HAL
open science

Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm

Claire Herrbach, Alain Denise, Serge Dulucq

To cite this version:

Claire Herrbach, Alain Denise, Serge Dulucq. Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm. Theoretical Computer Science, 2010, 411, pp.2423-2432. inria-00541269

HAL Id: inria-00541269

https://inria.hal.science/inria-00541269v1

Submitted on 30 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm

Claire Herrbach [a,b,*] Alain Denise [a,b] Serge Dulucq [c]

[a] *LRI, Université Paris-Sud 11, CNRS; Bât. 490; 91405 Orsay Cedex*

[b] *IGM, Université Paris-Sud 11, CNRS; Bât. 400; 91405 Orsay Cedex*

[c] *LaBRI, Université Bordeaux 1, CNRS; 33405 Talence Cedex*

Abstract

We prove that the average complexity of the pairwise ordered tree alignment algorithm of Jiang, Wang and Zhang is in $O(nm)$, where n and m stand for the sizes of the two trees, respectively. We show that the same result holds for the average complexity of pairwise comparison of RNA secondary structures, using a set of biologically relevant operations.

Key words: average complexity, tree alignment, RNA alignment, RNA structure

1 Introduction

Pairwise comparison of ordered trees has been subject to a number of works in the recent years. One reason is that there are natural and important applications in bioinformatics, notably in the domain of RNA structure analysis. Two main approaches have been developed so far for comparing labeled ordered trees: tree edition and tree alignment. Both are based on the following edit operations on nodes:

- Substitution: the label of the node is modified.

* Corresponding author.
Email addresses: `Claire.Herrbach@lri.fr` (Claire Herrbach), `Alain.Denise@lri.fr` (Alain Denise), `Serge.Dulucq@labri.fr` (Serge Dulucq).

- Insertion / Deletion: these are two symmetrical operations. When a node is deleted, all of its children become children of its father. When a node is inserted, it takes place between two (possibly non consecutive) siblings, and all the nodes between these brothers become children of the new node.

A cost function is associated to each operation and its arguments. Given two trees, it is possible to find a sequence of operations that changes the first one into the other one. The cost of the sequence equals the sum of the costs of its operations. Now, given two trees T_1 and T_2,

- the edition problem consists in finding the minimal sequence of operations, in terms of cost, that changes T_1 into T_2.
- the alignment problem consists in finding a third tree T such that both T_1 and T_2 can be changed into T by using only substitutions and insertions, in such a way to minimize the total cost of the operations that have been performed.

The alignment and edition problems are not equivalent for trees (though they are equivalent for sequences). The first efficient edition algorithm for ordered rooted trees is due to Zhang and Shasha [13]. It runs in $O(n^2m^2)$ worst-case complexity, and in $O(n^{3/2}m^{3/2})$ average-case complexity [5], where n and m stand for the numbers of nodes of the two trees, respectively. Some authors have given variants of the algorithm which improve the worst-case complexity [6,11]. Alignment of trees was first investigated by Jiang, Wang and Zhang [10]. They gave an algorithm whose worst-case complexity is in $O(n^2m^2)$. Up to now, the average complexity of their algorithm was an open question.

RNA structure comparison is a natural application of tree comparison, since any so-called *secondary structure without pseudoknots* can be modeled by a labeled ordered tree [14,12]. However, it turns out that the edit operations on trees are not suitable to compare RNA secondary structures in a biologically relevant way. Some natural changes on RNAs do require some additional and more complex operations. Jiang *et al.* have defined in [9] a set of operations suitable for RNA structures, and have studied the complexity of the edition problem for a complete hierarchy of RNA structures. They let open the question of the complexity of the problem for two secondary structures without pseudoknots. This question was answered later by Blin *et al.* [2]: the edition problem of two secondary structures without pseudoknots is SNP-hard. The authors of the present work, together with Blin and Touzet, have shown that, in contrast, the alignment problem of two secondary structures without pseudoknots is polynomial, in $O(n^2m^2)$ worst-case complexity [3,8,1] [1]. Roughly, the algorithm consists in a generalisation of the tree alignment algorithm to

[1] Additionally, a complete hierarchy of alignment problems has been thoroughly studied by Blin and Touzet in [3].

the new operations. Up to now, its average complexity was unknown.

In the present paper, we prove that the average complexity of the tree alignment algorithm of [10] is in $O(nm)$, as well as the average complexity of the RNA structure alignement algorithm of [1,3,8]. The paper is organised as follows. In Section 2, we briefly outline the tree alignment algorithm; then Theorem 1 states the average complexity of the algorithm, and its proof is given. In Section 3, we generalise the result to RNA structures: after briefly recalling the RNA edit operations and sketching the algorithm, we give in Theorem 2 its average complexity.

2 Average complexity analysis of the tree alignment algorithm

Let us introduce some notation. We write $v(f)$ for a tree, where v stands for the root node and f is its subforest. We note $p \circ s$ the forest composed by the concatenation of the subforests p and s. Since we deal with ordered trees and ordered forests, $p \circ s$ is not the same forest as $s \circ p$.

Let F be a forest. Two or several nodes of F are said to be *siblings* if they all have the same parent, or if no one of them has a parent. A *closed subforest* of F is a subforest $v_1(f_1) \circ \ldots \circ v_k(f_k)$ of F, such that $v_1, \ldots, v_k$ are consecutive sibling nodes. A *suffix subforest* of F is a closed subforest $v_1(f_1) \circ \ldots \circ v_k(f_k)$ of F, such that v_k has no right sibling node. We note γ for the cost function: for two vertices v and v', $\gamma(v, v')$ is the substitution cost of v by v'. By convention, $\gamma(v, -)$ and $\gamma(-, v')$, respectively, stand for the costs of node deletion and node insertion.

