algorithms

MDPI

Article

Nearest Embedded and Embedding Self-Nested Trees

Romain Azaïs

Laboratoire Reproduction et Développement des Plantes, Univ Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRA, Inria, F-69342 Lyon, France; romain.azais@inria.fr

Received: 25 June 2019; Accepted: 27 August 2019; Published: 29 August 2019

Abstract: Self-nested trees present a systematic form of redundancy in their subtrees and thus achieve optimal compression rates by directed acrylic graph (DAG) compression. A method for quantifying the degree of self-similarity of plants through self-nested trees was introduced by Godin and Ferraro in 2010. The procedure consists of computing a self-nested approximation, called the nearest embedding self-nested tree, that both embeds the plant and is the closest to it. In this paper, we propose a new algorithm that computes the nearest embedding self-nested tree with a smaller overall complexity, but also the nearest embedded self-nested tree. We show from simulations that the latter is mostly the closest to the initial data, which suggests that this better approximation should be used as a privileged measure of the degree of self-similarity of plants.

Keywords: unordered trees; self-nested trees; approximation of trees; structural self-similarity

1. Introduction

Trees form a wide family of combinatorial objects that offers many application fields, e.g., plant modeling and XML files analysis. Modern databases are huge and thus stored in compressed form. Compression methods take advantage of repeated substructures appearing in the tree. As explained in [1], one often considers the following two types of repeated substructures: subtree repeat (used in DAG compression [2–5]) and tree pattern repeat (exploited in tree grammars [6,7] and top tree compression [1]). We restrict ourselves to DAG compression of unordered rooted trees, which consists of building a Directed Acyclic Graph (DAG) that represents a tree without the redundancy of its identical subtrees (see Figure 1). Two different algorithms exist for computing the DAG reduction of a tree τ [5] (2.2 Computing Tree Reduction), which share the same time-complexity in $O(\#\mathcal{V}(\tau)^2 \times \mathcal{D}(\tau) \times \log(\mathcal{D}(\tau)))$ where $\mathcal{V}(\tau)$ denotes the set of vertices of τ and $\mathcal{D}(\tau)$ its outdegree.

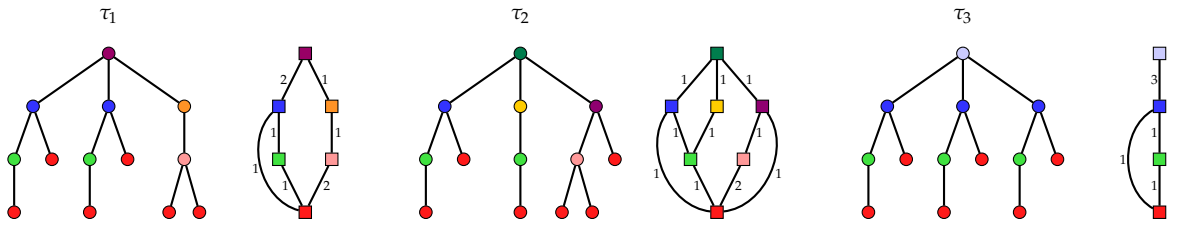

Figure 1. Trees and their DAG (Directed Acyclic Graph) reduction. In the tree, roots of isomorphic subtrees are colored identically. In the DAG, vertices are equivalence classes colored according to the class of isomorphic subtrees that they represent.

Trees that are the most compressed by DAG compression present the highest level of redundancy in their subtrees: all the subtrees of a given height are isomorphic. In this case, the DAG related to a tree τ is linear, i.e., there exists a path going through all vertices, with exactly $\mathcal{H}(\tau)+1$ vertices, $\mathcal{H}(\tau)$ denoting the height of τ, which is the minimal number of vertices among trees of this height (see τ_3 in

Figure 1). This family of trees has been introduced in [8] as the first interesting class of trees for which the subtree isomorphism problem is in NC^2. It has been known under the name of nested trees [8] and next self-nested trees [5] to insist on their recursive structure and their proximity to the notion of structural self-similarity.

