Decomposition algorithms for the tree edit distance problem

Serge Dulucq

LaBRI - UMR CNRS 5800 - Université Bordeaux I
33 405 Talence cedex, France
Serge.Dulucq@labri.fr

Hélène Touzet

LIFL - UMR CNRS 8022 - Université Lille 1
59 655 Villeneuve d'Ascq cedex, France
Helene.Touzet@lifl.fr

Abstract

We study the behavior of dynamic programming methods for the tree edit distance problem, such as [5,13]. We show that those two algorithms may be described as *decomposition strategies*. We introduce the general framework of *cover strategies*, and we provide an exact characterization of the complexity of cover strategies. This analysis allows us to define a new tree edit distance algorithm, that is optimal for cover strategies.

1 Introduction

Many processes can be described as transformations of ordered trees. Examples include the analysis of hierarchically structured data [1], such as XML documents, or comparison of RNA secondary structures in computational biology [8]. One way of comparing two ordered trees is by measuring their edit distance: the minimum cost to transform one tree into another using elementary operations. There are many variants of tree edit distance: alignment [4], constrained edit distances [1,7,14], distance with non-linear gap costs [10], ... We focus here on the general editing problem: insertions and deletions may take place in any order at any node within the tree. In 1979, Tai [9] introduced an algorithm for this problem with run time in $O(n^6)$. Many others solution have been designed, based on back tracking, divide-and-conquer,... (see [11] for a survey). Amongst these paradigms, dynamic programming appears to be a fruitful approach. There are currently two main dynamic programming algorithms for solving the tree edit distance problem: Zhang-Shasha's [13] and Klein's [5]. Both algorithms may be seen as an extension of the basic string edit distance algorithm. The difference lies in the set of sub-forests that are involved in the decomposition. This leads to different complexities: Zhang-Shasha is in $O(n^4)$ in worst case, as Klein is in $O(n^3 \log(n))$. However, this does not mean that Klein's approach is strictly better than Zhang-Shasha's. Depending on the input trees, Zhang and Shasha's

algorithm may be faster. The performance of each algorithm depends on the shape of the two trees to be compared.

The purpose of this paper is to present a general analysis of dynamic programming for tree edit distance algorithms. For that, we introduce a class of tree decompositions, called *cover strategies*, which involves Zhang-Shasha and Klein algorithms. We study the complexity of those decompositions by counting the exact number of distinct recursive calls for the underlying corresponding algorithm. As a corollary, this gives the number of recursive calls for Zhang-Shasha, that was a known result [13], and for Klein, that was not known. In the last section, we take advantage of this analysis to define a new edit distance algorithm for trees, which improves Zhang-Shasha and Klein algorithms with respect to the number of recursive calls.

2 Edit distance for trees and forests

2.1 Definitions

Definition 1 (Trees and forests) *A* tree *is a node (called the root) connected to an ordered sequence of disjoint trees. Such a sequence is called a* forest. *We write* $\ell(A_1 \circ \cdots \circ A_n)$ *for the tree composed of the node* ℓ *connected to the sequence of trees* $A_1, \ldots, A_n$.

This definition assumes that trees are ordered trees, and that the nodes are labeled. When it is clear from the context, we shall not distinguish between a node and its label. Trees with labeled edges can by handled similarly. In the sequel, we may use the word *forest* for denoting both forests and trees, a tree being a sequence reduced to a single element. We introduce some classical notations for trees and forests.

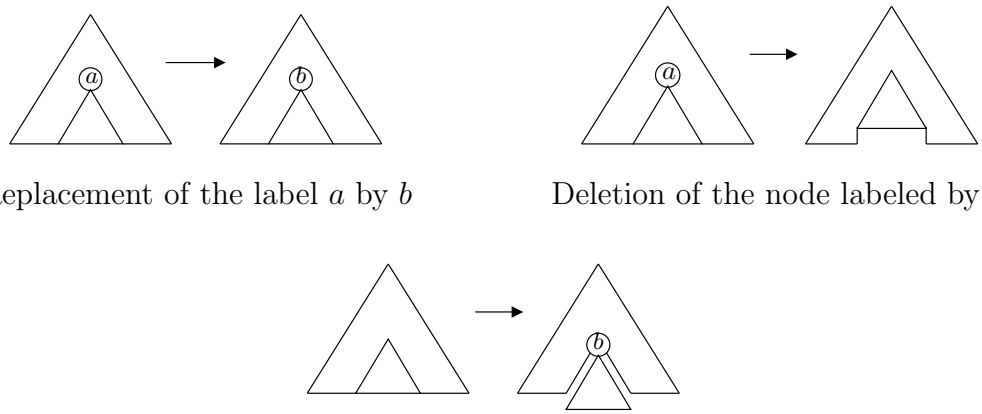
Notation 1 *Let* F *be a forest.*

- $|F|$ *denotes the size of* F*, that is the number of nodes of the forest* F*,*
- $\texttt{\#leaves}(F)$ *denote the number of leaves of* F*,*
- $\texttt{height}(F)$ *denotes the height of* F*, that is the maximal height of the trees composing* F*,*
- $F(i)$*, where* i *is a node of* F*, denotes the subtree of* F *rooted at* i*,*
- $deg(i)$ *is the degree of* i*, that is the number of children of* i*,*
- $\mathsf{F}(F)$ *is the set of all sub-forests of* F*.*

The edit distance relies on three elementary edit operations:

- *replacement*, which consists in replacing a label of a node by another label. This operation does not affect the skeleton of the tree,
- *insertion* of a node,
- *deletion* of a node.

Those three operations are depicted in Figure 1. Each edit operation is assigned a cost: c_r, c_i and c_d denote the costs for (respectively) renaming a label, inserting and deleting a node.



Replacement of the label a by b Deletion of the node labeled by a

Insertion of a node, labeled by b

Fig. 1. Edit operations for the tree edit distance problem

Definition 2 (Edit distance) *Let F and G be two forests. The* edit distance *between F and G, denoted $d(F, G)$, is the minimal cost of edit operations needed to transform F into G.*

2.2 Zhang and Shasha's algorithm

Before investigating Zhang and Shasha's algorithm for trees, we recall the usual well-known dynamic programming algorithm for the string editing problem [6,12]. It proceeds by solving the more general problem of measuring the distance for all pairs of prefixes of the two strings.

$$d(ux, vy) = \min \begin{cases} c_d(x) + d(u, vy) & - \text{ deletion of } x \\ c_i(y) + d(ux, v) & - \text{ insertion of } y \\ c_r(x, y) + d(u, v) & - \text{ replacement of } x \text{ by } y \end{cases}$$

where u and v are strings, x and y are alphabet symbols. Implementation is completed with a two-dimensional table. This gives a $O(n^2)$ solution. As for strings, the tree edit distance may be computed with dynamic programming techniques. The first decomposition applies necessarily to the roots of the trees:

$$d(\ell(f), \ell'(g)) = \min \begin{cases} c_d(\ell) + d(f, \ell'(g)) & - \text{ deletion of } \ell \\ c_i(\ell') + d(\ell(f), g) & - \text{ insertion of } \ell' \\ c_r(\ell, \ell') + d(f, g) & - \text{ replacement of } \ell \text{ by } \ell' \end{cases} \quad (1)$$

We now have to solve the problem for forests. Deletion and insertion cases are similar to the recurrence formulae for strings. Replacement generates two novel recursive calls: one call for the subtrees of the replaced node, that have to match together, and one call for the remaining nodes of the forests.

$$d(t \circ \ell(f), v \circ \ell'(g)) \; = \; \min \begin{cases} c_d(\ell) + d(t \circ f, v \circ \ell'(g)) & - \text{ deletion of } \ell \\ c_i(\ell') + d(\circ \ell(f), v \circ g) & - \text{ insertion of } \ell' \\ d(\ell(f), \ell'(g)) + d(t, v) & - \text{ replacement of } \ell \text{ by } \ell' \end{cases} \tag{2}$$

The sub-forests that can occur in the recursion are either subtrees, or prefixes of subtrees. We name it *leftmost forests*.

Definition 3 (Leftmost Forests) *Let A be a tree. The set of* leftmost sub-forests of A *is the least set satisfying*

- *for each node i of A, $A(i)$ is a leftmost forest,*
- *if $t \circ \ell(g)$ is a leftmost sub-forest, then $t \circ g$ is a leftmost sub-forest too.*

We write $\texttt{\#left}(A)$ *for the number of leftmost sub-forests of A.*

The number of leftmost sub-forests of A is bounded by $n(n+1)/2$, where n is the size of A. It is possible to get a better majorization of that value with the definition of the *collapsed depth*.

Definition 4 (collapsed depth [13]) *Let A be a tree. The set* $\text{keyroots}(A)$ *is defined as* $\text{keyroots}(A) = \{\text{root}(A)\} \cup \{i \in A, \; i \text{ has a left sibling}\}$. *For each node i of A,* $\text{collapsed\_depth}(i)$ *is the number of keyroot ancestors of i and*

$$\text{collapsed\_depth}(A) = \max\{\text{collapsed\_depth}(i), \; i \in A\}.$$

Proposition 5 ([13]) *For any tree A,* $\texttt{\#left}(A) \leq |A| \times \textit{collapsed\_depth}(A)$.

