Available online at www.sciencedirect.com
ScienceDirect
Journal of Discrete Algorithms 5 (2007) 696–705
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

Comparing similar ordered trees in linear-time

Hélène Touzet

LIFL, UMR CNRS 8022, Université Lille 1, 59 655 Villeneuve d'Ascq cedex, France

Received 5 December 2005; accepted 16 July 2006

Available online 20 October 2006

Abstract

We describe a linear-time algorithm for comparing two similar ordered rooted trees with node labels. The method for comparing trees is the usual tree edit distance. We show that an optimal mapping that uses at most k insertions or deletions can then be constructed in $O(nk^3)$ where n is the size of the trees. The approach is inspired by the Zhang–Shasha algorithm for tree edit distance in combination with an adequate pruning of the search space based on the tree edit graph.

Keywords: Ordered trees; Edit distance; Computational biology

1. Introduction

The problem of determining the similarity between two labeled ordered trees occurs in several areas of computer science: hierarchically structured data, image decomposition, RNA secondary structures in computational biology, ...

There are at least two paradigms to compare ordered trees: edit distance and alignment. The original edit distance is based on three edit operations on nodes—insertion, deletion and substitution—and operations can apply without any restriction in any order. Tai [1] introduced a first solution for this problem. Zhang and Shasha [2] gave a faster dynamic programming algorithm. The time required is $O(n^2 collapsed\_depth^2)$ where n is the size of the trees and *collapsed_depth* is a parameter that depends of the shape of the tree. The value of *collapsed_depth* is in $O(n)$, but it is bounded by the depth and by the number of leaves of the trees, which makes it smaller in average. Klein [3] proposed an improved algorithm in $O(n^3 \log(n))$. Further analyses of Zhang–Shasha and Klein approaches can be found in [4,5].

Alignment has been introduced by Jiang et al. [6] as an alternative to tree edit. All insertions should be performed before deletions. In other words, an alignment between two trees is obtained by applying insert operations on the two trees so they become isomorphic when labels are ignored. Alignment is less expressive than the edit distance approach. Fig. 1 provides a typical example where the alignment requires more edition operations than the edit distance. Moreover, Fig. 2 shows that there is no bounded ratio between the scores of the distance and the alignment. The best known algorithm for alignment runs in $O(n^2 d^2)$ where d is the maximal degree of the trees. So for trees with bounded degrees, the alignment algorithm turned to be quadratic. Recent results dealing with local and gapped alignments have been proposed in [7,8].

E-mail address: helene.touzet@lifl.fr.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2006.07.002

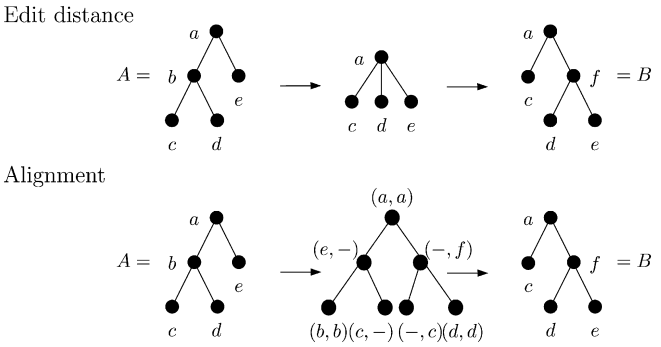

Fig. 1. Edit distance versus alignment 1—borrowed from [6]. The first picture shows the optimal mapping for the two trees A and B with the edit distance: the mapping contains a deletion and an insertion. That makes two errors. With the alignment, the optimal mapping involves four errors. The cost of the edit operations is defined as follows: $\texttt{sub} = \texttt{ins} = \texttt{del} = 1$.

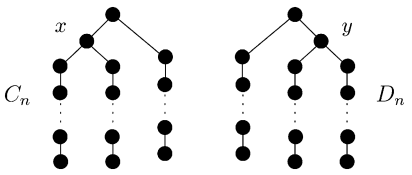

Fig. 2. Edit distance versus alignment 2. For each natural number n, define the trees C_n and D_n of height $n+1$ and of size $3n+2$ as displayed above. This family of trees shows that the difference between the distance and the optimal alignment can be linear in the size of the trees. The value of the edit distance between C_n and D_n does not depend of n: the optimal mapping is composed of the deletion of x followed by the insertion of y. As for the optimal alignment, the cost is in $\mathrm{O}(n)$, since it involves at least $n-1$ deletions.