The authors of [5] are interested in capturing the self-similarity of plants through self-nested trees. They propose to construct a self-nested tree that minimizes the distance of the original tree to the set of self-nested trees that embed the initial tree. The distance to this Nearest Embedding Self-nested Tree (NEST) is then used to quantify the self-nestedness of the tree and thus its structural self-similarity (see τ and NEST(τ) in Figure 2). The main result of [5] (Theorem 1 and E. NEST Algorithm) is an algorithm that computes the NEST of a tree τ from its DAG reduction in $O(\mathcal{H}(\tau)^2 \times \mathcal{D}(\tau))$.

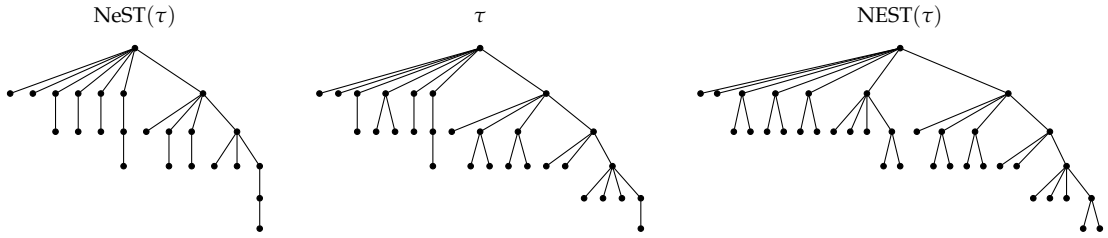

Figure 2. A tree τ (**middle**) with 30 nodes and its approximations NeST(τ) (**left**) with 24 nodes and NEST(τ) (**right**) with 37 nodes.

The goal of the present article is three-fold. We aim at proposing a new and more explicit algorithm that computes the NEST of a tree τ with the same time-complexity $O(\mathcal{H}(\tau)^2 \times \mathcal{D}(\tau))$ as in [5] but that takes as input the height profile of τ and not its DAG reduction. We establish that the height profile of a tree τ can be computed in $O(\#\mathcal{V}(\tau) \times \mathcal{D}(\tau))$ reducing the overall complexity of a linear factor. Based on this work, we also provide an algorithm in $O(\mathcal{H}(\tau)^2)$ that computes the Nearest embedded Self-nested Tree (NeST) of a tree τ (see τ and NeST(τ) in Figure 2). Finally, we show from numerical simulations that the distance of a tree τ to its NeST is much lower than the distance to its NEST. The NeST is most of the time a better approximation of a tree than the NEST and thus should be privileged to quantify the degree of self-nestedness of plants.

The paper is organized as follows. The structures of interest in this paper, namely unordered trees, DAG compression and self-nested trees, are defined in Section 2. Section 3 is dedicated to the definition and the study of the height profile of a tree. The approximation algorithms are presented in Section 4. We give a new insight on the definitions of the NEST and of the NeST in Section 4.1. Our NEST algorithm is presented in Section 4.2, while the NeST algorithm is given in Section 4.3. Section 5 is devoted to simulations. We state that the NeST is mostly a better approximation of a tree than the NEST in Section 5.1. An application to a real rice panicle is presented in Section 5.2. A summary of the paper and concluding remarks can be found in Section 6. All the figures and numerical experiments presented in the article have been made with the `Python` library `treex` [9].

2. Preliminaries

2.1. Unordered Rooted Trees

A rooted tree τ is a connected graph containing no cycle, i.e., without chain from any vertex v to itself, and such that there exists a unique vertex $\mathcal{R}(\tau)$, called the root, which has no parent, and any vertex different from the root has exactly one parent. The leaves of τ are all the vertices without children. The set of vertices of τ is denoted by $\mathcal{V}(\tau)$. The height of a vertex v may be recursively defined as $\mathcal{H}(v) = 0$ if v is a leaf of τ and

$$\mathcal{H}(v) = 1 + \max_{w \in \mathcal{C}_\tau(v)} \mathcal{H}(w)$$

otherwise, $\mathcal{C}_\tau(v)$ denoting the set of children of v in τ. The height of the tree τ is defined as the height of its root, $\mathcal{H}(\tau) = \mathcal{H}(\mathcal{R}(\tau))$. The outdegree $\mathcal{D}(\tau)$ of τ is the maximal branching factor that can be found in τ, i.e.,

$$\mathcal{D}(\tau) = \max_{v \in \tau} \#\mathcal{C}_\tau(v).$$

A subtree $\tau[v]$ rooted in v is a particular connected subgraph of τ. Precisely, $\tau[v] = (V[v], E[v])$ where $V[v]$ is the set of the descendants of v in τ and $E[v]$ is defined as

$$E[v] = \left\{(\xi, \xi') \in \mathcal{E}(\tau) \ : \ \xi \in V[v], \, \xi' \in V[v]\right\},$$

with $\mathcal{E}(\tau)$ the set of edges of τ.