This proposition yields an upper bound in $O(n^4)$ for Zhang-Shasha's algorithm. This result appears to be a least upper bound in the worst case. Consider filiform binary trees b_n of size $2n+1$: each internal nodes has exactly two children and he leftmost child is a leaf. That is $b_1 = \bullet$ and $b_{n+1} = \bullet(\bullet \circ b_n)$. The number of leftmost sub-forests is $(2n^2+1)$ for a tree and so the number of sub-forests for the pair of trees is in $O(n^4)$.

However the average complexity is better. It can be shown that $|A| \times \textit{collapsed\_depth}(A) \leq |A| \min(\texttt{\#leaves}(A), \texttt{height}(A))$, and the average complexity is in $O(n^3)$ [2].

We introduce now a variation of Zhang-Shasha's algorithm, that comes from a simple but crucial remark. For string edit distance, it is likely to develop an alternative dynamic programming algorithm by constructing the distance for all pairs of suffixes. This gives the following recursive relationship:

$$d(xu, yv) = \min \begin{cases} c_d(x) + d(u, yv) & - \text{ deletion of } x \\ c_i(y) + d(xu, v) & - \text{ insertion of } y \\ c_r(x, y) + d(u, v) & - \text{ replacement of } x \text{ by } y \end{cases}$$

The same point of view is applicable to trees and forests. A forest can be decomposed *from the left* instead of being decomposed *from the right*.

$$d(\ell(f)\circ t, \ell'(g)\circ v) \;=$$
$$\min\begin{cases} c_d(\ell) + d(f\circ t, \ell'(g)\circ v) & - \text{ deletion of } \ell \\ c_i(\ell') + d(\ell(f)\circ t, g\circ v) & - \text{ insertion of } \ell' \\ d(\ell(f),\ell'(g)) + d(t,v) & - \text{ replacement of } \ell \text{ by } \ell' \end{cases} \tag{3}$$

This alternative recursive scheme is based on suffixes and suffixes of subtrees. It gives rise to rightmost forests.

Definition 6 (Rightmost Forests) *Let A be a tree. The set of* rightmost sub-forests of A *is the least set satisfying*

- *for each node i of A, $A(i)$ is a rightmost forest,*
- *if $\ell(g)\circ t$ is a rightmost sub-forest, then $g\circ t$ is a rightmost sub-forest too.*

We write `#right`(A) *for the number of rightmost sub-forests of A.*

We call decomposition according to Equation 2, involving leftmost forests, a *right* decomposition, and decomposition according to Equation 3, involving rightmost forests, a *left* decomposition. In the sequel, we use the word *direction* to indicate that the decomposition is left or right. For strings, left and right decompositions are equivalent, since the number of pairs of prefixes equals the number of pairs of suffixes. This equivalence is no longer valid for tree edit distance. The choice between Equations 2 and 3 may lead to different numbers of recursive calls. The asymptotic and average complexity of both algorithms, measured by the number of rightmost or leftmost sub-forests, are of course identical. However, when you analyze trees individually, it can induce a significant difference, due to the shape of the tree. For example, for the family of filiform binary trees, each tree b_n generates $3n+1$ rightmost sub-forests only. Figure 2 shows all leftmost and rightmost sub-forests for $n = 2$. So when you compute the distance $d(b_n, b_n)$, the complexity decreases from $O(n^4)$ to $O(n^2)$ for this example. This remark is the basis for the definition of Klein's algorithms.

2.3 Klein's algorithm

Klein's algorithm brings two new ideas: alternating the directions of decomposition in the dynamic programming scheme, and adding the employment of a tree decomposition into heavy paths. The concept of heavy paths was introduced by Harel and Tarjan [3].

Definition 7 (heavy paths [3]) *Given a rooted tree A, define the weight of each node i of A be the size $A(i)$. For each internal node i, let* $\mathrm{heavy}(i)$ *denote the child of i having greatest weight. The sequence of nodes $i, \mathrm{heavy}(i), \mathrm{heavy}(\mathrm{heavy}(i)), \ldots$ defines a descending path which is called the heavy path.*

Klein's algorithm uses the heavy path as a guide for the decomposition of the tree:

(1) let $(\ell(f)\circ t, F')$ be the pair of forests to compare,
(2) compute the heavy path P for $\ell(f)\circ t$,

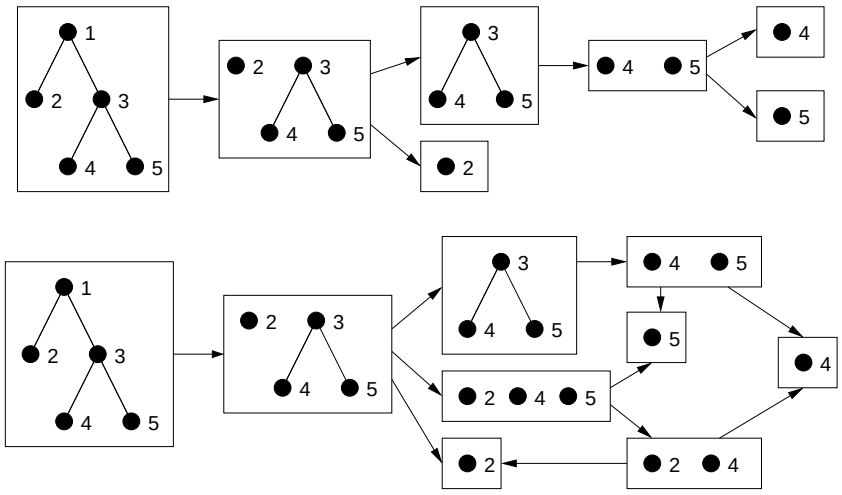


Fig. 2. These two graphics show two strategies of decomposition for the five-node tree $b_2 = \bullet(\bullet \circ \bullet(\bullet \circ \bullet))$. In the first case, all decompositions are left decompositions (according to Equation 3), and in the latter case, all decompositions are right decompositions (according to Equation 2). This gives respectively 7 and 9 sub-forests.

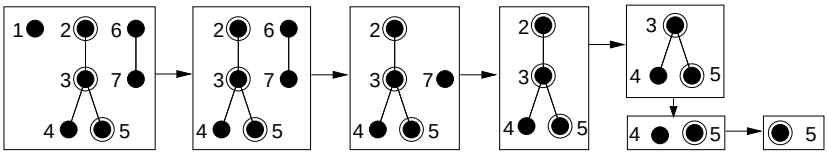


Fig. 3. Example of decomposition for Klein's algorithm. For each step, the nodes on the heavy path are indicated by circles.

(3) if ℓ belongs to P, apply Equation 2, otherwise apply Equation 3,
(4) apply this scheme recursively to all sub-forests of $\ell(f) \circ t$.

Figure 3 shows an example of Klein's decomposition for a tree. In Zhang-Shasha's algorithm, the analogue of decomposition into heavy paths might be called "leftmost"paths. The leftmost path descends via leftmost children. Decomposition into heavy paths has a great benefit for a single tree: it generates at most $O(n \log(n))$ sub-forests, instead of $O(n^2)$ leftmost or rightmost sub-forests in Zhang-Shasha's algorithm. The number of sub-forests for the other tree remains in $O(n^2)$.

Proposition 8 ([5]) *Klein's algorithm is in* $O(n^3 \log(n))$.

Proposition 8 does not mean that Klein's approach is always better than Zhang-Shasha's. Depending on the input trees, Zhang and Shasha's algorithm may be faster. For example, for the pair of trees (A, B) of Figure 4, Klein's approach needs 84 distinct recursive calls, whereas the Zhang-Shasha's approach needs only 72 distinct recursive calls.

The reason comes from the asymmetrical construction of Klein's algorithms. Only one tree is affected by the heavy path decomposition. The number of sub-forests for the second tree may be greater for Klein's algorithm than for Zhang-Shasha's. This is the price to pay for the clever decomposition of the first tree.

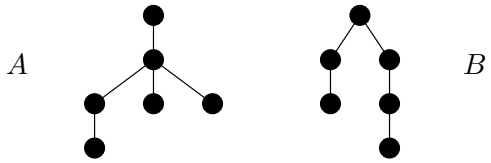


Fig. 4. (A, B)

3 Decomposition strategies

To make the definition of alternation of Equations 2 and 3 more formal, we introduce the framework of *decomposition strategies*. We then give several general properties of decomposition strategies.

Definition 9 (Strategy) *Let F and G be two forests. A* strategy *is a mapping from $\mathsf{F}(F) \times \mathsf{F}(G)$ to $\{left, right\}$.*

Each strategy is associated with a specific set of recursive calls. We name *relevant forests* the forests that are involved in these recursive calls. This terminology is adapted from [5], where it is used for a single tree. We consider here relevant forests for the whole pair of trees.