We note $\mathsf{align}(f, g)$ for the alignment distance between the subforests f and g. Jiang *et al.* have given a dynamic programming based algorithm in [10] for computing it, based on a recurrence relation. We give here an equivalent recurrence with simplified notations:

$$\mathsf{align}(\varepsilon, \varepsilon) = 0$$

and for any pair of closed subforests $v(f) \circ g$ and $v'(f') \circ g'$, where f, g, f' and g' are possibly empty,

$$\mathsf{align}(v(f) \circ g, v'(f') \circ g') =$$

$$\min \begin{cases} \gamma(v, -) + \min\{\mathsf{align}(f, p') + \mathsf{align}(g, s') \mid p' \circ s' = v'(f') \circ g'\} \\ \gamma(-, v') + \min\{\mathsf{align}(p, f') + \mathsf{align}(s, g') \mid p \circ s = v(f) \circ g\} \\ \gamma(v, v') + \mathsf{align}(f, f') + \mathsf{align}(g, g'). \end{cases}$$

The alignment distance between two trees T_1 and T_2 is given by $\mathsf{align}(T_1, T_2)$. The alignment algorithm consists, first, in computing $\mathsf{align}(T_1, T_2)$ with the above recurrence, and saving the intermediate results in tables. Then a trace-back stage gives the alignment. This last stage is linear according to the size of the largest tree. Thus its complexity is negligible compared to the first stage : computing $\mathsf{align}(T_1, T_2)$. We focus on it.

Theorem 1 *The average complexity of the Jiang-Wang-Zhang tree alignment algorithm for two trees T_1 and T_2 is in $\mathcal{O}(|T_1|.|T_2|)$, where $|T|$ stands for the number of nodes of the tree T.*

For the sake of clarity, the proof has been splitted into four lemmas below. Lemma 1 states a preliminary formula for the average complexity, where some parameters of the trees are present, such as their number of *closed subforests* or *suffix subforests*. Then Lemmas 2, 3 and 4 allow to simplify the formula. The first two lemmas were also given in [8,1], we recall them here for the sake of self-containment.

Now let us give some additional definitions and notations. Let T be a tree, v be a vertex of T and f be a subforest of T. The *degree* of v, denoted d_v, is its number of children. The *rank* of v, denoted r_v, is the position of v among its siblings according to the left-to-right order. The *width* of f, denoted $w(f)$, is the number of trees contained in f, *i.e.* the number of nodes that have no parent node in f. We write $n_T = |T|$, and ℓ_T for the number of leaves of T. Finally, $\mathcal{C}(T)$ denotes the set of closed subforests of T and $\mathcal{S}(T)$ denotes the set that contains the suffix subforests and the subtrees of T.

Lemma 1 *Let T_1 and T_2 be two trees. The average number of operations required to compute the alignment distance between two trees having $n+1$ and $m+1$ nodes respectively is of the order of*

$$\frac{\sum_{|T_1|=n+1}\sum_{|T_2|=m+1}\mathsf{N}(T_1,T_2)}{\frac{1}{n+1}\binom{2n}{n}\frac{1}{m+1}\binom{2m}{m}},$$

where $\mathsf{N}(T_1, T_2) =$

$$|\mathcal{S}(T_1)| \times \sum_{f\in\mathcal{C}(T_2)} w(f) \;+\; \sum_{f\in\mathcal{S}(T_1)} w(f) \times |\mathcal{C}(T_2)|$$

$$+ \sum_{f\in\mathcal{C}(T_1)} w(f) \times |\mathcal{S}(T_2)| \;+\; |\mathcal{C}(T_1)| \times \sum_{f\mathcal{S}(T_2)} w(f).$$

Proof The first part is straightforward: the number of required operations is summed over all pairs of trees, and divided by the number of pairs, that is a product of two Catalan numbers.

Regarding the second part, it was stated in [10] that, for computing $\mathsf{align}(T_1, T_2)$, it is necessary and sufficient to compute $\mathsf{align}(F_1, F_2)$ for each pair $(F_1, F_2) \in (\mathcal{S}(T_1) \times \mathcal{C}(T_2)) \cup (\mathcal{C}(T_1) \times \mathcal{S}(T_2))$. Then, for such a pair (F_1, F_2) and according to the recurrence relation, the computation of $\mathsf{align}(F_1, F_2)$ requires to choose the minimal cost among $(w(F_1) + 1) + (w(F_2) + 1) + 1$ possibilities. Indeed, computing the first line of the recurrence relation requires to choose the best decomposition of F_2 into two subforests p_2 and s_2 such that $F_2 = p_2 \circ s_2$. This means choosing the best decomposition among $w(F_2)+1$ possible ones. In the same manner, computing the second line of the recurrence relation requires to choose the best decomposition of F_1 among $w(F_1)+1$ possible decompositions. At last, computing the third line of the recurrence relation requires no comparison and offers only one possibility.