In all the sequel, we consider unordered rooted trees for which the order among the sibling vertices of any vertex is not significant. A precise characterization is obtained from the additional definition of isomorphic trees. Let τ and θ two rooted trees. A one-to-one correspondence $\varphi : \mathcal{V}(\tau) \to \mathcal{V}(\theta)$ is called a tree isomorphism if, for any edge $(v, w) \in \mathcal{E}(\tau)$, $(\varphi(v), \varphi(w)) \in \mathcal{E}(\theta)$. Structures τ_1 and τ_2 are called isomorphic trees whenever there exists a tree isomorphism between them. One can determine if two n-vertex trees are isomorphic in $O(n)$ [10] (Example 3.2 and Theorem 3.3). The existence of a tree isomorphism defines an equivalence relation on the set of rooted trees. The class of unordered rooted trees is the set of equivalence classes for this relation, i.e., the quotient set of rooted trees by the existence of a tree isomorphism.

2.2. DAG Compression

Now we consider the equivalence relation "existence of a tree isomorphism" on the set of the subtrees of a tree τ. We consider the quotient graph $\mathcal{Q}(\tau) = (V, E)$ obtained from τ using this equivalence relation. V is the set of equivalence classes on the subtrees of τ, while E is a set of pairs of equivalence classes (C_1, C_2) such that $\mathcal{R}(C_2) \in \mathcal{C}_\tau(\mathcal{R}(C_1))$ up to an isomorphism. The graph $\mathcal{Q}(\tau)$ is a DAG [5] (Proposition 1) that is a connected directed graph without path from any vertex v to itself.

Let (C_1, C_2) be an edge of the DAG $\mathcal{Q}(\tau)$. We define $N(C_1, C_2)$ as the number of occurrences of a tree of C_2 just below the root of any tree of C_1. The tree reduction $\mathcal{R}(\tau)$ is defined as the quotient graph $\mathcal{Q}(\tau)$ augmented with labels $N(C_1, C_2)$ on its edges [5] (Definition 3 (Reduction of a tree)). Intuitively, the graph $\mathcal{R}(\tau)$ represents the original tree τ without its structural redundancies (see Figure 1).

2.3. Self-Nested Trees

A tree τ is called self-nested [5] (III. Self-nested trees) if for any pair of vertices v and w, either the subtrees $\tau[v]$ and $\tau[w]$ are isomorphic, or one is (isomorphic to) a subtree of the other. This characterization of self-nested trees is equivalent to the following statement: for any pair of vertices v and w such that $\mathcal{H}(v) = \mathcal{H}(w)$, $\tau[x] = \tau[y]$, i.e., all the subtrees of the same height are isomorphic.

Linear DAGs are DAGs containing at least one path that goes through all their vertices. They are closely connected with self-nested trees by virtue of the following result.

Proposition 1 (Godin and Ferraro [5])**.** *A tree τ is self-nested if and only if its reduction $\mathcal{R}(\tau)$ is a linear DAG.*

This result proves that self-nested trees achieve optimal compression rates among trees of the same height whatever their number of nodes (compare τ_3 with τ_1 and τ_2 in Figure 1). Indeed, $\mathcal{R}(\tau)$ has at least $\mathcal{H}(\tau) + 1$ nodes and the inequality is saturated if and only if τ is self-nested.

3. Height Profile of the Tree Structure

3.1. Definition and Complexity

This section is devoted to the definition of the height profile ρ_τ of a tree τ and to the presentation of an algorithm to calculate it. In the sequel, we assume that the tree τ is always traversed in the same order, depth-first search to set the ideas down. In particular, when vectors are indexed by nodes of τ sharing the same property, the order of the vector is important and should be always the same.