Definition 10 (Relevant Forests) *Let (F, G) be a pair of forests provided with a strategy ϕ. The set $\mathsf{R}(F, G)$ of* relevant *forests is defined as the least subset of $\mathsf{F}(F) \times \mathsf{F}(G)$ such that if the decomposition of (F, G) meets the pair (F', G'), then (F', G') belongs to $\mathsf{R}(F, G)$.*

For instance, for Zhang-Shasha's algorithm, the set of relevant forests is the set of leftmost forests.

Lemma 11 *Let (F, F') be a pair of forests provided with a strategy ϕ. The set $\mathsf{R}(F, F')$ of* relevant forests *satisfies:*

- *If F or F' is an empty forest:*

$$\mathsf{R}(\varepsilon, \ell(g) \circ t) = \{(\varepsilon, \ell(g) \circ t)\} \cup \mathsf{R}(\varepsilon, g \circ t) \text{ whenever } \phi(\varepsilon, \ell(g) \circ t) = \text{left}$$

$$\mathsf{R}(\varepsilon, t \circ \ell(g)) = \{(\varepsilon, t \circ \ell(g))\} \cup \mathsf{R}(\varepsilon, t \circ g) \text{ whenever } \phi(\varepsilon, t \circ \ell(g)) = \text{right}$$

$$\mathsf{R}(\ell(g) \circ t, \varepsilon) = \{(\ell(g) \circ t, \varepsilon)\} \cup \mathsf{R}(g \circ t, \varepsilon) \text{ whenever } \phi(\ell(g) \circ t, \varepsilon) = \text{left}$$

$$\mathsf{R}(t \circ \ell(g), \varepsilon) = \{(t \circ \ell(g), \varepsilon)\} \cup \mathsf{R}(t \circ g, \varepsilon) \text{ whenever } \phi(t \circ \ell(g), \varepsilon) = \text{right}$$

$$\mathsf{R}(\varepsilon, \varepsilon) = \emptyset$$

- *If $(F, F') = (\ell(g) \circ t, \ell'(h) \circ u)$ and $\phi(F, F') = \text{left}$, then $\mathsf{R}(F, F')$ is*

$$\{(F, F')\} \cup \mathsf{R}(g \circ t, F') \cup \mathsf{R}(F, h \circ u) \cup \mathsf{R}(\ell(g), \ell'(h)) \cup \mathsf{R}(t, u)$$

- *If* $(F, F') = (t \circ \ell(g), u \circ \ell'(h))$ *and* $\phi(F, F') = \text{right}$*, then* $\mathsf{R}(F, F')$ *is*

$$\{(F, F')\} \cup \mathsf{R}(t \circ g, F') \cup \mathsf{R}(F, u \circ h) \cup \mathsf{R}(\ell(g), \ell'(h)) \cup \mathsf{R}(t, u)$$

PROOF. By induction on $|F| + |F'|$, using Equations 1, 2 and 3. □

We write $\mathsf{R}(F)$ and $\mathsf{R}(G)$ to denote the projection of $\mathsf{R}(F, G)$ on $\mathsf{F}(F)$ and $\mathsf{F}(G)$ respectively.

Lemma 12

$$\mathsf{R}(\ell(g)) = \{\ell(g)\} \cup \mathsf{R}(g) \text{ no matter what the direction is}$$

$$\mathsf{R}(\ell(g) \circ t) = \{\ell(g) \circ t\} \cup \mathsf{R}(g \circ t) \cup \mathsf{R}(\ell(g)) \cup \mathsf{R}(t) \text{ if the direction is left}$$

$$\mathsf{R}(t \circ \ell(g)) = \{t \circ \ell(g)\} \cup \mathsf{R}(t \circ g) \cup \mathsf{R}(\ell(g)) \cup \mathsf{R}(t) \text{ if the direction is right}$$

PROOF. Straightforward implication of Lemma 11. □

Lemma 13 *Let* F *and* G *be two forests. For any strategy* ϕ*, for any nodes* i *of* F *and* j *of* G*,* $(F(i), G(j))$ *is an element of* $\mathsf{R}(F, G)$*.*

PROOF. By induction on the size of F and G. □

Given a decomposition strategy, the number of relevant sub-forests is a measure of the complexity of the associated edit distance algorithm.

Notation 2 *We denote* `#rel` *the number of relevant forests:*

- $\texttt{\#rel}(F, G)$ *is the cardinality of* $\mathsf{R}(F, G)$*,*
- $\texttt{\#rel}(F)$ *is the cardinality of* $\mathsf{R}(F)$*.*

Lemma 14 *Given a tree* $A = \ell(A_1 \circ \cdots \circ A_n)$*, for any strategy we have*

$$\texttt{\#rel}(A) \geq |A| - |A_i| + \texttt{\#rel}(A_1) + \cdots + \texttt{\#rel}(A_n),$$

where $i \in [1..n]$ *is such that the size of* A_i *is maximal.*

PROOF. Let $F = A_1 \circ \cdots \circ A_n$. We first prove that

$$\texttt{\#rel}(F) \geq |F| - |A_i| + \texttt{\#rel}(A_1) + \cdots + \texttt{\#rel}(A_n) \tag{4}$$

The proof is by induction on n. If $n = 1$, then the result is direct. If $n > 1$, assume that a left operation is applied to F (the other case with a right operation is identical). Let ℓ, g, t such that $A_1 = \ell(g)$ and $t = A_2 \circ \cdots \circ A_n$. By Lemma 12, we have

$$\mathsf{R}(F) = \{F\} \cup \mathsf{R}(A_1) \cup \mathsf{R}(T) \cup \mathsf{R}(g \circ t) \tag{5}$$

It is possible to prove by induction on $|g| + |t|$ that the number of relevant forests of $g \circ t$ containing both nodes of g and nodes of t is greater than $\min\{|g|, |t|\}$. Therefore Equation 5 implies

$$\texttt{\#rel}(F) \geq 1 + \texttt{\#rel}(A_1) + \texttt{\#rel}(t) + \min\{|g|, |t|\} \tag{6}$$

Let $j \in [2..n]$ such that A_j has the maximal size among $\{A_2, \ldots, A_n\}$. Applying induction hypothesis for t, we have $\texttt{\#rel}(t) \geq |t| - |A_j| + \texttt{\#rel}(A_2) + \cdots + \texttt{\#rel}(A_n)$. So Equation 6 becomes

$$\texttt{\#rel}(F) \geq 1 + \texttt{\#rel}(A_1) + \cdots + \texttt{\#rel}(A_n) + |t| - |A_j| + \min\{|g|, |t|\} \tag{7}$$

To establish Equation 4, it remains to verify that

$$1 + |t| - |A_j| + \min\{|g|, |t|\} \geq |F| - |A_i| \tag{8}$$

There are two cases, depending on the size of g and t. In the first case, if $|g| \leq |t|$, then $1 + |t| + \min\{|g|, |t|\} = |F|$. Since $|A_j| \leq |A_i|$, it follows that $1 + |t| + \min\{|g|, |t|\} - |A_j| \geq |F| - |A_i|$. In the latter case, if $|t| < |g|$, then A_1 is the largest subtree of A. So $i = 1$ and $|F| - |A_i| = |t|$, that implies $1 + |t| - |A_j| + \min\{|g|, |t|\} \geq |F| - |A_i|$.

We now show how Equation 4 gives the expected result. By Lemma 12 again, we have $\mathsf{R}(A) = \{A\} \cup \mathsf{R}(F)$. Hence

$$\begin{aligned}
\texttt{\#rel}(A) &= 1 + \texttt{\#rel}(F) \\
&\geq 1 + |F| - |A_i| + \texttt{\#rel}(A_1) + \cdots + \texttt{\#rel}(A_n) \text{ (Equation 4)} \\
&= |A| + \texttt{\#rel}(A_1) + \cdots + \texttt{\#rel}(A_n)
\end{aligned}$$

This concludes the proof. □

Lemma 15 *For every natural number n, there exists a tree A of size n such that for any strategy, $\texttt{\#rel}(A)$ has a lower bound in $O(n \log(n))$.*

PROOF. Let T_n be a complete balanced binary tree of size n. We prove by induction on n that

$$\texttt{\#rel}(T_n) \geq \frac{(n+1)\log_2(n+1)}{2} \tag{9}$$

- If $n = 1$, then $\texttt{\#rel}(T_n) = 1$, that is consistent with Equation 9.