Thus, the number of operations required to compute $\mathsf{align}(T_1, T_2)$ is of the order of

$$\sum_{(F_1,F_2)\in(\mathcal{S}(T_1)\times\mathcal{C}(T_2))\cup(\mathcal{C}(T_1)\times\mathcal{S}(T_2))} (w(F_1) + w(F_2)) \,.$$

By developping this sum, we get:

$$\begin{aligned}
&\sum_{(F_1,F_2)\in(\mathcal{S}(T_1)\times\mathcal{C}(T_2))\cup(\mathcal{C}(T_1)\times\mathcal{S}(T_2))} (w(F_1) + w(F_2)) \\
&= \sum_{F_1\in\mathcal{S}(T_1)} \sum_{F_2\in\mathcal{C}(T_2)} (w(F_1) + w(F_2)) \;+\; \sum_{F_1\in\mathcal{C}(T_1)} \sum_{F_2\in\mathcal{S}(T_2)} (w(F_1) + w(F_2)) \\
&\qquad - \sum_{F_1\in\mathcal{S}(T_1)} \sum_{F_2\in\mathcal{S}(T_2)} (w(F_1) + w(F_2)) \\
&= \sum_{F_1\in\mathcal{S}(T_1)} w(F_1) \times |\mathcal{C}(T_2)| \;+\; |\mathcal{S}(T_1)| \times \sum_{F_2\in\mathcal{C}(T_2)} w(F_2) \\
&\qquad + \sum_{F_1\in\mathcal{C}(T_1)} w(F_1) \times |\mathcal{S}(T_2)| \;+\; |\mathcal{C}(T_1)| \times \sum_{F_2\in\mathcal{S}(T_2)} w(F_2) \\
&\qquad - \sum_{F_1\in\mathcal{S}(T_1)} w(F_1) \times |\mathcal{S}(T_2)| \;-\; |\mathcal{S}(T_1)| \times \sum_{F_2\in\mathcal{S}(T_2)} w(F_2) \\
&= \mathcal{O}(\mathsf{N}(T_1, T_2)).
\end{aligned}$$

□

Lemma 2 *Let T be a tree. The following equalities hold:*

(1) $|\mathcal{S}(T)| = n_T + \ell_T - 1$,
(2) $|\mathcal{C}(T)| = 1 + \sum_{v\in T} \binom{d_v+1}{2}$,
(3) $\sum_{f\in\mathcal{S}(T)} w(f) = \ell_T + \sum_{v\in T} \binom{d_v+1}{2}$,
(4) $\sum_{f\in\mathcal{C}(T)} w(f) = 1 + \sum_{v\in T} \binom{d_v+2}{3}$.

Proof

(1) There are n_T subtrees in T whose roots are the n_T nodes of T, respectively. There are also n_T suffix subforests in T whose leftmost trees are rooted at the n_T nodes of T, respectively. Among those n_T suffix subforests, $n_T + 1 - \ell_T$ ones are also subtrees. Indeed, they correspond to subtrees that are rooted at the rightmost child of each internal node, plus the tree T itself. Thus, the set $\mathcal{S}(T)$ contains $n_T + n_T - (n_T + 1 - \ell_T) = n_T + \ell_T - 1$ elements.

(2) For each vertex v in T, there are as many closed subforests whose rightmost trees is rooted at v as the rank of v. Then, the cardinality of $\mathcal{C}(T)$ equals the sum of ranks of all vertices of T :

$$\begin{aligned}\sum_{v\in T} r_v &= 1 + \sum_{v\in T}\sum_{1\le k\le d_v} k \\ &= 1 + \sum_{v\in T}\left(\frac{d_v(d_v+1)}{2}\right) \\ &= 1 + \sum_{v\in T}\binom{d_v+1}{2}.\end{aligned}$$

(3) The width of a tree is equal to 1. Then the sum of the widths of the subtrees of T is equal to n_T. The width of a suffix subforest whose leftmost tree is rooted at v and whose parent node (if exists) degree is d, is equal to $d - r_v + 1$. The width of the suffix subforest corresponding to the tree T is 1. Then, the sum of the widths of the suffix subforests is

$$\begin{aligned}1 + \sum_{v\in T}\sum_{1\le k\le d_v}(d_v - k + 1) &= 1 + \sum_{v\in T}\sum_{1\le m\le d_v} m \\ &= 1 + \sum_{v\in T}\binom{d_v+1}{2}.\end{aligned}$$

There are $n_T + 1 - \ell_T$ suffix subforests that are also subtrees, then the sum of their widths is equal to $n_T + 1 - \ell_T$. Thus, the sum of widths of the elements of the set $\mathcal{S}(T)$ is

$$n_T + 1 + \sum_{v\in T}\binom{d_v+1}{2} - (n_T + 1 - \ell_T) = \ell_T + \sum_{v\in T}\binom{d_v+1}{2}.$$

(4) For each vertex v in T, there are r_v closed subforests whose rightmost tree is rooted at v. Their widths are equal to 1, 2, ..., r_v, respectively. Thus, the sum of widths of closed subforests equals

$$
\begin{aligned}
1+\sum_{v\in T}\sum_{1\le k\le d_v}\sum_{1\le m\le k} m &= 1+\sum_{v\in T}\sum_{1\le k\le d_v}\frac{k(k+1)}{2}\\
&= 1+\sum_{v\in T}\frac{d_v(d_v+1)(d_v+2)}{6}\\
&= 1+\sum_{v\in T}\binom{d_v+2}{3}.
\end{aligned}
$$

□

Lemma 3

$$\forall n\ge 0,\ \sum_{|T|=n+1}\sum_{v\in T}\binom{d_v+1}{2}=\binom{2n+1}{n+1}.$$

Proof

As stated in Lemma 2, for $n\ge 0$, $\sum_{|T|=n+1}\sum_{v\in T}\binom{d_v+1}{2}$ is the number of closed subforests in all trees T with $n+1$ vertices, not counting the whole tree T. Let $a_{n,k}$ be the number of vertices whose degree is greater or equal to k in all trees with n edges (thus $n+1$ vertices), for each $n\ge k\ge 0$.