Given a vertex $v \in \mathcal{V}(\tau)$,

$$\gamma_h(v) = \#\left\{v' \in \mathcal{C}_\tau(v) \,:\, \mathcal{H}(\tau[v']) = h\right\}$$

is the number of subtrees of height h directly under v. Now, we consider the vector

$$\rho_\tau(h_1, h_2) = \left(\gamma_{h_2}(v) \,:\, v \in \mathcal{V}(\tau),\, \mathcal{H}(\tau[v]) = h_1\right)$$

made of the concatenation of the integers $\gamma_{h_2}(v)$ over subtrees $\tau[v]$ of height h_1 ordered in depth-first search. Consequently, ρ_τ is an array made of vectors with varying lengths.

Let A_1 and A_2 be two arrays for which each entry is a vector. We say that A_1 and A_2 are equivalent if, for any line i, there exists a permutation σ_i such that for any column j,

$$A_1(i,j) = \sigma_i(A_2(i,j)).$$

In particular, i being fixed, all the vectors $A_1(i,j)$ and $A_2(i,j)$ must have the same length. This condition defines an equivalence relation. The height profile of τ is the array ρ_τ as an element of the quotient space of arrays of vectors under this equivalence relation. In other words, the vectors $\rho_\tau(h_1, h_2)$, $0 \leq h_2 < h_1$ and h_1 fixed, must be ordered in the same way but the choice of the order is not significant. Finally, it should be already remarked that $\rho_\tau(h_1, h_2) = \emptyset$ when $h_2 \geq h_1$ or $h_1 > \mathcal{H}(\tau)$. Consequently, the height profile can be reduced to the triangular array

$$\rho_\tau = \left[\rho_\tau(h_1, h_2)\right]_{0 \leq h_2 < h_1 \leq \mathcal{H}(\tau)}.$$

The application ρ_τ provides the distribution of subtrees of height h_2 just below the root of subtrees of height h_1 for all couples (h_1, h_2), which typically represents the height profile of τ. For clarity's sake, we give the values of ρ_{τ_k} for the trees τ_k of Figure 1, coefficient (i,j) of the matrix being $\rho_{\tau_k}(i, j-1)$,

$$\rho_{\tau_1} = \rho_{\tau_2} = \begin{bmatrix} (1,1,2) & \emptyset & \emptyset \\ (0,1,1) & (1,1,1) & \emptyset \\ (0) & (0) & (3) \end{bmatrix} \quad \text{and} \quad \rho_{\tau_3} = \begin{bmatrix} (1,1,1) & \emptyset & \emptyset \\ (1,1,1) & (1,1,1) & \emptyset \\ (0) & (0) & (3) \end{bmatrix}. \tag{1}$$

It should be noticed that the height profile does not contain all the topology of the tree since trees τ_1 and τ_2 of Figure 1 are different but share the same height profile (1). However, the height of a tree τ can be recovered from its height profile through the relation $\mathcal{H}(\tau) = \dim(\rho_\tau)$, the dimension of ρ_τ being defined by

$$\dim(\rho_\tau) = \min\left\{n \geq 0 \,:\, \forall\, i \geq 0,\, \rho_\tau(n+1, i) = \emptyset\right\}.$$

Proposition 2. *ρ_τ can be computed in $O(\#\mathcal{V}(\tau) \times \mathcal{D}(\tau))$-time.*

Proof. First, attribute to each node $v \in \mathcal{V}(\tau)$ the height of the subtree $\tau[v]$ with complexity $O(\#\mathcal{V}(\tau))$. Next, traverse the tree in depth-first search in $O(\#\mathcal{V}(\tau))$ and calculate for each vertex v the vector $(\gamma_h(v))_{0 \leq h < \mathcal{H}(\tau[v])}$ in $\#\mathcal{C}_\tau(v) \leq \mathcal{D}(\tau)$ operations. Finally, append this vector to $\rho_\tau(\mathcal{H}(\tau[v]), \cdot)$ component by component. □

3.2. Relation with Self-Nested Trees

Self-nested trees are characterized by their height profile considering the following result.