In this article, we address the problem of comparing similar trees. We assume that the number of errors between the two input trees can be bounded in advance by a positive integer k. Errors are edit operations that affect the skeleton of the tree: insertions or deletions. This question of comparing similar trees occurs when comparing functionally related RNA secondary structures for instance. A fast algorithm has been proposed for tree alignment with k errors [9]: in this context, the method can construct an optimal alignment in $\mathrm{O}(n\log(n)d^3k^2)$ time. In particular, if both trees are of bounded degree the running time reduces to $\mathrm{O}(n\log(n)k^2)$.

We focus here on the tree edit distance paradigm with k errors. The good news is that it is then possible to convert the $\mathrm{O}(n^4)$ method of Zhang–Shasha into a linear-time algorithm. The approach is inspired by the known k-band method for an optimal alignment between similar strings and uses a graph-theoretical formalisation called *edit graph*. We show that if there is an optimal mapping between the two input trees that uses at most k errors, then the edit distance can be computed in $\mathrm{O}(nk^3)$.

2. Graph representation for the tree edit problem

Definition 1 *(Tree).* A *tree* is a node (called the root) connected to an ordered sequence of disjoint trees. Each node is assigned a label. ε denotes the empty tree.

Let A be a tree. In the sequel, we shall use the following notations:

- given a node x of A, $A(x)$ denotes the subtree of A rooted at x,
- $\texttt{size}(A)$ denotes the number of nodes of the tree A, and $\texttt{size}(x)$ is the size of $A(x)$,
- $\texttt{depth}(x)$ is the length of the path from the root of A to x,
- $\texttt{height}(x)$ is the length of the longest path originated from x. The height of a leaf is 0.

As usual, the edit distance relies on three elementary edit operations: the *substitution*, which consists in replacing a label of a node by another label, the *insertion* of a node, and the *deletion* of a node. Each edit operation is assigned a cost: sub, ins and del denote the costs for respectively substituting, inserting and deleting a node.

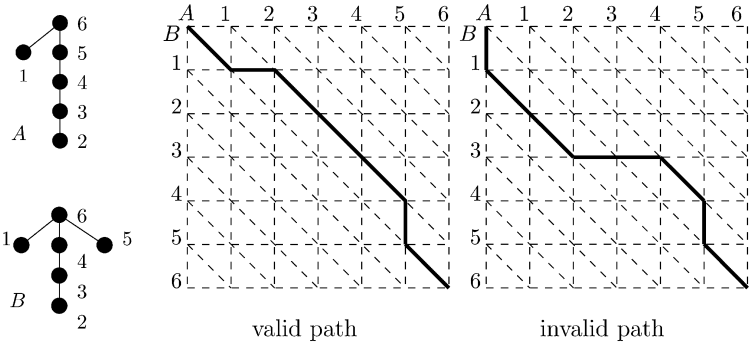

Fig. 3. Example of two paths in the edit graph for trees A and B, represented in postorder notation. The first path is a valid path. The other path is not correct: node 1 of A is matched with node 2 of B, and node 2 of A is matched with node 3 of B, which violates the tree structure.

Definition 2 *(Edit distance).* Let A and B be two trees. The *tree edit distance* between A and B, denoted $td(A, B)$, is the minimal cost of edit operations needed to transform A into B. For each pair of nodes (x, y) in $A \times B$, we write $td(x, y)$ for the distance between the two subtrees $A(x)$ and $B(y)$.

As mentioned in the introduction section, Zhang–Shasha and Klein proposed efficient algorithms for this problem. Both methods use dynamic programming decompositions on the input trees. We introduce an alternative formulation that is based on a graph-theoretical representation. It is known that the alignment of two strings can be reduced to the question of searching for an optimal path in a grid. Let u and v be two strings of length m and n respectively. The construction of the optimal alignment is based on a bidimensional array that can be represented as a weighted directed graph. Vertices are in $0..m \times 0..n$. Three arcs are originated from each vertex (x, y):

$(x, y) \rightsquigarrow (x - 1, y)$ labeled by `del`
$(x, y) \rightsquigarrow (x, y - 1)$ labeled by `ins`
$(x, y) \rightsquigarrow (x - 1, y - 1)$ labeled by $\texttt{subs}(x, y)$