- If $n > 1$, let $m = (n-1)/2$. T_n is of the form $\ell(A \circ B)$, where A and B are two complete balanced trees of size m. By Lemma 14, we have

$$\texttt{\#rel}(T_n) \geq n - m + 2 \times \texttt{\#rel}(T_m).$$

By induction hypothesis for T_m, it follows that

$$\begin{aligned}
\texttt{\#rel}(T_n) &\geq n - m + (m+1)\log_2(m+1) \\
&\geq \frac{(n+1)(\log_2(m+1)+1)}{2} \\
&= \frac{(n+1)\log_2(2m+2)}{2} \\
&= \frac{(n+1)\log_2(n+1)}{2}
\end{aligned}$$

□

Corollary 16 *Let A and B be two trees of size n. For any decomposition strategy, $\texttt{\#rel}(A, B)$ has a lower bound in $O(n^2 \log^2(n))$.*

4 Cover Strategies

In this section, we define a main family of strategies that we call *cover strategies*. The idea comes from the following remark. Assume that $\ell(f) \circ f$ is a relevant forest for a given strategy. Suppose that the direction of $\ell(f) \circ t$ is left. This decomposition generates three relevant forests: $\ell(f)$, t and $f \circ t$. The forest f is a sub-forest of $f \circ t$. An opportune point of view is then to first eliminate nodes of f in $f \circ t$, so that $f \circ t$ and t share relevant forests as most as possible. We make this intuitive property more formal by defining *covers*. Covers are generalization of path decompositions.

Definition 17 (Cover) *Let F be a forest. A* cover *r of F is a mapping from F to $F \cup \{right, left\}$ satisfying for each node i in F*

- *if $deg(i) = 0$ or $deg(i) = 1$, then $r(i) \in \{right, left\}$,*
- *if $deg(i) > 1$, then $r(i)$ is a child of i.*

In the first case, $r(i)$ is called the direction *of i, and in the latter case, $r(i)$ is called the* favorite child *of i.*

Definition 18 (Cover Strategy) *Given a pair of trees (A, B) and a cover r for A, we associate a unique strategy ϕ as follows.*

- *if $deg(i) = 0$ or $deg(i) = 1$, then $\phi(A(i), G) = r(i)$, for each forest G of B.*
- *if $A(i)$ is of the form $\ell(A_1 \circ \cdots \circ A_n)$ with $n > 1$, then let $p \in \{1, \ldots, n\}$ such that the favorite child $r(i)$ is the root of A_p. For each forest G of B, we define*

$$\begin{aligned}
\phi(A(i), G) &= \text{right } \textit{whenever } p = 1, \text{ left } \textit{otherwise,} \\
\phi(T \circ A_p \circ \cdots \circ A_n, G) &= \text{left, } \textit{for each forest } T \textit{ of } A_1 \circ \cdots \circ A_{p-1}, \\
\phi(A_p \circ T, G) &= \text{right, } \textit{for each forest } T \textit{ of } A_{p+1} \circ \cdots \circ A_n.
\end{aligned}$$

The tree A is called the cover tree. *A strategy is a* cover strategy *if there exists a cover associated to it.*

The family of cover strategies includes Zhang-Shasha and Klein algorithms. The Zhang-Shasha algorithm corresponds to the cover

- for any node of degree 0 or 1, the direction is *right*,
- for any other node, the favorite child is the leftmost child.

The associated strategy involves only right decomposition rules, according to Equation 2. Klein's algorithm may be described as the cover strategy

- for any node of degree 0 or 1, the direction is *left*,
- for any other node, the favorite child is the root of the heaviest subtree.

The aim is now to study the number of relevant forests for a cover strategy. This task is divided into two steps. First, we compute the number of strategies for the cover tree alone. This intermediate result will play a great part in our final objective.

Lemma 19 *Let f be a forest, t a nonempty forest and ℓ a labeled node.*

- *Let ϕ be a cover strategy for $\ell(f) \circ t$ such that $\phi(\ell(f) \circ t) = left$, let $k = |f|$. We write $f_1, \ldots, f_k$ for denoting the k sub-forests of f corresponding to the successive left decompositions of f: f_1 is f, and each f_{i+1} is obtained from f_i by a left deletion.*

$$\mathsf{R}(\ell(f) \circ t) = \{\ell(f) \circ t, f_1 \circ t, \ldots, f_k \circ t\} \cup \mathsf{R}(\ell(f)) \cup \mathsf{R}(t).$$

- *Let ϕ be a cover strategy for $t \circ \ell(f)$ such that $\phi(t \circ \ell(f)) = right$, let $k = |f|$. We write $g_1, \ldots, g_k$ for denoting the k sub-forests of f corresponding to the successive right decompositions of f: g_1 is f, and each g_{i+1} is obtained from g_i by a right deletion.*

$$\mathsf{R}(t \circ \ell(f)) = \{t \circ \ell(f), t \circ g_1, \ldots, t \circ g_k\} \cup \mathsf{R}(\ell(f)) \cup \mathsf{R}(t).$$

PROOF. We show the first claim, the proof of the other one being symmetrical. By Lemma 11, we have

$$\mathsf{R}(\ell(f) \circ t) = \{\ell(f) \circ t\} \cup \mathsf{R}(\ell(f)) \cup \mathsf{R}(t) \cup \mathsf{R}(f \circ t) \tag{10}$$

Let's have a closer look at $f \circ t$. We establish that

$$\mathsf{R}(f \circ t) = \{f_1 \circ t\} \cup \cdots \cup \{f_k \circ t\} \cup_{i \in f} \mathsf{R}(f(i)) \cup \mathsf{R}(t) \tag{11}$$

The proof is by induction on k. If $k = 1$, then $f = f_1$ and $\mathsf{R}(f \circ t) = \{f_1 \circ t\} \cup \mathsf{R}(t)$, that is the expected result. If $k > 1$, let ℓ', f' and t' such that $f = \ell'(f') \circ t'$. We have

$$\mathsf{R}(f \circ t) = \{f \circ t\} \cup \mathsf{R}(\ell'(f')) \cup \mathsf{R}(f \circ t' \circ t) \cup \mathsf{R}(t' \circ t).$$

On the other hand $F = f_1$, $f' \circ t' = f_2$ and $t' = f_{|\ell'(f')|+1}$. Since ϕ is a cover strategy, the direction for $f \circ t$ and for successive sub-forests containing t is left. We apply the induction hypothesis to t' and $f' \circ t'$ and it concludes the proof of Equation 11.

We come back to Equation 10. With Equation 11, we get

$$\mathsf{R}(\ell(f) \circ t) = \{\ell(f) \circ t\} \cup \{f_1 \circ t\} \cup \cdots \cup \{f_k \circ t\} \cup_{i \in f} \mathsf{R}(f(i)) \cup \mathsf{R}(\ell(f)) \cup \mathsf{R}(t).$$

According to Lemma 13, for each node i in f, $\mathsf{R}(f(i))$ is included in $\mathsf{R}(\ell(f))$. It follows that

$$\mathsf{R}(\ell(f) \circ t) = \{\ell(f) \circ t\} \cup \{f_1 \circ t\} \cup \cdots \cup \{f_k \circ t\} \cup \mathsf{R}(\ell(f)) \cup \mathsf{R}(t).$$

□

The first consequence of this Lemma is that cover strategies can reach the $O(n \log(n))$ lower bound of Lemma 15. We give a criterion for that.

Lemma 20 *Let F be a forest provided with a cover strategy ϕ, satisfying the following property: for each relevant forest $A \circ t \circ B$ (A and B are trees, t is a forest)*

- *if $|B| > |A \circ t|$, then $\phi(A \circ t \circ B) =$ left, and*
- *if $|A| > |t \circ B|$, then $\phi(A \circ t \circ B) =$ right,*

then $\texttt{\#rel}(F) \leq |F| \log_2(|F| + 1)$.

PROOF. The proof is by induction on the size F.

- If F is empty, the result is direct.

- If F is a single tree: F is of the form $\ell(g)$, and $\texttt{\#rel}(F) = 1 + \texttt{\#rel}(g)$. By induction hypothesis for g, $\texttt{\#rel}(g) \leq |g| \log_2(|g|+1)$. Since $|F| = |g|+1$, it follows that $\texttt{\#rel}(F) \leq |F| \log_2(|F| + 1)$.

- If F is not a single tree, there are two nonempty trees A and B and a, possibly empty, forest t such that $F = A \circ t \circ B$. Assume $\phi(F) =$ *left* (the case $\phi(F) =$ *right* is identical). Let $n = |A|$ and $m = |t \circ B|$. By Lemma 19,

$$\texttt{\#rel}(F) = n + \texttt{\#rel}(A) + \texttt{\#rel}(t \circ B) \tag{12}$$

By induction hypothesis for A and $t \circ B$, it follows that $\texttt{\#rel}(F) \leq n + n \log_2(n + 1) + m \log_2(m + 1)$. Since $\phi(F) =$ left, the hypothesis of the Lemma implies that $n \leq m$ and so $\texttt{\#rel}(F) \leq (n + m) \log_2(n + m + 1)$. □

Another application of Lemma 19 is that it is possible to know the exact number of relevant forests for the cover tree.