Proposition 3. *τ is self-nested if and only if, for any $0 \leq h_2 < h_1 \leq \mathcal{H}(\tau)$, all the components of the vector $\rho_\tau(h_1, h_2)$ are the same (for instance see the profile (1) of the tree τ_3 presented in Figure 1). In addition, a self-nested tree τ can be reconstructed from ρ_τ (see Algorithm 1).*

Proof. If τ is self-nested, the N_{h_1} subtrees of height h_1 appearing in τ are isomorphic and thus have the same number n_{h_1,h_2} of subtrees of height h_2 just below their root. Consequently,

$$\rho_\tau(h_1, h_2) = (\underbrace{n_{h_1,h_2}, \ldots, n_{h_1,h_2}}_{N_{h_1}}).$$

The reciprocal result may be established considering the following lemma which proof presents no difficulty. □

Lemma 1. *If all the subtrees of height $0 \leq h < H$ appearing in a tree τ are isomorphic, and if all the subtrees of height H have the same number of subtrees of height $0 \leq h < H$ just below their root, then all the subtrees of height H appearing in τ are isomorphic.*

All the subtrees of height 1 in τ are isomorphic because all the components of $\rho_\tau(1, 0)$ are the same. The expected result is shown by induction on the height thanks to the previous lemma which assumptions are satisfied since ρ_τ always contains vectors for which all the entries are equal. The previous reasoning also provides a way (presented in Algorithm 1) to build a unique (self-nested) tree $\mathcal{T}$ from the height profile ρ_τ. In addition, this is easy to see that τ and $\mathcal{T}$ are isomorphic.

To present the algorithm of reconstruction of a self-nested tree from its height profile, we need to define the restriction of a height profile to some height. Let p be a height profile. The restriction $p_{|_h}$ of p to height $h \geq 0$ is the array defined by

$$\begin{cases} \forall\, 1 \leq h_1 \leq h, & \forall\, h_2 \geq 0, \quad p_{|_h}(h_1, h_2) = p(h_1, h_2), \\ \forall\, h_1 > h, & \forall\, h_2 \geq 0, \quad p_{|_h}(h_1, h_2) = \emptyset. \end{cases}$$

Consequently, $\dim(p_{|_h}) = \min(dim(p), h)$. A peculiar case is $p_{|_0}$ for which each entry is the empty set and thus $\dim(p_{|_0}) = 0$. It should be also remarked that there may exist no tree τ such that $p_{|_h}$ is the height profile of τ.

Algorithm 1: Construction of a self-nested tree from its height profile.

```
Function SN(p):
    Data: a height profile p such that all the components of p(h1, h2) are the same
    Result: the unique self-nested tree τ such that ρ_τ = p
    τ = •
    for i from 0 to dim(p) − 1 do
        add SN (p_|i) as child of R(τ) p(dim(p), i)_1 times
    return τ
```

As we can see in the proof of Proposition 3 or in Algorithm 1, the lengths of the vectors $\rho_\tau(h_1, h_2)$ are not significant to reconstruct a self-nested tree τ. Consequently, since all the components of $\rho_\tau(h_1, h_2)$ are the same, we can identify the height profile of a self-nested tree with the integer-valued array $[\rho_\tau(h_1, h_2)_1]$.

Proposition 4. *The number of nodes of a self-nested tree τ can be computed from ρ_τ in $O(\mathcal{H}(\tau)^2)$.*

Proof. By induction on the height, one has $\#\mathcal{V}(\tau) = \mathcal{N}(\mathcal{H}(\tau))$, where the sequence $\mathcal{N}$ is defined by $\mathcal{N}(0) = 1$ (number of nodes of a tree reduced to a root) and,

$$\forall 1 \leq H \leq \mathcal{H}(\tau),\ \mathcal{N}(H) = 1 + \sum_{h=0}^{H-1} \rho_\tau(H,h)\mathcal{N}(h). \tag{2}$$

The number of operations required to compute $\mathcal{N}(\mathcal{H}(\tau))$ is of order $O(\mathcal{H}(\tau)^2)$. □

The authors of [5] (Proposition 6) calculate the number of nodes of a tree (self-nested or not) from its DAG reduction by a formula very similar to (2), and which achieve the same complexity on self-nested trees. As mentioned before, a tree cannot be recovered from its height profile in general, thus we cannot expect such a result from the height profile of any tree.

4. Approximation Algorithms

4.1. Definitions

4.1.1. Editing Operations

We shall define the NEST and the NeST of a tree τ. As in [5] (Equation (5)), we ask these approximations to be consistent with Zhang's edit distance between unordered trees [11] denoted D_Z in this paper. Thus, as in [11] (2.2 Editing Operations), we consider the following two types of editing operations: adding a node and deleting a node. Deleting a node w means making the children of w become the children of the parent v of w and then removing w (see Figure 3). Adding w as a child of v will make w the parent of a subset of the current children of v (see Figure 4).