The set of paths from the node (m, n) to the node $(0, 0)$ is exactly the set of all possible alignments between the two strings. The optimal alignment is the path of smallest weight. One can adapt this point of view to the problem of the edit distance for trees. This has been done in [10] or [11] for a restrained version of the edit distance: substitution can occurs only between nodes with the same depth. We make it more general here and apply the construction to the full tree edit distance. The motivating idea is that a mapping between two trees induces a sequence alignment between the two associated postorder traversal: enumerating the nodes of the subtrees and then the root. Of course every possible path in the grid does not lead to a correct tree mapping. The mapping should be consistent with the structures of the trees. If the node x of A is matched with the node y of B, then the subtrees $A(x)$ and $B(y)$ should be matched together too. Concerning the graph representation, it means that every path containing the diagonal arc $(x, y) \rightsquigarrow (x - 1, y - 1)$ should visit the vertex $(x - \texttt{size}(x), y - \texttt{size}(y))$ (see Fig. 3 for an example). We take into account this constraint and modify the edit graph consequently. The set of vertices constituting the grid is identical, as well as arcs for insertions and deletions. Only arcs corresponding to substitution operations are modified.

Definition 3 *(Tree edit graph).* Let A and B be two trees of size m and n respectively. Nodes of A and B are enumerated in the postorder notation, indices starting at 1. The *tree edit graph* of A and B is a weighted directed graph composed by nodes in $\{0, \ldots, m\} \times \{0, \ldots, n\}$ whose incident arcs are defined as follows:

deletion arc: $(x, y) \rightsquigarrow (x - 1, y)$ labeled by `del`
insertion arc: $(x, y) \rightsquigarrow (x, y - 1)$ labeled by `ins`
substitution arc: $(x, y) \rightsquigarrow \big(x - \texttt{size}(x), y - \texttt{size}(y)\big)$ labeled by $td(x, y)$

Fig. 4 gives an example of a tree edit graph. Before explaining how to compute the labels *td* of the substitution arcs, we establish that this construction is correct. This is the purpose of Lemmas 5 and 6.

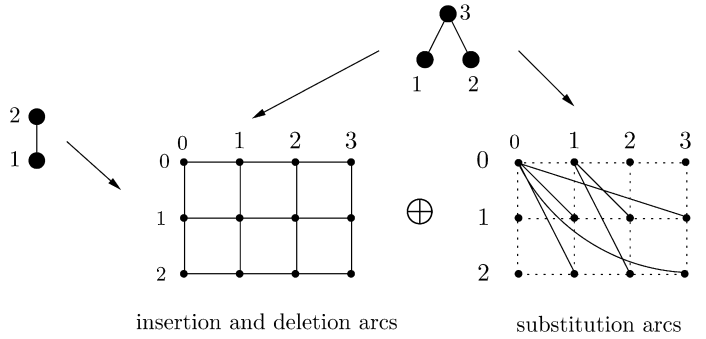

Fig. 4. Tree edit graph for two small trees (two nodes and three nodes respectively). For clarity's sake, the insertion and deletion arcs and the substitution arcs are drawn on two separate grids. The edit graph is obtained by overlaying the two grids. All arcs are oriented from the bottom-right node to the upper-left node.

Definition 4 *(Subgraph and top-level path).* Let A and B be two trees and let (x, y) be a pair of nodes of $A \times B$. We write G for the edit graph of A and B. The subgraph $G(x, y)$ associated to $A(x)$ and $B(y)$ is the restriction of G to nodes and arcs in $\{\mathtt{size}(x) - x..x\} \times \{\mathtt{size}(y) - y..y\}$ without the substitution arc $(x, y) \rightsquigarrow (x - \mathtt{size}(x), y - \mathtt{size}(y))$ and with an extra arc $(x, y) \rightsquigarrow (x - 1, y - 1)$ labeled by $\mathtt{sub}(x, y)$. A *top-level* path of $A(x)$ and $B(y)$ is an optimal path in $G(x, y)$ from the bottom-right corner to the upper-left corner.