Lemma 21 *Let $A = \ell(A_1 \circ \cdots \circ A_n)$ be a cover tree such that $n = 1$ or the root of A_j is the favorite child.*

$$\mathsf{R}(A) = \{A\} \cup \{f_1, \ldots, f_k\} \cup \{g_1, \ldots, g_h\} \cup_i \mathsf{R}(A_i)$$

where k is the size of $A_1 \circ \cdots \circ A_{j-1}$, h is the size of $A_{j+1} \circ \cdots \circ A_n$, f_1 is $A_1 \circ \cdots \circ A_n$, and each f_{i+1} is obtained from f_i by a left deletion, g_1 is $A_j \circ \cdots \circ A_n$ and each g_{i+1} is obtained from g_i by a right deletion.

PROOF. The proof is by induction on the size of A. If $|A| = 1$, then $\mathsf{R}(A)$ is $\{A\}$. If $|A| > 1$, by Lemma 11, $\mathsf{R}(A)$ is $\{A\} \cup \mathsf{R}(A_1 \circ \cdots \circ A_n)$. Iterated application of Lemma 19 yields the expected result. □

Lemma 22 *Let $A = \ell(A_1 \circ \cdots \circ A_n)$ be a cover tree such that $n = 1$ or the root of A_j is the favorite child.*

$$\texttt{\#rel}(A) = |A| - |A_j| + \texttt{\#rel}(A_1) + \cdots + \texttt{\#rel}(A_n).$$

PROOF. Direct consequence of Lemma 21. □

Remark 23 *As a corollary of Lemma 22 and Lemma 14, we know that Klein's algorithm is a strategy that minimizes the number of relevant forests for the cover tree, and Lemma 20 implies that the number of relevant forests for the cover tree is in $O(n \log(n))$. Together with Lemma 35, this gives a new, simpler proof of Proposition 8 concerning the class of complexity of Klein's algorithm.*

With the analysis of the number of relevant forests for the cover tree, we are now able to look for the total number of relevant forests for a pair of trees. Given a pair of trees (A, B) provided with a cover for A, it appears that all relevant forests of A fall within three categories:

- (α) those that are compared with all *rightmost* forests of B,
- (β) those that are compared with all *leftmost* forests of B,
- (γ) those that are compared with all *special* forests of B.

Definition 24 (Special forests) *Let F be a tree. The set $\mathsf{S}(F)$ of* special forests *of F is the set of sub-forests that can deduced from F by a series of right or left deletions.*

$$\mathsf{S}(\ell(f) \circ t \circ \ell'(g)) = \{(\ell(f) \circ t \circ \ell'(g)\}\mathsf{S}(f \circ t \circ \ell'(g)) \cup \mathsf{S}(\ell(f) \circ t \circ g)$$

We write $\texttt{\#spec}(A)$ for the number of special forests of A.

Lemma 25 *For any forest F, for any strategy ϕ, the set of relevant forests $\mathsf{R}(F)$ is included in the set of special forests of $\mathsf{S}(F)$.*

PROOF. By induction on $|F|$. □

The goal of the next lemmas is to find criteria to assign the proper category (α), (β) or (γ) to each relevant forest of A. We have this immediate result.

Lemma 26 *Let (A, B) be a pair of trees, A being a cover tree, and let f be a relevant forest of A.*

- *if the direction of f is left, then f is at least compared with all rightmost forests of B.*
- *if the direction of f is right, then f is at least compared with all leftmost forests of B,*

PROOF. Let i be the node of A such that $A(i)$ is the smallest tree containing f. Let g be a forest of B and let j be the node of B such that $B(j)$ is the smallest tree containing g. By Lemma 13, we know that $(A(i), B(j))$ is a relevant pair.

- For the first case, we prove that if g is a rightmost forest, then $(f, g) \in \mathsf{R}(A, B)$. By definition of cover strategies, f is a rightmost forest, since only rightmost forests can be assigned a left direction. This implies that the direction of i is left. The pair (f, g) can be deduced from $(A(i), B(j))$ by a series of right deletions on A and right insertions on B.

- For the latter case, we prove that if g is a leftmost forest, then $(f, g) \in \mathsf{R}(A, B)$. There are three cases. If f is $A(i)$, then (f, g) is deduced from $(f, B(j))$ by a series of right insertions on B. If f is a leftmost forest, by definition of cover strategies, it means that the favorite child of i is its leftmost child. It implies that the direction of i is right. So (f, g) can be deduced from $(A(i), B(j))$ by a series of right deletions on A a right insertions on B. If f is not a leftmost forest, then the first node of f is the favorite child of i and the direction of i is left. Let f' be the smallest rightmost forest containing f. The pair $(f', B(j))$ can be deduced from $(A(i), B(j))$ by a series of left deletions on A. The direction of f' is right, since f' begins with the favorite child, like f. (f, g) can be deduced from $(f', B(j))$ by a series of right deletions on A and right insertions on B. □

For subtrees whose roots are *free nodes*, the category is entirely determined by the direction.

Definition 27 (Free Node) *Let A be a cover tree. A node i is* free *if i is the root of A, or if its parent is of degree greater than one and i is not the favorite child.*

Lemma 28 *Let i be a free node of A.*

(1) if the direction of i is left, then $A(i)$ is (α),
(2) if the direction of i is right, then $A(i)$ is (β).

PROOF. We establish the first claim. Assume there exists a free node i with direction left, such that $A(i)$ is not (α). Applying Lemma 26, it means that $A(i)$ is compared with a forest that is not a rightmost forest. It is then possible to consider g, the largest forest of B such that $(A(i), g)$ belongs to $\mathsf{R}(A, B)$ and g is not a rightmost forest. g is not the whole tree B, since B is a particular rightmost forest. So there are four possibilities for the generation of $(A(i), g)$: $(A(i), g)$ is generated by an insertion, $(A(i), g)$ is generated by a deletion and $(A(i), g)$ is generated by a substitution, which gives two cases.

- If $(A(i), g)$ is generated by an insertion: since the direction of i is *left*, there exists a node ℓ and two forests h and p such that $g = h \circ p$ and $(A(i), \ell(h) \circ p)$ is in $\mathsf{R}(A, B)$. Since the size of $\ell(h) \circ p$ is greater than the size of g, $\ell(h) \circ p$ is a rightmost forest, that implies that $g = h \circ p$ is also a rightmost forest.

- If $(A(i), g)$ is generated by a deletion: there exists a node ℓ such that either $\ell \circ A(i)$, $A(i) \circ l$ or $\ell(A(i))$ is a relevant forest. In the two first cases, this would imply that i is a favorite child, and in the third case, that the degree of the parent of i is 1. In all cases, this contradicts the hypothesis that i is a free node.

- If $(A(i), g)$ is generated by a replacement, being the matching part of the replacement: this would imply that g is a tree, that contradicts the hypothesis that g is not a rightmost forest.

- If $(A(i), g)$ is generated by a replacement, being the remaining part of the replacement: $(A(i), g)$ should be obtained from a relevant pair of the form $(A' \circ A(i), B' \circ g)$ or $(A(i) \circ A', g \circ B')$, where A' and B' are subtrees of A and B respectively. In both cases, this contradicts the hypothesis that i is not a favorite child. □

For nodes that are not free and for forests, the situation is more complex. It is then necessary to take into account the category of the parent too. The two following lemmas establish that those nodes inherit the relevant forests of B from their parents.

Lemma 29 *Let F be a relevant forest of A that is not a tree. Let i be the lower common ancestor of the set of nodes of F and let j be the favorite child of i.*

(1) if F is a rightmost forest whose leftmost tree is not $A(j)$, then F has the same category as $A(i)$,
(2) if F is a leftmost forest, then F has the same category as $A(i)$,
(3) otherwise F is (γ).

PROOF. The proof of the two first cases is similar to the proof of Lemma 28. We give a detailed proof of the third case. By definition of cover strategies, the first tree of F is $A(j)$ and the direction of F is right. Assume there is a special forest g of B such that (F, g) is not a relevant pair. We suppose that g is of maximal size.

- If g is a rightmost forest. Since F is not a leftmost forest, j is not the leftmost child of i, that implies that the direction of $A(i)$ is *left*. Lemma 26 implies that $(A(i), g)$ is a relevant pair. The pair (F, g) is obtained from $(A(i), g)$ by successive left deletions until j and right deletions until F.

- If g is not a rightmost forest. There exists a node ℓ such that $g \circ \ell$ is a special forest. By construction of g, $(F, g \circ \ell)$ is a relevant pair. Since the direction of F is right, a right deletion gives (F, g). □

Lemma 30 *Let A be a cover tree, i be a node of A that is not free, and j be the parent of i.*

- *if the direction of i is* left*, if i is the rightmost child of j and $A(j)$ is (α), then $A(i)$ is (α),*
- *if the direction of i is* right*, if i is the leftmost child of j and $A(j)$ is (β), then $A(i)$ is (β),*
- *otherwise $A(i)$ is (γ).*

PROOF. Similar to proof of Lemma 29. □

As a corollary of Lemmas 28 and 30, it appears that the category (α), (β) and (γ) of a tree A rooted at i depends on

- the category of the parent j of i and the favorite child of j,
- the direction of i, that is associated to the favorite child of i.