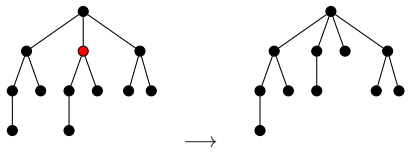

Figure 3. Deleting a node.

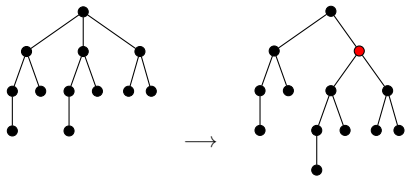

Figure 4. Inserting a node.

4.1.2. Constrained Editing Operations

Zhang's edit distance is defined from the above editing operations and from constrained mappings between trees [11] (3.1 Constrained Edit Distance Mappings). A constrained mapping between two trees τ and θ is a mapping [11] (2.3.2 Editing Distance Mappings), i.e., a one-to-one correspondence φ from a subset of $\mathcal{V}(\tau)$ into a subset of $\mathcal{V}(\theta)$ preserving the ancestor order, with an additional condition on the Least Common Ancestors (LCAs) [11] (condition (2) p. 208): if, for $1 \leq i \leq 3$, $v_i \in \mathcal{V}(\tau)$ and $w_i = \varphi(v_i) \in \mathcal{V}(\theta)$, then $\mathrm{LCA}(v_1, v_2)$ is a proper ancestor of v_3 if and only if $\mathrm{LCA}(w_1, w_2)$ is a proper ancestor of w_3.

Let θ be a tree that approximates τ obtained by inserting nodes in τ only and consider the induced mapping $M_{\tau\to\theta}$ that associates nodes of τ with themselves in θ. We want the approximation process

to be consistent with Zhang's edit distance D_Z, i.e., we want the mapping $M_{\tau\to\theta}$ to be a constrained mapping in the sense of Zhang, which in particular implies $D_Z(\theta,\tau) = \#\mathcal{V}(\theta) - \#\mathcal{V}(\tau)$. We shall prove that this requirement excludes some inserting operations in our context.

Indeed, the mapping $M_{\tau\to\theta}$ involved in the inserting operation of Figure 4 is partially displayed in Figure 5, nodes v_i of τ being associated with nodes w_i of θ. The LCA of v_1 and v_2 in τ is a proper ancestor of v_3. However, the LCA of w_1 and w_2 in θ is not a proper ancestor of w_3. Consequently, this mapping is not a constrained mapping as defined by Zhang. A necessary and sufficient condition for $M_{\tau\to\theta}$ to be a constrained mapping is given in Lemma 2.

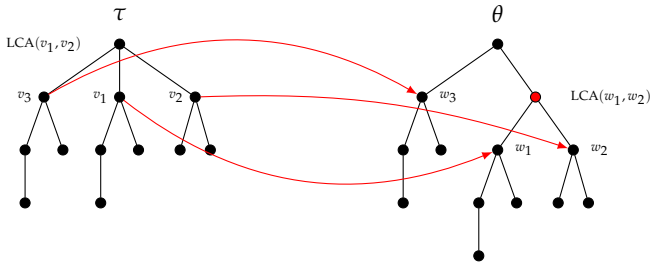

Figure 5. The tree θ is obtained from τ by inserting an internal node. The associated mapping does not satisfy the conditions imposed by Zhang [11] because the LCA (Least Common Ancestor) of v_1 and v_2 is a proper ancestor of v_3 whereas the LCA of w_1 and w_2 is not a proper ancestor of w_3.