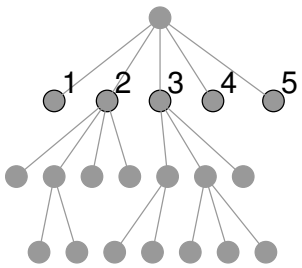

Fig. 1. Numbers of closed subforests whose root node of the rightmost tree is v_i respectively, for a sequence of sibling vertices (v_i).

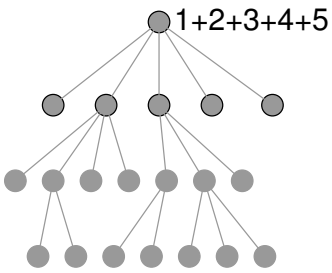

Fig. 2. Sum from 1 to the degree of the parent node of siblings v_i of Figure 1.

Figure 1 shows, for each vertex v_i of a sequence of siblings, the number of closed subforests whose root node of the rightmost tree is v_i. Figure 2 shows the sum from 1 to the degree of the parent node of these siblings. We can see that :

- counting 1 time each vertex $v \in T$ with $d_v \geq 1$ is equivalent to sum all vertices labeled 1 in T except the root node,
- counting 2 times each vertex $v \in T$ with $d_v \geq 2$ is equivalent to sum all vertices labeled 2 in T,
- $\vdots$
- counting $n_T - 1$ times each vertex $v \in T$ with $d_v \geq n_T - 1$ is equivalent to sum all vertices labeled $n_T - 1$ in T.

Hence,

$$\sum_{|T|=n+1} \sum_{v \in T} \binom{d_v + 1}{2} = \sum_{1 \leq k \leq n} k a_{n,k}.$$

Dershowitz and Zaks have shown in [4] that the number of vertices of degree k in all trees with n edges, denoted $d_{n,k}$, is equal to $\binom{2n-k-1}{n-1}$. Thus we can write $\forall n \geq 0,\ \forall 0 \leq k \leq n,\ a_{n,k} = \sum_{k \leq p \leq n} d_{n,p}$. We get

$$\begin{aligned} a_{n,k} &= \sum_{k \leq p \leq n} \binom{2n-p-1}{n-1} \\ &= \sum_{n-1 \leq q \leq 2n-k-1} \binom{q}{n-1} \qquad \text{by setting } q = 2n-p-1 \\ &= \binom{2n-k}{n} \qquad \text{by [7, formula (5.10) page 160].} \end{aligned}$$

Then we have:

$$\begin{aligned} \sum_{k \leq p \leq n} a_{n,p} &= \sum_{k \leq p \leq n} \binom{2n-p}{n} \\ &= \binom{2n-k+1}{n+1} \\ &= a_{n+1,k+1}. \end{aligned}$$

And now, we get:

$$\begin{aligned}\sum_{1\le k\le n} k a_{n,k} &= \sum_{1\le k\le n}\sum_{k\le p\le n} a_{n,p}\\ &= \sum_{1\le p\le n} a_{n+1,p+1}\\ &= a_{n+2,3}\\ &= \binom{2n+1}{n-1}.\end{aligned}$$

Hence,

$$\forall n \ge 0, \sum_{|T|=n+1}\sum_{v\in T}\binom{d_v+1}{2} = \binom{2n+1}{n+1}.$$

□

Lemma 4

$$\forall n \ge 0, \sum_{|T|=n+1}\sum_{v\in T}\binom{d_v+2}{3} = \binom{2n+2}{n-1}.$$

Proof

For $n \ge 0$, $\sum_{|T|=n+1}\sum_{v\in T}\binom{d_v+2}{3}$ is the sum of widths of closed subforests in all trees T with $n+1$ vertices, not counting the whole tree T. Let $b_{n,k}$ be the number of vertices whose rank is greater or equal to k in all trees with n edges (thus $n+1$ vertices), not counting the root node, for each $n \ge k \ge 0$.

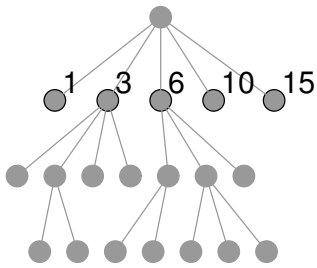

Fig. 3. Sums of widths of closed subforests whose root node of the rightmost tree is v_i respectively, for a sequence of sibling vertices (v_i).

Figure 3 shows, for each vertex v_i of a sequence of siblings, the sum of widths of closed subforests whose root node of the rightmost tree is v_i. Figure 4 shows sums from 1 to the rank of each of these siblings.

- We count 1 time each vertex $v \in T$ with $r_v \ge 1$ except the root node,
- we count 2 supplementary times each vertex $v \in T$ with $r_v \ge 2$,
- $\vdots$
- and we count $n_T - 1$ supplementary times each vertex $v \in T$ with $r_v \ge n_T - 1$.

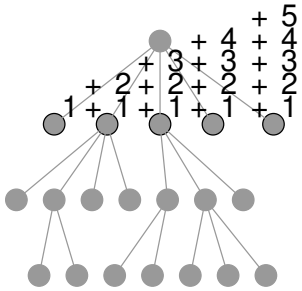

Fig. 4. Sums from 1 to ranks of siblings v_i of the Figure 3, respectively.