Given a path P in the edit graph, define $M(P)$ as the set of vertices that are the origin of a diagonal arc:

$$M(P) = \big\{(x, y) \in A \times B, (x, y) \rightsquigarrow \big(x - \mathtt{size}(x), y - \mathtt{size}(y)\big) \in P\big\}$$

Lemma 5. *Let A and B be two trees and P a path for A and B. Then $M(P)$ is a mapping of A to B. Conversely, let M be a mapping of A to B such that*

$$(x, y) \in M \wedge (x', y') \in M \Rightarrow x \notin A(x') \wedge x' \notin A(x)$$

Then there is a path P in the edit graph from $(\mathtt{size}(A),\mathtt{size}(B))$ *to* $(0, 0)$ *such that $M = M(P)$.*

How to derive an optimal mapping from the edit graph? The mapping is no longer obtained with a single path, but with a series of top-level paths.

Lemma 6. *Let A and B be two trees. An optimal mapping can be built up recursively as follows*:

$$\mathtt{Mapping}(A, \varepsilon) = \emptyset$$
$$\mathtt{Mapping}(\varepsilon, B) = \emptyset$$
$$\mathtt{Mapping}(A, B) = M(P) \bigcup_{(x,y)\in M(P)} \mathtt{Mapping}\big(A(x), B(y)\big)$$

where P is an optimal path for A and B in the associated edit graph. The cost of the mapping is the weight of the top-level path for A and B.

At each step, the time needed to compute the top-level path for (x, y) is in $\mathrm{O}(\mathtt{size}(x) \times \mathtt{size}(y))$. Since a mapping contains only a linear number of matching pairs, the overall computation time is $\mathrm{O}(n^3)$.

We now come to the problem of determining the labels of the substitution edges in the edit graph. $td(x, y)$ is obtained as the weight of a top-level path for $A(x)$ and $B(y)$. Let (h, l) be the coordinates of the first node of $G(x, y)$: $h = x - \mathtt{size}(x)$, $l = y - \mathtt{size}(y)$. For each (i, j) in $G(x, y)$ distinct from (x, y), we denote $fd(i, j, h, l)$ the cost of an optimal path from (k, l) to (i, j). The usual dynamic programming algorithm for an optimal in a graph gives the following equations for fd.

$$fd(h, l, h, l) = 0$$

$$
\begin{aligned}
fd(i,l,h,l) &= fd(i-1,l,h,l) + \texttt{del} \\
fd(h,j,h,l) &= fd(h,j-1,h,l) + \texttt{ins} \\
fd(i,j,h,l) &= \min \begin{cases} fd(i-1,j,h,l) + \texttt{del} \\ fd(i,j-1,h,l) + \texttt{ins} \\ fd(i-\texttt{size}(i), j-\texttt{size}(j), h, l) + td(i,j) \end{cases}
\end{aligned} \tag{1}
$$

Readers that are familiar with the original formulation of Zhang–Shasha algorithm may note that Eq. (1) is the main recurrence formula for the computation of the tree edit distance (Lemmas 3 and 4 in [2]). The value of $fd(i, j, h, l)$ is exactly the distance between the subforest $[h..i]$ of A and the subforest $[l..j]$ in B, denoted $\texttt{forestdist}(h..i, l..j)$ in Zhang–Shasha article.

From Eq. (1), we can deduce the distance $td(x, y)$ for $A(x)$ and $B(y)$. This is

$$
td(x,y) = \min \begin{cases} fd(x-1,y,h,l) + \texttt{del} \\ fd(x,y-1,h,l) + \texttt{ins} \\ fd(x-1,y-1,h,l) + \texttt{sub}(x,y) \end{cases} \tag{2}
$$

Klein's algorithm is based on another strategy for the traversal of the grid for finding optimal paths. For the problem of the edit distance with k errors, we shall focus on the direct strategy of Zhang–Shasha.

3. Tree edit distance with k errors

The general idea of the algorithm with k errors is to prune the edit graph to keep only relevant vertices and arcs. For that we implement three optimisations, that are given by Lemmas 8, 10 and 11. The first property is analogous to the k-band alignment algorithm for strings of [12]. If the trees are similar, then the best mappings have their paths near the main diagonal in the edit graph. It means that is not necessary to build the entire graph.

Definition 7 *(k-strip).* Let k be a positive integer. For two trees A and B, we define the set of pairs of nodes $k\text{-strip}(A, B)$ as

$$k\text{-strip}(A,B) = \big\{(x,y) \in G(A,B), |x-y| \leqslant k\big\}$$

where $G(A, B)$ is the tree edit graph of A and B. We write $k\text{-strip}(x, y)$ for $k\text{-strip}(A(x), B(y))$.