It means that we have to look at the parent j of i and at the favorite child of i. Dealing with the favorite child of i can be captured by a bottom-up computation. For the parent j, we have to consider all cases regarding the category of j and the favorite child of j. Lemmas 28 and 30 enable us to reduce all possible cases to four possibilities.

Definition 31 *Let A be a tree, i be a node of A, and j be the parent of i (if i is not the root).*

$$
\begin{array}{rcl}
\mathit{Free}(A(i)) & : & \text{cardinality of } \mathsf{R}(A,B) \cap (A(i),B) \text{ if } i \text{ is free (Lemma 28),} \\
\mathit{Right}(A(i)) & : & \text{cardinality of } \mathsf{R}(A,B) \cap (A(i),B) \text{ if } A(j) \text{ is } (\alpha),\ i \text{ is the} \\
 & & \text{favorite child and the rightmost child of } j \text{ (Lemma 30-1),} \\
\mathit{Left}(A(i)) & : & \text{cardinality of } \mathsf{R}(A,B) \cap (A(i),B) \text{ if } A(j) \text{ is } (\beta),\ i \text{ is the} \\
 & & \text{favorite child and the leftmost child of } j \text{ (Lemma 30-2),} \\
\mathit{All}(A(i)) & : & \text{cardinality of } \mathsf{R}(A,B) \cap (A(i),B) \text{ otherwise (Lemma 30-3).}
\end{array}
$$

With this notation, $\texttt{\#rel}(A,B)$ equals $\mathit{Free}(A)$. We are now able to formulate the main result of this section, which gives the total number of relevant forests for a cover strategy.

Theorem 32 *Let (A,B) be a pair of trees, A being a cover tree.*

1. *If A is reduced to a single node whose direction is right*

$$
\begin{aligned}
\mathit{Free}(A) = \ \mathit{Left}(A) \ &= \texttt{\#left}(B) \\
\mathit{All}(A) \ = \mathit{Right}(A) &= \texttt{\#spec}(B)
\end{aligned}
$$

2. *If A is reduced to a single node whose direction is left*

$$
\begin{aligned}
\mathit{Free}(A) = \mathit{Right}(A) &= \texttt{\#right}(B) \\
\mathit{All}(A) \ = \ \mathit{Left}(A) \ &= \texttt{\#spec}(B)
\end{aligned}
$$

3. *If $A = \ell(A')$ and the direction of ℓ is right*

$$
\begin{aligned}
\mathit{Free}(A) = \ \mathit{Left}(A) \ &= \texttt{\#left}(B) + \mathit{Right}(A') \\
\mathit{All}(A) \ = \mathit{Right}(A) &= \texttt{\#spec}(B) + \mathit{All}(A')
\end{aligned}
$$

4. *If $A = \ell(A')$ and the direction of ℓ is left*

$$\begin{aligned} \mathit{Free}(A) = \mathit{Right}(A) &= \texttt{\#right}(B) + \mathit{Left}(A') \\ \mathit{All}(A) \;=\; \mathit{Left}(A) \;&= \texttt{\#spec}(B) + \mathit{All}(A') \end{aligned}$$

5. *If $A = \ell(A_1 \circ \cdots \circ A_n)$ and the favorite child is the leftmost child*

$$\begin{aligned} \mathit{Free}(A) = \;\mathit{Left}(A) \;&= \textstyle\sum_{i>1} \mathit{Free}(A_i) + \mathit{Left}(A_1) + \texttt{\#left}(B)(|A| - |A_1|) \\ \mathit{All}(A) \;= \mathit{Right}(A) &= \textstyle\sum_{i>1} \mathit{Free}(A_i) + \mathit{All}(A_1) + \texttt{\#spec}(B)(|A| - |A_1|) \end{aligned}$$

6. *If $A = \ell(A_1 \circ \cdots \circ A_n)$ and the favorite child is the rightmost child*

$$\begin{aligned} \mathit{Free}(A) = \mathit{Right}(A) &= \textstyle\sum_{i<n} \mathit{Free}(A_i) + \mathit{Right}(A_n) + \texttt{\#right}(B)(|A| - |A_n|) \\ \mathit{All}(A) \;=\; \mathit{Left}(A) \;&= \textstyle\sum_{i<n} \mathit{Free}(A_i) + \mathit{All}(A_n) + \texttt{\#spec}(B)(|A| - |A_n|) \end{aligned}$$

7. *If $A = \ell(A_1 \circ \cdots \circ A_n)$ and the favorite child is A_j with $1 < j < n$*

$$\begin{aligned} \mathit{Free}(A) &= \textstyle\sum_{i\neq j} \mathit{Free}(A_i) + \mathit{All}(A_j) + \texttt{\#right}(B)(1 + |A_1 \circ \cdots \circ A_{j-1}|) \\ &\quad + \texttt{\#spec}(B)|A_{j+1} \circ \cdots \circ A_n| \\ \mathit{Right}(A) &= \textstyle\sum_{i\neq j} \mathit{Free}(A_i) + \mathit{All}(A_j) + \texttt{\#right}(B)(1 + |A_1 \circ \cdots \circ A_{j-1}|) \\ &\quad + \texttt{\#spec}(B)|A_{j+1} \circ \cdots \circ A_n| \\ \mathit{Left}(A) &= \textstyle\sum_{i\neq j} \mathit{Free}(A_i) + \mathit{All}(A_j) + \texttt{\#spec}(B)(|A| - |A_j|) \\ \mathit{All}(A) &= \textstyle\sum_{i\neq j} \mathit{Free}(A_i) + \mathit{All}(A_j) + \texttt{\#spec}(B)(|A| - |A_j|) \end{aligned}$$

PROOF. The proof of the theorem is based on Lemmas 21, 28, 29 and 30. We give the detailed proof for cases 5 and 7, that are representative of the other cases. For case 5, by Lemma 21, we have

$$\mathsf{R}(A) = \{A\} \cup \{g_1, \ldots, g_h\} \cup_i \mathsf{R}(A_i)$$

where h is the size of $A_{j2} \circ \cdots \circ A_n$, g_1 is $A_1 \circ \cdots \circ A_n$ and each g_{i+1} is obtained from g_i by a right deletion. We classify each forest in (α), (β) and (γ). For $\mathit{Free}(A)$, we have

- by definition of a cover strategy, the direction of ℓ is right, and by hypothesis ℓ is a free node. It follows that A is (β), by Lemma 28-2,
- this implies that $g_1, \ldots, g_h$ are (β) by Lemma 29-2,
- since the root of A_1 is the favorite child, this also implies that the number of pairs of forests for $\mathsf{R}(A_1)$ is $\mathit{Left}(A_1)$, by Definition 31,
- for the other nodes, the number of pairs of forests for $\mathsf{R}(A_i)$, $i > 1$, is $\mathit{Free}(A_i)$, by Definition 31.

Hence

$$\begin{aligned} \mathit{Free}(A) &= (1 + h)\ \texttt{\#left}(B) + \mathit{Left}(A_1) + \mathit{Free}(A_2) + \cdots + \mathit{Free}(A_n) \\ &= (|A| - |A_1|)\ \texttt{\#left}(B) + \mathit{Left}(A_1) + \mathit{Free}(A_2) + \cdots + \mathit{Free}(A_n) \end{aligned}$$

For *Right*(A), we have

- A is (γ) by Lemma 30-3,
- $g_1, \ldots, g_h$ are (γ) by Lemma 29-2,
- the number of pair of forests for $\mathsf{R}(A_1)$ is $All(A_1)$, by Definition 31,
- for the other nodes, the number of pair of forests for $\mathsf{R}(A_i)$, $i > 1$, is $Free(A_i)$.

For *Left*(A), we have

- A is (β) by Lemma 30-2,
- $g_1, \ldots, g_h$ are (β) by Lemma 29-2,
- the number of pair of forests for $\mathsf{R}(A_1)$ is $All(A_1)$, by Definition 31,
- for the other nodes, the number of pair of forests for $\mathsf{R}(A_i)$, $i > 1$, is $Free(A_i)$.

For *All*(A), we have

- A is (γ) by Lemma 30-3,
- $g_1, \ldots, g_h$ are (γ) by Lemma 29-2,
- the number of pair of forests for $\mathsf{R}(A_1)$ is $All(A_1)$, by Definition 31,
- for the other nodes, the number of pair of forests for $\mathsf{R}(A_i)$, $i > 1$, is $Free(A_i)$.