Thus, each vertex $v \in T$ whose rank is equal to k is counted $1+2+\ldots+k$ times, *i.e* as many time as the sum of widths of closed subforest whose rightmost tree is rooted at v. Thus, it is equivalent to sum the widths of closed subforests of T, except the width of T. Hence,

$$\forall n \geq 0, \sum_{|T|=n+1} \sum_{v \in T} \binom{d_v + 2}{3} = \sum_{1 \leq k \leq n} k b_{n,k}.$$

There are as many vertices of rank k in T not counting the root node as vertices of degree greater or equal to k in T, hence $b_{n,k} = \sum_{k \leq p \leq n} a_{n,p}$. Thus, we get

$$\begin{aligned} b_{n,k} &= a_{n+1,k+1} \\ &= \binom{2n-k+1}{n+1}. \end{aligned}$$

We notice the following equalities :

$$\begin{aligned} \sum_{k \leq p \leq n} b_{n,p} &= \sum_{k \leq p \leq n} \binom{2n-p+1}{n+1} \\ &= \binom{2n-k+2}{n+2} \\ &= b_{n+1,k+1}. \end{aligned}$$

And now, we get:

$$\begin{aligned}\sum_{1\le k\le n} k b_{n,k} &= \sum_{1\le k\le n}\sum_{k\le p\le n} b_{n,p}\\ &= \sum_{1\le p\le n} b_{n+1,p+1}\\ &= b_{n+2,3}\\ &= \binom{2n+2}{n-1}.\end{aligned}$$

Hence,

$$\forall n \ge 0, \sum_{|T|=n+1}\sum_{v\in T}\binom{d_v+2}{3} = \binom{2n+2}{n-1}.$$

□

Now we can prove Theorem 1.

Proof (*of Theorem 1*)

As we have seen in Lemma 1, the average number of operations for computing $\mathsf{align}(T_1, T_2)$, where $|T_1| = n+1$ and $|T_2| = m+1$ is on the order of

$$\frac{\sum_{|T_1|=n+1}\sum_{|T_2|=m+1}\mathsf{N}(T_1,T_2)}{\frac{1}{n+1}\binom{2n}{n}\frac{1}{m+1}\binom{2m}{m}}. \tag{1}$$

By using the second part of Lemma 1 and developing the double sum on n and m in the previous expression, we obtain an expression whose main components are the four following ones, where T stands for either T_1 or T_2.

- According to Lemma 2, the average number of subtrees and suffix subforests in a tree of size $n+1$ is

$$\frac{\sum_{|T|=n+1}(n+1+\ell_T-1)}{\frac{1}{n+1}\binom{2n}{n}} = \mathcal{O}(n).$$

- According to Lemmas 2 and 3, the average number of closed subforests in a tree of size $n+1$ is

$$\begin{aligned}\frac{\sum_{|T|=n+1}\left(1+\sum_{v\in T}\binom{d_v+1}{2}\right)}{\frac{1}{n+1}\binom{2n}{n}} &= \frac{\frac{1}{n+1}\binom{2n}{n}+\binom{2n+1}{n-1}}{\frac{1}{n+1}\binom{2n}{n}}\\ &= 1+\frac{(n+1)\ (2n+1)!\ n!\ n!}{(n-1)!\ (n+2)!\ (2n)!}\\ &= 1+\frac{n\ (2n+1)}{n+2}\\ &= \mathcal{O}(n).\end{aligned}$$

- According to Lemmas 2 and 3, the average sum of widths of subtrees and suffix subforests in a tree of size $n+1$ is

$$\frac{\sum_{|T|=n+1}\left(\ell_T+\sum_{v\in T}\binom{d_v+1}{2}\right)}{\frac{1}{n+1}\binom{2n}{n}}=\frac{\sum_{|T|=n+1}\ell_T}{\frac{1}{n+1}\binom{2n}{n}}+\frac{\binom{2n+1}{n-1}}{\frac{1}{n+1}\binom{2n}{n}}$$
$$=\mathcal{O}(n).$$

- According to Lemmas 2 and 4, the average sum of widths of closed subforests in a tree of size $n+1$ is

$$\frac{\sum_{|T|=n+1}\left(1+\sum_{v\in T}\binom{d_v+2}{3}\right)}{\frac{1}{n+1}\binom{2n}{n}}=\frac{\frac{1}{n+1}\binom{2n}{n}+\binom{2n+2}{n-1}}{\frac{1}{n+1}\binom{2n}{n}}$$
$$=1+\frac{(n+1)\ (2n+2)!\ n!\ n!}{(n-1)!\ (n+3)!\ (2n)!}$$
$$=1+\frac{n\ (2n+1)\ (2n+2)}{(n+2)\ (n+3)}$$
$$=\mathcal{O}(n).$$

Now, injecting these four results in the expression 1, we find that the average number of operations to compute alignment of two trees of sizes n and m, respectively, is on the order of $n.m$. □

3 RNA secondary structure alignment

We briefly present here the set of operations that are to be considered on RNA structures [9] and we outline the alignment algorithm given in [3,8,1]. Then we state its average complexity.

An RNA secondary structure without pseudoknots can be described as a sequence S on the alphabet {A, C, G, U} and a set P of pairs of positions $(i,j)\in[\![1,|S|]\!]^2$, where $|S|$ stands for the length of S, such that $i<j$ and such that for any $(i,j)\in P$, $(k,l)\in P$, $i\leq k$, one of the following assertions is true :

- $i=k$ and $j=l$,
- $i<j<k<l$,
- $i<k<l<j$.

The elements of P are also called *arcs*. Then a non-paired base is represented by a letter that is not incident to an arc, and a pair of bases is represented by an arc and its two incident letters.

Fig. 5. On the left : an RNA secondary structure without pseudoknot represented as a sequence and a set of arcs. On the right : the same RNA structure modeled by a tree.