Lemma 8. *Let A and B be two trees and let x and y be two nodes in A and B respectively. If there is a mapping between A and B which uses at most k errors and that visits the vertex (x, y) in the edit graph, then $(x, y) \in k\text{-strip}(A, B)$.*

Proof. Each mapping visiting the vertex (x, y) contains at least $|x - y|$ errors: $|x - y|$ insertions if $x \leqslant y$, or $|x - y|$ deletions if $y < x$. □

Definition 9 *(k-relevant).* Let k be a positive integer. Given two trees A and B and a pair of nodes (x, y) in $k\text{-strip}(A, B)$, the pair (x, y) is called *k-relevant* if the value

$$\big|\texttt{size}(A) - x - \texttt{size}(B) + y\big| + \big|\texttt{size}(x) - \texttt{size}(y)\big| + \big|x - \texttt{size}(x) - y + \texttt{size}(y)\big|$$

is lesser than k. Furthermore, we define the *maximal number of errors* $e(x, y)$ for $A(x)$ and $B(y)$ as

$$e(x,y) = k - \big|x - \texttt{size}(x) - y + \texttt{size}(y)\big| - \big|\texttt{size}(A) - x - \texttt{size}(B) + y\big|.$$

Lemma 10. *Let A and B be two trees and let x and y be two nodes in A and B respectively. If there is a mapping between A and B which uses at most k errors so that $A(x)$ is mapped to $B(y)$, then (x, y) is k-relevant and the number of errors to compare $A(x)$ and $B(y)$ is bounded by $e(x, y)$.*

Proof. If $A(x)$ is mapped to $B(y)$, then it implies that the path should visit four vertices in the edit graph: $(\texttt{size}(A), \texttt{size}(B))$, (x, y), $(x - \texttt{size}(x), y - \texttt{size}(y))$ and finally $(0, 0)$. The total amount of errors along this

path is greater than $|\mathtt{size}(A) - x - \mathtt{size}(B) + y| + |\mathtt{size}(x) - \mathtt{size}(y)| + |x - \mathtt{size}(x) - y + \mathtt{size}(y)|$. In particular, the number of errors outside $A(x) \times B(y)$ is greater than $|x - \mathtt{size}(x) - y + \mathtt{size}(y)| + |\mathtt{size}(A) - x - \mathtt{size}(B) + y|$. If the total number of errors should be bounded by k, it remains $e(x, y)$ errors for $A(x)$ and $B(y)$. □

Lemmas 8 and 10 lead to the following outline for the computation of the distance with k errors.

Procedure *Distance*
input: two trees A and B, integer k
output: fills up the array td. For each (x, y) of $A \times B \cap k\text{-}strip(A, B)$,
$td(x, y)$ is the distance between $A(x)$ and $B(y)$.The whole distance is $td(\mathtt{size}(A), \mathtt{size}(B))$.
for $(x, y) \in A \times B \cap k\text{-}strip(A, B)$ **do**
  if not k-relevant(x, y)
    then $td(x, y) \leftarrow +\infty$
    else $e \leftarrow |x - \mathtt{size}(x) - y + \mathtt{size}(y)| + |\mathtt{size}(A) - x - \mathtt{size}(B) + y|$
      compute $td(x, y)$ with e errors with Eqs. (1) and (2)

Eqs. (1) and (2) of the previous section have to be adapted to compute $td(x, y)$ for paths in $k\text{-}strip(A, B)$ and fd for paths $k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$. In each case, borders of the search space have to be delineated explicitly. This gives two functions `ForestDist1`, for fd, and `TreeDist1`, for td.

Function *ForestDist1*
input: $i \in A(x)$, $j \in B(y)$, integers e, k
output: compute the weight $fd(i, j)$ of an optimal path from (i, j) to $(x - \mathtt{size}(x), y - \mathtt{size}(y))$
/* initial cases */
if $i = x - \mathtt{size}(x)$ **and** $j = y - \mathtt{size}(y)$
  then return 0
if $i = x - \mathtt{size}(x)$
  then return $f(i, j - 1) + \mathtt{ins}$
if $j = y - \mathtt{size}(y)$
  then return $f(i - 1, j) + \mathtt{del}$
/* general case */
$fd(i, j) \leftarrow +\infty$
if $(i - \mathtt{size}(i), j - \mathtt{size}(j)) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$ /* substitution */
  then $r \leftarrow td(i, j) + fd(i - \mathtt{size}(i), j - \mathtt{size}(j))$
if $(i - 1, j) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$ /* deletion */
  then $r \leftarrow \min(r, fd(i - 1, j)) + \mathtt{del}$
if $(i, j - 1) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$ /* insertion */
  then $r \leftarrow \min(r, fd(i, j - 1)) + \mathtt{ins}$
return r

Note that the two last parameters for fd in Eq. (1) are omitted. $fd(i, j)$ stands here for $fd(i, j, x - \mathtt{size}(x), y - \mathtt{size}(y))$.