For case 7, by Lemma 21, we have

$$\mathsf{R}(A) = \{A\} \cup \{f_1, \ldots, f_k\} \cup \{g_1, \ldots, g_h\} \cup_i \mathsf{R}(A_i)$$

where k is the size of $A_1 \circ \cdots \circ A_{j-1}$, h is the size of $A_{j+1} \circ \cdots \circ A_n$, f_1 is $A_1 \circ \cdots \circ A_n$, and each f_{i+1} is obtained from f_i by a left deletion, g_1 is $A_j \circ \cdots \circ A_n$ and each g_{i+1} is obtained from g_i by a right deletion. For *Free*(A), we have

- by definition of a cover strategy, the direction of ℓ is left, and by hypothesis ℓ is a free node. It follows that A is (α), by Lemma 28-1.
- $f_1, \ldots, f_k$ are (α), by Lemma 29-1
- $g_1, \ldots, g_h$ are (γ), by Lemma 29-3
- the number of pair of forests for $\mathsf{R}(A_j)$ is $All(A_j)$, since j is the favorite child and is neither the leftmost child, nor the rightmost child.
- for the other nodes, the number of pair of forests for $\mathsf{R}(A_i)$, $i \neq j$, is $Free(A_i)$.

Hence

$$\begin{aligned} Free(A) &= (1+k)\ \texttt{\#right}(B) + h\ \texttt{\#spec}(B) + All(A_j) + \textstyle\sum_{i \neq j} Free(A_i) \\ &= \texttt{\#right}(B)\ (1 + |A_1 \circ \cdots \circ A_{j-1}|) + \texttt{\#spec}(B)\ |A_{j+1} \circ \cdots \circ A_n| \\ &\quad + All(A_j) \textstyle\sum_{i \neq j} + Free(A_i) \end{aligned}$$

For *Right*(A), we have

- A is (α), by Lemma 30-1,
- $f_1, \ldots, f_k$ are (α), by Lemma 29-1,
- $g_1, \ldots, g_h$ are (γ), by Lemma 29-3,
- the number of pair of forests for $\mathsf{R}(A_j)$ is $All(A_j)$, since j is the favorite child and is neither the leftmost child, nor the rightmost child.

- for the other nodes,the number of pair of forests for $\mathsf{R}(A_i)$, $i \neq j$, is $\mathit{Free}(A_i)$.

For $\mathit{Left}(A)$, we have

- A is (γ), by Lemma 30-3,
- $f_1, \ldots, f_k$ are (γ), by Lemma 29-1,
- $g_1, \ldots, g_h$ are (γ), by Lemma 29-3,
- the number of pair of forests for $\mathsf{R}(A_j)$ is $\mathit{All}(A_j)$, since j is the favorite child and is neither the leftmost child, nor the rightmost child.
- for the other nodes, the number of pair of forests for $\mathsf{R}(A_i)$, $i \neq j$, is $\mathit{Free}(A_i)$.

For $\mathit{All}(A)$, we have

- A is (γ), by Lemma 30-3,
- $f_1, \ldots, f_k$ are (γ), by Lemma 29-1,
- $g_1, \ldots, g_h$ are (γ), by Lemma 29-3,
- the number of pair of forests for $\mathsf{R}(A_j)$ is $\mathit{All}(A_j)$, since j is the favorite child and is neither the leftmost child, nor the rightmost child.
- the number of pair of forests for $\mathsf{R}(A_i)$, $i \neq j$, is $\mathit{Free}(A_i)$.

□

Lemma 33 *For Zhang-Shasha's algorithm,* $\texttt{\#rel}(A, B) = \texttt{\#left}(A) \cdot \texttt{\#left}(B)$.

PROOF. In Theorem 32, it appears that the cases *1*, *3* and *5* are the only useful cases. Applying Lemma 22, it follows that $\texttt{\#rel}(A, B) = \texttt{\#left}(A) \cdot \texttt{\#left}(B)$ (by induction on the size of A). □

We get the symmetrical result for the symmetrical strategy: in this case, $\texttt{\#rel}(A, B) = \texttt{\#right}(A) \cdot \texttt{\#right}(B)$.

For Theorem 32 to be effective, it remains to evaluate the values of `#right`, `#left` and `#spec`.

Lemma 34 *Let A be a tree.*

$$\texttt{\#right}(A) = \sum(|A(i)|, i \in A) - \sum(|A(j)|, j \textit{ is a rightmost child})$$

$$\texttt{\#left}(A) = \sum(|A(i)|, i \in A) - \sum(|A(j)|, j \textit{ is a leftmost child})$$

PROOF. Since rightmost forests are special cases of relevant forests, Lemma 22 gives

$$\texttt{\#right}(A) = 1, \text{if t}|A| = 11,$$

$$\texttt{\#right}(\ell(A_1, \ldots, A_n)) = \textstyle\sum_i \texttt{\#right}(A_i) + |A| - |A_n|, \text{otherwise.}$$

Induction on the size of A concludes the proof. The result for $\texttt{\#left}(A)$ is identical. □

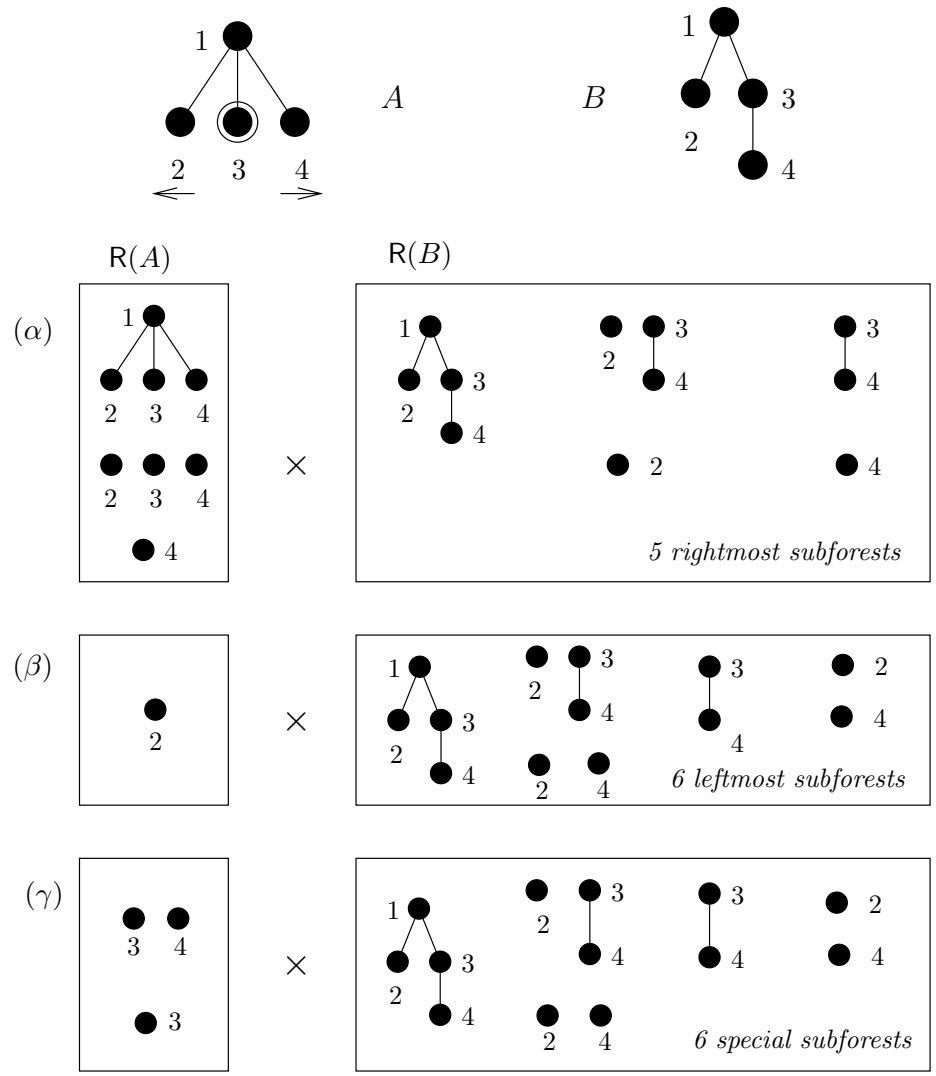


Fig. 5. **Set of all relevant forests for the pair of trees** (A, B)**.**
The tree A is the cover tree. The favorite child of 1 is 3, the direction of 2 is right, the direction of 4 is left. We display all relevant forests for A and B. The three blocks for A are the three categories (α), (β) and (γ). For each subgroup of A, we indicate the corresponding relevant forests of B: B $\texttt{\#right}(B) = 5$, $\texttt{\#left}(B) = 6$ (Lemma 34) and $\texttt{\#spec}(B) = 6$ (Lemma 35). This gives 33 pairs of relevant forests for the pair(A, B). This result is consistent with Theorem 32: the number of relevant forests is given by $\mathit{Free}(A)$, and applying case 7, we have

$$\begin{aligned}\mathit{Free}(A) &= \mathit{Free}(2) + \mathit{Free}(4) + \mathit{All}(3) + \texttt{\#spec}(B) \times 1 + \texttt{\#right}(B) \times 2\\ &= \texttt{\#left}(B) + \texttt{\#right}(B) + \texttt{\#spec}(B) + \texttt{\#spec}(B) + \texttt{\#right}(B) \times 2\\ &= 33\end{aligned}$$

Lemma 35 *Let F be a forest of size n.*

$$\texttt{\#spec}(F) = \frac{n(n+3)}{2} - \sum_{i \in F} |F(i)|.$$

PROOF. The proof is by induction on the size of n. If $n = 0$, then $\texttt{\#spec}(F) = 0$, which is consistent with the Lemma. If $n > 0$, then $F = \ell(g) \circ t$, where g and t are (possibly empty) sub-forests of F. There are two kinds of special sub-forests of F to be considered:

(1) those containing the node ℓ: there are $|t| + 1$ such sub-forests,
(2) those not containing the node ℓ: there are $\texttt{\#spec}(g \circ t)$ such sub-forests.