From this representation, we can easily build a tree, by reading the sequence letter by letter and applying the following algorithm.

(1) If the current letter is not incident to an arc, a leaf is added and its label is the current letter. It becomes current vertex. The next vertex will be added as right sibling of this leaf, except if the next case is case 3.
(2) If the current letter is incident to the left branch of an arc, an internal node is added and its label is the concatenation of the current letter and the other letter that is incident to the same arc. It becomes the current vertex. The next vertex will be added as child of this internal node, except if the next case is case 3.
(3) If the current letter is incident to the right branch of an arc, no vertex is added. The parent node of the current vertex becomes the current vertex and the next vertex will be added as right sibling of it, except if the next case is case 3.

We obtain a forest, to which we add a root node. In such a tree, leaves correspond to non-paired bases and internal nodes correspond to pairs of bases, as shown in Figure 5.

Now we can define a set of edit operations on such trees, that correspond to the edit operations introduced by Jiang *et al.* on arc-annotated sequences. Moreover, we give a cost for each edit operation, depending on the labels of the involved vertices. These operations are illustrated on Figure 6 and detailed below.

- Base-mismatch stands for replacing a letter by another one. In trees, it corresponds to change the label of a leaf v, becoming the leaf v', and it is called *base-substitution*. It costs $\mathsf{BSub}(v, v')$.
- Base-deletion stands for either deleting a letter or inserting a letter. In trees, it corresponds to delete a leaf or insert a leaf v, respectively, and it is called *base-deletion* or *base-insertion*, respectively. It costs $\mathsf{BDel}(v)$ or $\mathsf{BIns}(v)$, respectively.
- Arc-mismatch stands for replacing the two letters incident to an arc, by two other letters. In trees, it corresponds to change the label of an internal node

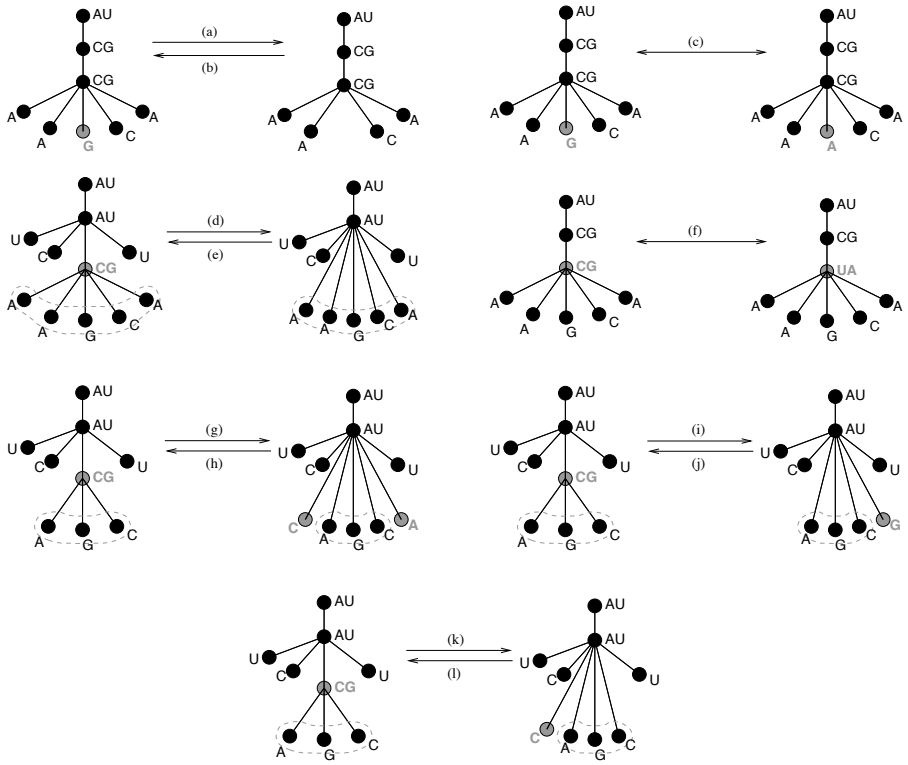

Fig. 6. Relevant edit operations for RNA, presented on trees. (a) Base-deletion. (b) Base-insertion. (c) Base-substitution. (d) Pair-deletion. (e) Pair-insertion. (f) Pair-substitution. (g) Scission. (h) Fusion. (i) Left-alteration. (j) Left-completion. (k) Right-alteration. (l) Right-completion.

v, becoming the internal node v', and it is called *pair-substitution*. It costs $\mathsf{PSub}(v, v')$.

- Arc-removing stands for either deleting or inserting an arc and its two incident bases. In trees, in the first case, it corresponds to delete an internal node v. The children of this node become children of its parent node. It is called *pair-deletion* and it costs $\mathsf{PDel}(v)$. In the second case, it correponds to insert an internal node v as parent node of a set of consecutive siblings. It is called *pair-insertion* and it costs $\mathsf{PIns}(v)$.
- Arc-breaking stands for either deleting an arc or inserting an arc. In trees, in the first case, it corresponds to replace an internal node v by two sibling leaves v' and v'_c such that v' is on the left of v'_c. The children of the internal node v become right siblings and left siblings v' and v'_c, respectively. It is called *scission* and it costs $\mathsf{Sci}(v, (v', v'_c))$. In the second case, it corresponds to replace two sibling leaves v and v_c such that v is on the left of v_c by an internal node v', siblings nodes between v and v_c becoming children of the internal node v'. It is called *fusion* and it costs $\mathsf{Fus}((v, v_c), v')$.
- Arc-altering stands for either deleting an arc and one if its incident letters, or inserting an arc and a letter incident to this arc. The deleted or inserted letter can be incident either to the left or to the right branch of the arc.