Function *TreeDist1*
input: $x \in A$, $y \in B$, integers e, k
output: compute the distance $td(x, y)$ between $A(x)$ and $B(y)$ with e errors
for $i \in x - \mathtt{size}(x)..x$ **do**
  for $j \in y - \mathtt{size}(y)..y$ such that $(i, j) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$ **do**
    compute $fd(i, j)$ with function ForestDist1
return $\min\{fd(x - 1, y) + \mathtt{del}, fd(x, y - 1) + \mathtt{ins}, fd(x - 1, y - 1) + \mathtt{sub}(x, y)\}$

It gives raise to a algorithm in $O(n^2k^2)$ that can be further improved with the following lemma.

Lemma 11. *Let A and B be two trees. Let x and i be two nodes in A, such that i is a descendant of x. If there exists a mapping between $A(x)$ and B which uses at most e errors and such that the associated top-level path contains a vertex of the form (i, j) with j in B, then* $\texttt{depth}(i) \leqslant \texttt{depth}(x) + e + 1$.

Proof. If (i, j) belongs to a top-level path for $A(x)$ and B, then no node along the path from x (x excluded) to i is involved in a substitution. So the top-level path contains at least $\texttt{depth}(i) - \texttt{depth}(x) - 1$ deletions. □

This lemma ensures that when calculating $td(x, y)$ with at most e errors for $A(x)$ and $B(y)$, then it is not necessary to inspect pairs (i, j) where $\texttt{depth}(i) > \texttt{depth}(x) + e + 1$.

Definition 12 *(Truncated tree).* Let A be a tree, x be a node in A. Let e be a natural number. The *truncated tree* $A(x, e)$ is defined in a unique way as the largest subtree of A whose root is x and whose height is e. We write $\texttt{size}(A, x, e)$ for the size of $A(x, e)$.

The functions `ForestDist1` and `TreeDist1` should be modified to take into account this new property.

Function *ForestDist2*
input: $i \in A(x)$, $j \in B(y)$, integers e, k
output: compute the weight $fd(i, j)$ of an optimal path
  from (i, j) to $(x - \texttt{size}(x), y - \texttt{size}(y))$
/* initial cases */
if $i = x - \texttt{size}(x)$ **and** $j = y - \texttt{size}(y)$
  then return 0
if $i = x - \texttt{size}(x)$
  then return $f(i, j - 1) + \texttt{ins}$
if $j = y - \texttt{size}(y)$
  then return $f(i - 1, j) + \texttt{del}$
/* general case: substitution, deletion or insertion */
$fd(i, j) \leftarrow +\infty$
if $(i - \texttt{size}(i), j - \texttt{size}(j)) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$
  then $r \leftarrow td(i, j) + fd(i - \texttt{size}(i), j - \texttt{size}(j))$
if $\texttt{depth}(i - 1) - \texttt{depth}(x) \leqslant e + 1$ and $(i - 1, j) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$
  then $r \leftarrow \min(r, fd(i - 1, j)) + \texttt{del}$
if $(i, j - 1) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$
  then $r \leftarrow \min(r, fd(i, j - 1)) + \texttt{ins}$
return r

Function *TreeDist2*
input: $x \in A$, $y \in B$, integers e, k
output: compute the distance $td(x, y)$ between $A(x)$ and $B(y)$
  with e errors, with optimization of Lemma 11
/* traversal of $A(x, e)$ */
for $i \in x - \texttt{size}(x)..x$ such that $\texttt{depth}(i) - \texttt{depth}(x) \leqslant e + 1$ **do**
  for $j \in y - \texttt{size}(y)..y$ such that $(i, j) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$ **do**
    compute $fd(i, j)$ with function ForestDist2
return $\min\{fd(x - 1, y) + \texttt{del}, fd(x, y - 1) + \texttt{ins}, fd(x - 1, y - 1) + \texttt{sub}(x, y)\}$

