] Walter Fontana, Danielle A.M. Konings, Peter F. Stadler, and Peter Schuster. Statistics of RNA secondary structures. *Biopolymers*, 33(9):1389–1404, 1993.

[16] Robert Giegerich, Björn Voß, and Marc Rehmsmeier. Abstract shapes of RNA. *Nucleic Acids Research*, 32(16):4843–4851, 2004.

[17] Claire Herrbach, Alain Denise, and Serge Dulucq. Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm. *Theoretical Computer Science*, 411:2423–2432, 2010.

[18] Steffen Heyne, Sebastian Will, Michael Beckstette, and Rolf Backofen. Lightweight comparison of RNAs based on exact sequence-structure matches. *Bioinformatics*, 25(16):2095–2102, 2009.

[19] Matthias Hoechsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. *Proceedings of the IEEE Computational Systems Bioinformatics Conference (CSB 2003)*, 2:159–168, 2003.

[20] Matthias Hoechsmann, Björn Voß, and Robert Giegerich. Pure multiple RNA secondary structure alignments: A progressive profile approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1:53–62, 2004.

[21] Ivo L. Hofacker, Walter Fontana, Peter F. Stadler, L.Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie*, 125:167–188, 1994.

[22] Stefan Janssen, Jens Reeder, and Robert Giegerich. Shape based indexing for faster search of RNA family databases. *BMC Bioinformatics*, 9(1):131, 2008.

[23] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–388, 2002.

[24] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees – an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[25] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *In Proceedings of the 6th annual European Symposium on Algorithms (ESA)*, pages 91–102. Springer-Verlag, 1998.

[26] Ronny Lorenz, Stephan H. Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA Package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011.

[27] Antoni Lozano, Ron Y. Pinter, Oleg Rokhlenko, Gabriel Valiente, and Michal Ziv-Ukelson. Seeded tree alignment. *IEEE Transactions on Computational Biology and Bioinformatics*, pages 503–513, 2008.

[28] Mathias Moehl, Sebastian Will, and Rolf Backofen. Lifting prediction to alignment of RNA pseudoknots. *Journal of Computational Biology*, 17(3):429–442, 2010.

[29] Mathias Möhl, Sebastian Will, and Rolf Backofen. Fixed parameter tractable alignment of rna structures including arbitrary pseudoknots. In *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, 2008.

[30] Cedric Notredame, Desmond G. Higgins, and Jaap Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 302:205–217, 2000.

[31] Baharak Rastegari and Anne Condon. Parsing nucleic acid pseudoknotted secondary structure: Algorithm and applications. *Journal of Computational Biology*, 14:16–32, 2007.

[32] Christian M. Reidys, Fenix W. D. Huang, Jørgen E. Andersen, Robert Penner, Peter F. Stadler, and Markus Nebel. Topology and prediction of RNA pseudoknots. *Bioinformatics*, 27(8):1076–1085, 2011.

[33] William Ritchie, Matthieu Legendre, and Daniel Gautheret. RNA stem loops: to be or not to be cleaved by RNAse III. *RNA*, 13(4), February 2007.

[34] Elena Rivas and Sean R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J Mol Biol*, 285:2053–2068, 1999.

[35] Francesc Rosselló and Gabriel Valiente. An algebraic view of the relation between largest common subtrees and smallest common supertrees. *Theoretical Computer Science*, 362(1):33–53, 2006.

[36] Stefanie Schirmer and Robert Giegerich. Forest alignment with affine gaps and anchors. In *Combinatorial Pattern Matching*, pages 104–117. Springer, 2011.

[37] Bruce A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer Applications in Bioscience*, 4(3):387–393, Aug 1988.

[38] Bruce A. Shapiro and Kaizhong Z. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in Bioscience*, 6(4):309–318, Oct 1990.

[39] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, Mar 1981.

[40] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[41] Hélène Touzet. Tree edit distance with gaps. *Information Processing Letters*, 85(3):123–129, 2003.

[42] Hélène Touzet. A linear tree edit distance algorithm for similar ordered trees. In *CPM '05: Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching*, pages 334–345, 2005.

[43] Andreas Wilm, Kornelia Linnenbrink, and Gerhard Steger. Construct: improved construction of RNA consensus structures. *BMC Bioinformatics*, 9(219), 2008.

[44] Kaizhong Zhang and Dennis Shasha. On the editing distance between trees and related problems. *Ultra-computer Note 122, NYU C.S TR 310,*, August 1987.

[45] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262, December 1989.

[46] Christian Höner zu Siederdissen and Ivo L. Hofacker. Discriminatory power of rna family models. *Bioinformatics*, 26(18):i453–i459, 2010.

[47] Michael Zuker. *The use of dynamic programming algorithms in RNA secondary structure prediction*, pages 159–184. CRC Press, Boca Raton, RL, 1989.

Contents

Version of May 26, 2014

Index