In trees, in the first case, it corresponds to replace an internal node v by a leaf v'. If the deleted letter is the left one, then children of the internal node v become left siblings of the leaf v'. It is called *left-alteration* and it costs $\mathsf{LAlt}(v, v')$. If the deleted letter is the right one, then children of the internal node v become right siblings of the leaf v'. It is called *right-alteration* and it costs $\mathsf{RAlt}(v, v')$. In the second case, it corresponds to replace a leaf v by an internal node v'. If the inserted letter is the left one, then left siblings of the leaf v become children of the internal node v'. It is called *left-completion* and it costs $\mathsf{LComp}(v, v')$. If the inserted letter is the right one, then right siblings of the leaf v become children of the internal node v'. It is called *right-completion* and it costs $\mathsf{RComp}(v, v')$.

For a vertex v, the letter v denotes both the vertex and its label. We assume that leaves are labeled in the alphabet Σ, and that internal nodes are labeled in $\Sigma.\Sigma$. We also assume that $- \notin \Sigma$ (then $\forall x \in \Sigma,\ x- \notin \Sigma.\Sigma,\ -x \notin \Sigma.\Sigma,\ -- \notin \Sigma.\Sigma$).

We also note $\Sigma^- = \Sigma \cup \{-\}$, $\Sigma^{-2*} = (\Sigma^- \times \Sigma^-) \setminus \{(-,-)\}$, $\Sigma_2 = \Sigma.\Sigma$, $\Sigma_2^- = \Sigma_2 \cup \{(-,-)\}$ and $\Sigma_2^{-2*} = (\Sigma_2^- \times \Sigma_2^-) \setminus \{((-,-),(-,-))\}$. Finally, we note $\widetilde{\Sigma} = \Sigma^{-2*} \cup \Sigma_2^{-2*} \cup (\Sigma_2 \times \Sigma^{-2*}) \cup (\Sigma^{-2*} \times \Sigma_2)$.

Now we define the following cost function γ :

$$\gamma : \quad \widetilde{\Sigma} \quad \longrightarrow \mathbb{R}$$

$$(a,b) \longmapsto \gamma(a,b) = \begin{cases} \mathsf{BDel}(a) & \text{if } (a,b) \in \Sigma \times \{-\} \\ & \textit{(base-deletion cost)} \\ \mathsf{BIns}(b) & \text{if } (a,b) \in \{-\} \times \Sigma \\ & \textit{(base-insertion cost)} \\ \mathsf{BSub}(a,b) & \text{if } (a,b) \in \Sigma \times \Sigma \\ & \textit{(base-substitution cost)} \\ \mathsf{PDel}(a) & \text{if } (a,b) \in \Sigma.\Sigma \times \{(-,-)\} \\ & \textit{(pair-deletion cost)} \\ \mathsf{PIns}(b) & \text{if } (a,b) \in \{(-,-)\} \times \Sigma.\Sigma \\ & \textit{(pair-insertion cost)} \\ \mathsf{PSub}(a,b) & \text{if } (a,b) \in \Sigma.\Sigma \times \Sigma.\Sigma \\ & \textit{(pair-substitution cost)} \\ \mathsf{Fus}(a,b) & \text{if } (a,b) \in (\Sigma \times \Sigma) \times \Sigma.\Sigma \\ & \textit{(fusion cost)} \\ \mathsf{Sci}(a,b) & \text{if } (a,b) \in \Sigma.\Sigma \times (\Sigma \times \Sigma) \\ & \textit{(scission cost)} \\ \mathsf{LAlt}(a,b) & \text{if } (a,b) \in \Sigma.\Sigma \times (\{-\} \times \Sigma) \\ & \textit{(left-alteration cost)} \\ \mathsf{RAlt}(a,b) & \text{if } (a,b) \in \Sigma.\Sigma \times (\Sigma \times \{-\}) \\ & \textit{(right-alteration cost)} \\ \mathsf{LComp}(a,b) & \text{if } (a,b) \in (\{-\} \times \Sigma) \times \Sigma.\Sigma \\ & \textit{(left-completion cost)} \\ \mathsf{RComp}(a,b) & \text{if } (a,b) \in (\Sigma \times \{-\}) \times \Sigma.\Sigma \\ & \textit{(right-completion cost)} \end{cases}$$

Now we give the recurrence relation to compute alignment distance between two RNA secondary structures modeled by trees. As for trees, the alignment algorithm is based on dynamic programming.

By convention,

$$\mathsf{align}(\varepsilon, \varepsilon) = 0$$

and, for any pair of closed subforests $v(f) \circ g$ and $v'(f') \circ g'$, where f, g, f' and g' are possibly empty,