The time for the computation of $td(x, y)$ with `TreeDist2` is $O(\texttt{size}(A, x, k)k)$, instead of $O(\texttt{size}(x)k)$ for the function `TreeDist1`. It remains one question to be answered: how to implement it in a effective way. The

tricky point is the traversal of the truncated tree $A(x, e)$. So far, nodes of A have always been ranked and visited in postorder. We now have to be able to skip all nodes whose depth is greater than $e + 1$ in $A(x)$. For that, it is necessary to combine postorder and bottom-up traversals. We introduce two arrays that allow to translate the index of a node from the postorder notation to the bottom-up notation, and vice-versa: for each $i \in \{1..\texttt{size}(A)\}$, define

- $\texttt{bottom\_up}(i)$ as the index in the bottom-up notation of the node of rank i in the postorder notation,
- $\texttt{post\_order}(i)$ as the index in the postorder notation of the node of rank i in the bottom-up notation.

These two attributes may be computed with a linear-time pre-processing. The function `TreeDist3` is a new version of `TreeDist2` with full details.

Function *TreeDist3*
input: $x \in A$, $y \in B$, integers e, k
output: compute the distance $td(x, y)$ between $A(x)$ and $B(y)$
with e errors, using the optimization of Lemma 11—full details.
/* initialisation of i and *nextnode* */
$i \leftarrow x - \texttt{size}(x)$
nextnode $\leftarrow$ first node of depth $\texttt{depth}(x) + e + 1$ in $A(x)$
/*traversal of $A(x, e)$ */
while $i \leqslant x$ **do**
for $j \in y - \texttt{size}(y)..y$ such that $(i, j) \in k\text{-}strip(A, B) \cap e\text{-}strip(x, y)$ **do**
compute $fd(i, j)$ with function ForestDist2
/* computation of the index i of the next node of A to be visited */
if $\texttt{depth}(i) - \texttt{depth}(x) = e + 1$
then *nextnode* $\leftarrow \texttt{post\_order}(\texttt{bottom\_up}(i) + 1)$
if $\texttt{depth}(i + 1) - \texttt{depth}(x) > e + 1$
then $i \leftarrow$ *nextnode*
else $i \leftarrow i + 1$
return $\min\{fd(x - 1, y) + \texttt{del}, fd(x, y - 1) + \texttt{ins}, fd(x - 1, y - 1) + \texttt{sub}(x, y)\}$

The variable *nextnode* is introduced to perform the traversal of $A(x, e)$. The initialisation of *nextnode* is done in $O(A(x, e))$ using the `height` attribute: if the height of x is smaller than $e + 1$, then *nextnode* is not usefull. Otherwise pick up the first child z of x whose height is at least e, and repeat this process from z and its first child of height at least $e - 1$ until a node of depth $\texttt{depth}(x) + e + 1$ is reached. So it does not affect the total complexity of `TreeDist3`.

Procedure *FinalDistance*
input: two trees A and B, integer k
output: fills up the array td. For each (x, y) of $A \times B \cap k\text{-}strip(A, B)$,
$td(x, y)$ is the distance between $A(x)$ and $B(y)$. The whole distance
is $td(\texttt{size}(A), \texttt{size}(B))$.
for $(x, y) \in A \times B \cap k\text{-}strip(A, B)$ **do**
if not k-relevant(x, y)
then $td(x, y) \leftarrow +\infty$
else $e \leftarrow |x - \texttt{size}(x) - y + \texttt{size}(y)| + |\texttt{size}(A) - x - \texttt{size}(B) + y|$
compute $td(x, y)$ with function TreeDist3

We conclude with the complexity analysis of the procedure `FinalDistance`, that is based on the size of $A(x, e)$.

Lemma 13. *Let A be a tree. Let e be a positive integer.*

$$\sum_{x \in A} \texttt{size}(A, x, e) \leqslant (e + 1)\texttt{size}(A).$$

Proof. The proof is by induction on e and on the size of A. □

Theorem 14. *The procedure* `FinalDistance` *computes the tree edit distance with k errors in* $O(nk^3)$ *time and* $O(kn)$ *space. Moreover it is then possible to reconstruct an optimal mapping in* $O(nk^2)$.