It follows that

$$\texttt{\#spec}(F) = |t| + 1 + \texttt{\#spec}(g \circ t).$$

On one hand $|t|+1 = n-|\ell(g)|+1$. On the other hand, the induction hypothesis applied to $g \circ t$, whose size is $n - 1$, ensures

$$\texttt{\#spec}(g \circ t) \;=\; \frac{(n-1)(n+2)}{2} - \sum_{i \in g \circ t} |g \circ t(i)|$$

Since $g \circ t$ is a sub-forest of F, this implies

$$\texttt{\#spec}(g \circ t) \;=\; \frac{(n-1)(n+2)}{2} - \sum_{i \in g \circ t} |F(i)|$$

It follows that

$$\begin{aligned}
\texttt{\#spec}(F) &= n - |\ell(g)| + 1 + \frac{(n-1)(n+2)}{2} - \textstyle\sum_{i \in g \circ t} |F(i)| \\
&= n + 1 + \frac{(n-1)(n+2)}{2} - \textstyle\sum_{i \in F} |F(i)| \\
&= \frac{n(n+3)}{2} - \textstyle\sum_{i \in F} |F(i)|
\end{aligned}$$

This concludes the proof. □

Figure 5 depicts an example of all relevant forests for a pair of trees provided with a cover strategy.

5 A new optimal cover strategy

In this last section, we show that Theorem 32 makes it possible to design an optimal cover strategy. An optimal cover strategy is a strategy that minimizes the total number of relevant forests. The algorithm is as follows. For any pair of trees (A, B), define four dynamic programming tables `Right`, `Left`, `Free`, and `All` indexed by nodes of A. The definition of `Right`, `Left`, `Free`, and `All` is directly borrowed from Theorem 32. The only difference is that the favorite child is not known. So at each step, the favorite child is chosen to be the child that minimizes the number of relevant forests. For instance, if

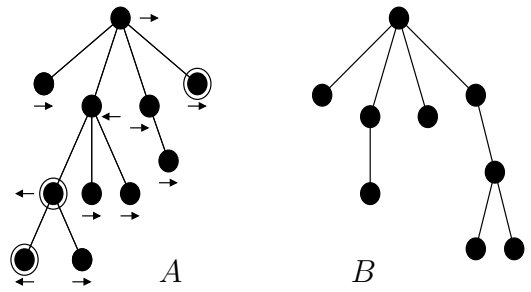


Fig. 6. This figure shows an optimal cover for A built up for the pair (A, B). For each node, the direction, left or right, is indicated with an arrow. For each node of degree greater than 1, the favorite child is the circled node. This cover yields 340 relevant forests. For the two Zhang-Shasha strategies, we get 405 and 350 relevant forests, and for the Klein strategy, we get 391 relevant forests.

$A = \ell(A_1 \circ \cdots \circ A_n)$ then

$$\texttt{Free}(A) = \sum_{i \geq 1} \texttt{Free}(A_i) + \min \begin{cases} \texttt{Left}(A_1) - \texttt{Free}(A_1) + \texttt{\#left}(B) \times (|A| - |A_1|) \\ \texttt{All}(A_j) - \texttt{Free}(A_j) + \texttt{\#spec}(B)|A_{j+1} \circ \cdots \circ A_n| \\ + \texttt{\#right}(B)(1 + |A_1 \circ \cdots \circ A_{j-1}|), \quad 1 < j < n \\ \texttt{Right}(A_n) - \texttt{Free}(A_n) + \texttt{\#right}(B)(|A| - |A_n|) \end{cases}$$

The favorite child is selected to be the root of the subtree A_j so that A_j gives the minimal value in the alternative. The optimal cover is then built up by tracing back from *Free*(A). Figure 6 shows an example of optimal cover.

Lemma 36 *The cost of this preprocessing is in* $O(|A| + |B|)$.

PROOF. First, you need to compute `#right`(B), `#left`(B) and `#spec`(B). This can be made in $O(|B|)$ using Lemmas 34 and 32. The size of each array `Right`, `Left`, `Free` and `All` is $|A|$. The time to fill up the cell of a node i is proportional to the degree of i. So the overall time computation for each array is in $O(\sum_{i \in A} deg(i))$, that is in $O(|A|)$. □

Once the cover is set up, it is possible to compute the distance. The algorithm is performed in a bottom-up fashion. As in [13], it is likely to distinguish between proper trees (Equation 1) and others forests (Equations 2 and 3). It gives rise to two data structures:

- a permanent array `Treedist` that stores the values for $d(i, j)$, where i is a node of A and j a node of B. The values are obtained from Equation 1.
- a temporary structure `Forestdist` that stores the values that are attached to the computation of the current cell (i, j) of `Treedist`. `Forestdist` is filled in using Equations 2, 3 and previously computed cells of `Treedist`.

At first glance, `Forestdist` could require cubic space, when the favorite child of i is neither the leftmost child, nor the rightmost child. But it can be made quadratic: let k be the favorite child of i, h the rightmost child of i in A, and g the rightmost child of j in B. For all nodes x of $B(j)$, compute each $d(k..h, x..g)$ separately with a succession of right decompositions, in accordance with the cover strategy, and store it. Then compute $d(i, j)$ with a succession of left decompositions.

The final and optional step is to recover the mapping corresponding to the distance, in order to know explicitly how to transform A into B. This can be achieved using the same space amount, at the price of additional computations. Starting from $d(A, B)$, fill up the associated `Forestdist` array, and then extract the set of matching nodes generated by replacement operations by tracing back. Then iterate the process to all pairs of matching subtrees.

6 Discussion

The goal of this paper is to enlighten and quantify the role of underlying decomposition strategies in tree edit algorithms. We have introduced the definition of *cover strategies*, that are natural and easy to handle extensions of Klein and Zhang-Shasha algorithms. In this framework, we define a novel algorithm that minimizes the number of distinct recursive calls. By construction, this algorithm involves less relevant forests than Zhang-Shasha and Klein strategies. So it is at most in $n^3 \log(n)$ in the worst cases, and in n^3 in average ([2]). But we do not claim that this is the best algorithm, since we do not take into account the cost of the preprocessing in comparison with the expected gain of number of relevant forests. This is still an open question.

It should be stressed too that our algorithm is optimal for cover strategies only, not for all decomposition strategies. Outside the framework of cover strategies, the number of relevant forests can formally be improved when considering the following trick : for each pair of subtrees, it is possible to choose the cover tree, and so the algorithm is an alternation of cover strategies each time you compute a new `Forestdist` array. Unfortunately the implementation becomes much more intricate, and the gain is poor.

Finally, in this paper, we have chosen to focus on the total number of relevant forests. It is possible to develop a analogous analysis for alternative criteria. Some examples are

- minimizing the total number of forests in the temporary `Forestdist` arrays
- minimizing the amount of space for `Forestdist` arrays, that is minimizing the size of the largest `forestdist` array.

The technical details are very close to those described here, and most of our Lemmas could be simply adapted to achieve this aim.

References

[1] S. Chawathe, Comparing hierarchical data in external memory *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases* (1999), p. 90-101

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

[3] D. Harel and R.E. Tarjan. Fats algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13(2):338-335, 184

[4] T. Jiang, L. Wang and K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Sciences* 143, (1995)

[5] P. Klein. Computing the edit-distance between unrooted ordered trees *Proceedings of 6th European Symposium on Algorithms* (1998), p. 91-102.

[6] Needleman S.B. and Wunsh C.D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48 (1970), p. 443-453

[7] SM Selkow. The tree-to-tree editing problem. *information Processing Letters*, 6 (1977), p. 184-186

[8] B. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons, *Comput. Appl. Biosciences*, Vol. 4-3 (1988), p. 387-393.

[9] K.C. Tai. The tre-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26 (1979) p. 422-433

[10] H. Touzet. Tree edit distance with gaps. *Information Processing Letters*, Vol 85-3 (2003), p. 123 - 129

[11] G. Valiente. *Algorithms on Trees and Graphs*, Springer Verlag, 2002

[12] R.A. Wagner and M.J. Fischer. The String-to-String Correction Problem. *Journal of the Association for Computing Machinery* Vol 21-1 (1974), p. 168 - 173

[13] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, Vol 18-6 (1989), p. 1245-1262.

[14] K. Zhang. Algorithms for the constrained editing problem between ordered labeled trees and related problems. *Pattern Recognition* 28 (1995), p. 463-474