$$\mathsf{align}(v(f) \circ g, v'(f') \circ g') =$$

$$
\min \begin{cases}
\gamma(v,-) + \mathsf{align}(g, v'(f') \circ g') \\
\qquad \text{if } v \text{ is a leaf} \\
\gamma(-,v') + \mathsf{align}(v(f) \circ g, g') \\
\qquad \text{if } v' \text{ is a leaf} \\
\gamma(v,v') + \mathsf{align}(g,g') \\
\qquad \text{if } v, v' \text{ are leaves} \\
\min\{\gamma(v,(-,-)) + \mathsf{align}(f,p') + \mathsf{align}(g,s') \mid p' \circ s' = v'(f') \circ g'\} \\
\qquad \text{if } v \text{ is an internal node} \\
\min\{\gamma((-,-),v') + \mathsf{align}(p,f') + \mathsf{align}(s,g') \mid p \circ s = v(f) \circ g\} \\
\qquad \text{if } v' \text{ is an internal node} \\
\gamma(v,v') + \mathsf{align}(f,f') + \mathsf{align}(g,g') \\
\qquad \text{if } v, v' \text{ are internal nodes} \\
\min\{\gamma(v,(v',v'_c)) + \mathsf{align}(f,p') + \mathsf{align}(g,s') \mid p' \circ v'_c \circ s' = g'\} \\
\qquad \text{if } v \text{ is an internal node, } v', v'_c \text{ are leaves} \\
\min\{\gamma((v,v_c),v') + \mathsf{align}(p,f') + \mathsf{align}(s,g') \mid p \circ v_c \circ s = g\} \\
\qquad \text{if } v, v_c \text{ are leaves, } v' \text{ is an internal node} \\
\min\{\gamma(v,(-,v'_c)) + \mathsf{align}(f,p') + \mathsf{align}(g,s') \mid p' \circ v'_c \circ s' = v'(f') \circ g'\} \\
\qquad \text{if } v \text{ is an internal node, } v'_c \text{ is a leaf} \\
\min\{\gamma(v,(v',-)) + \mathsf{align}(f,p') + \mathsf{align}(g,s') \mid p' \circ s' = g'\} \\
\qquad \text{if } v \text{ is an internal node, } v' \text{ is a leaf} \\
\min\{\gamma((-,v_c),v') + \mathsf{align}(p,f') + \mathsf{align}(s,g') \mid p \circ v_c \circ s = v(f) \circ g\} \\
\qquad \text{if } v_c \text{ is a leaf, } v' \text{ is an internal node} \\
\min\{\gamma((v,-),v') + \mathsf{align}(p,f') + \mathsf{align}(s,g') \mid p \circ s = g\} \\
\qquad \text{if } v \text{ is a leaf, } v' \text{ is an internal node}
\end{cases}
$$

Theorem 2 *The average complexity of the secondary structures alignment algorithm for two secondary structures S_1 and S_2 is in $\mathcal{O}(|S_1|.|S_2|)$, where $|S|$ stands for the length of the RNA sequence S.*

Proof The number of vertices in a tree modeling an RNA secondary structure is on the order of the length of the RNA sequence. Indeed, for an RNA sequence S of size n, there are at most $\frac{n}{2}$ pairs. Since each vertex of its corresponding tree T represent either a base-pair or a non-paired base, there are at least $\frac{n}{2}$

vertices and at most n vertices in T. Thus, the number of vertices in T is on the order of the size of the sequence S.

Now we can see that the pattern of that recurrence is the same as the one of Jiang-Wang-Zhang's recurrence. Moreover, the number of additional operations (that are involved in the additional cases) is of the same order as the number of the former ones. Thus, the total number of operations for aligning two trees or two RNA secondary structures only differ by a constant factor. □

References

[1] G. Blin, A. Denise, S. Dulucq, C. Herrbach, H. Touzet, Alignments of RNA structures, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2008, to appear.

[2] G. Blin, G. Fertin, I. Rusu, C. Sinoquet, Extending the Hardness of RNA Secondary Structure Comparison, in: B. Chen, M. Paterson, G. Zhang (eds.), The intErnational Symposium on Combinatorics, Algorithms, Probabilistic and Experimental methodologies (ESCAPE 2007), vol. 4614, Hangzhou, China, 2007.

[3] G. Blin, H. Touzet, How to Compare Arc-Annotated Sequences: The Alignment Hierarchy, in: F. Crestani, P. Ferragina, M. Sanderson (eds.), 13th International Symposium on String Processing and Information Retrieval (SPIRE 2006), vol. 4209, Glasgow, UK, 2006.

[4] N. Dershowitz, S. Zaks, Patterns in trees, Discrete Applied Mathematics 25 (1989) 241–255.

[5] S. Dulucq, L. Tichit, RNA secondary structure comparison: exact analysis of the Zhang-Shasha tree edit algorithm., Theoretical Computer Science 306 (1-3) (2003) 471–484.

[6] S. Dulucq, H. Touzet, Analysis of tree edit distance algorithms., in: Proc. 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003), 2003.

[7] R. L. Graham, D. E. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, 1994, second edition.

[8] C. Herrbach, A. Denise, S. Dulucq, H. Touzet, Alignment of RNA secondary structures using a full set of operations, Tech. rep., rapport de Recherche LRI n 1451 (2006).
URL `http://www.lri.fr/Rapports-internes/2006/RR1451.pdf`

[9] T. Jiang, G.-H. Lin, B. Ma, K. Zhang, A general edit distance between RNA structures., Journal of Computational Biology 9 (2) (2002) 371–388.

[10] T. Jiang, L. Wang, K. Zhang, Alignment of trees - an alternative to tree edit, Theoretical Computer Science 143 (1995) 137–148.

[11] P. N. Klein, Computing the edit-distance between unrooted ordered trees., in: Proc. 6th Annual European Symposium on Algorithms (ESA '98), 1998.

[12] B. A. Shapiro, An algorithm for comparing multiple RNA secondary structures., Computer Applications in the Biosciences 4 (3) (1988) 387–393.

[13] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems., SIAM J. Comput. 18 (6) (1989) 1245–1262.

[14] M. Zuker, D. Sankoff, RNA secondary structures and their prediction, Bull. Math. Biol. 46 (1984) 591–621.