Proof. The implementation of `Finaldistance` uses two bidimensional arrays *td* and *fd*. Both are indexed by $\{0, \ldots, \texttt{size}(A)\} \times \{-k, \ldots, k\}$:

- `td` stores the values $td(x, y)$ of the distance between $A(x)$ and $B(y)$ for (x, y) in k-*strip*(A, B). More precisely, `td[x,y-x]` equals $td(x, y)$.
- `fd` is a permanent array with temporary values. At each call of `TreeDist3` with parameters x and y, *fd* stores the values for optimal intermediate paths in $G(x, y)$. More precisely, `fd[i,j-i]` equals $fd(i, j)$, the weight of an optimal path from (i, j) to $(x - \texttt{size}(x), y - \texttt{size}(y))$.

The size of each array is $(2k + 1)n$. As noticed previously, the run time of `TreeDist3` for (x, y) is in $O(\texttt{size}(A, x, k)k)$. So the whole algorithm is in

$$\sum_{(x,y)\in k\text{-}strip(A,B)} \texttt{size}(A, x, k)k.$$

Lemma 13 yields the following bound:

$$(2k + 1)^2 \sum_{x\in A} \texttt{size}(A, x, k) \leqslant (k + 1)(2k + 1)k^2\, \texttt{size}(A).$$

Concerning the construction of an underlying optimal mapping, we have to consider all successive top-level paths for matching pairs of nodes x and y (as for the full edit distance, see Lemma 6). A property worth to notice is that the top-level path for $A(x)$ and $B(y)$ can be obtained in restricting the search to vertices in $A(x, k) \times B(y) \cap k\text{-}strip(A, B)$. It can be done in $O(k\,\texttt{size}(A, x, k))$. So the whole trace back needs only $O(nk^2)$ time. □

4. Final comment

Our algorithm can construct an optimal mapping for two trees A and B in $O(nk^3)$ time if the number of errors can be bounded in advance by k. When taking the trivial bound $k = \texttt{size}(A) + \texttt{size}(B)$, this leads to the same asymptotic complexity $O(n^4)$ as Zhang–Shasha algorithm. However the average complexity would be worse, because we do not use the optimization of Zhang–Shasha that consists in eliminating redundant computations between a pair of nodes and the pair of their leftmost children.

If k is not specified in advance, it is also possible to apply the same scheme as in [9,12]. The idea goes as follows. It uses iterated application of the distance with k errors with increasing values of k. The starting value for k is $|\texttt{size}(A) - \texttt{size}(B)|$ and then k is doubled at each step. The stopping condition for k is given by a relationship between the score and the number of errors.

References

[1] K. Tai, The tree-to-tree correction problem, Journal of the Association for Computing Machinery 26 (1979) 422–433.
[2] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM Journal of Computing 18 (6) (1989) 1245–1262.
[3] P. Klein, Computing the edit-distance between unrooted ordered trees, in: 6th European Symposium on Algorithms, 1998, pp. 91–102.
[4] 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.
[5] S. Dulucq, H. Touzet, Analysis of tree edit distance algorithms, in: Combinatorial Pattern Matching, in: Lecture Notes in Computer Science, vol. 2676, Springer, Berlin, 2003, pp. 83–95.
[6] T. Jiang, L. Wang, K. Zhang, Alignment of trees—an alternative to tree edit, Theoretical Computer Science 143 (1) (1995) 137–148.
[7] M. Höchsmann, T. Töller, R. Giegerich, S. Kurtz, Local similarity in RNA secondary structures, in: IEEE Bioinformatics Conference, 2003, pp. 159–168.

[8] J. Jansson, N.T. Hieu, W.-K. Sung, Local gapped subforest alignment and its application in finding RNA structural motifs, in: ISAAC, in: Lecture Notes in Computer Science, vol. 3341, Springer, Berlin, 2004, pp. 569–580.
[9] J. Jansson, A. Lingas, A fast algorithm for optimal alignment between similar ordered trees, Fundamenta Informaticae 56 (1,2) (2003) 105–120.
[10] S. Chawathe, Comparing hierarchical data in external memory, in: Twenty-Fifth International Conference on Very Large Data Bases, 1999, pp. 90–101.
[11] G. Valiente, Algorithms on Trees and Graphs, Springer, Berlin, 2002.
[12] J. Setubal, J. Meidanis, Introduction to Computational Biology, International Thomson Publishing Company, 1997.