Lemma 2. *Let τ be a tree and $v \in \mathcal{V}(\tau)$. Let θ be the tree obtained from τ by adding a node w as a child of v making the nodes of the subset $C \subset \mathcal{C}_\tau(v)$ children of w. The mapping $M_{\tau\to\theta}$ induced by these inserting operations is a constrained mapping in the sense of Zhang if and only if $C = \emptyset$, $\#C = 1$ or $\#C = \#\mathcal{C}_\tau(v)$.*

Proof. The proof is obvious if v has one or two children. Thus, we assume that v has at least three children c_1, c_2 and c_3. In τ, the LCA of c_1 and c_2 is v and v is an ancestor of c_3. Adding w as the parent of c_1 and c_2 makes it the LCA of these two nodes, but not an ancestor of c_3 in θ. The additional condition on the LCAs is then not satisfied. This problem appears only when making w the parent of at least two children and of not all the children of v. □

Consequently, we restrict ourselves to the following inserting operations which are the only ones that ensure that the associated mapping satisfies Zhang's condition: adding w as a child of v will make w (i) a leaf, (ii) the parent of one current child of v, or (iii) the parent of all the current children of v. However, it should be noticed that (iii) can always be expressed as (ii) (see Figure 6). Finally, we only consider the inserting operations that make the new child of v the parent of zero or one current child of v. For obvious reasons of symmetry, the allowed deleting operations are the complement of inserting operations, i.e., one can delete an internal node if and only if it has a unique child, which also ensures that the induced mapping is constrained in the sense of Zhang.

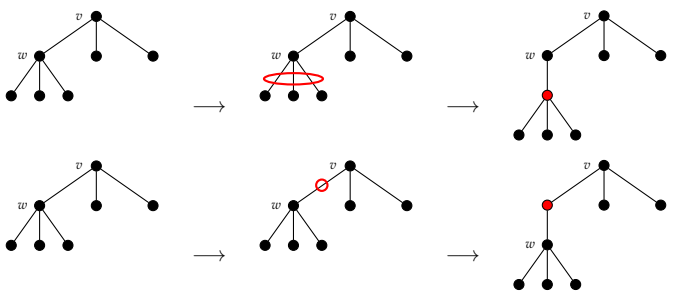

Figure 6. Adding a node as new child of w making all the current children of w children of this new node (**top**) provides the same topology as adding a new node between v and its child w (**bottom**).

4.1.3. Preserving the Height of the Pre-Existing Nodes

In [5] (Definition 9 and Figure 6), the NEST of a tree τ is obtained by successive partial linearizations of the (non-linear) DAG of τ which consist of merging all the nodes at the same height of the DAG. A consequence is that the height of any pre-existing node of τ is not changed by the inserting operations. For the sake of consistency with [5], we only consider inserting and deleting operations that preserve the height of all the pre-existing nodes of τ.

The next two results deal with inserting operations that preserve the height of the pre-existing nodes.

Lemma 3. *Let τ be a tree, $v \in \mathcal{V}(\tau)$ and $c \in \mathcal{C}_\tau(v)$. Let θ be the tree obtained from τ by adding the internal node w as a child of v making w the parent of c. Then,*

$$\forall\, u \in \mathcal{V}(\tau),\ \mathcal{H}(\theta[u]) = \mathcal{H}(\tau[u]) \iff \mathcal{H}(\tau[c]) + 1 < \mathcal{H}(\tau[v]).$$

Proof. Adding w may only increase the height of v and the one of its ancestors in τ. If the height of v is not changed by adding w, the height of its ancestors will not be modified. The height of v remains unchanged if and only if the height of w in θ, i.e., $\mathcal{H}(\tau[c]) + 1$, is strictly less than the height of $\tau[v]$. □

Lemma 4. *Let τ be a tree and $v \in \mathcal{V}(\tau)$. Let θ be the tree obtained from τ by adding a tree t as a child of v. Then,*

$$\forall\, u \in \mathcal{V}(\tau),\ \mathcal{H}(\theta[u]) = \mathcal{H}(\tau[u]) \iff \mathcal{H}(t) + 1 \leq \mathcal{H}(\tau[v]).$$

Proof. Adding a subtree t under v may only increase the height of v and the one of its ancestors in τ. If the height of v is not changed by adding t, the height of its ancestors will not be modified. Adding t will make the height of v increase if $\mathcal{H}(t)$ is strictly greater than the height of the higher child of v. □

A particular case of Lemma 4 is the insertion of leaves in a tree. Considering the above result, a leaf can be added under v if and only if $\mathcal{H}(\tau[v]) \geq 1$, i.e., v is not a leaf. The below results concern deleting operations that preserve the height of the remaining nodes of τ.

Lemma 5. *Let τ be a tree, $v \in \mathcal{V}(\tau)$, $w \in \mathcal{C}_\tau(v)$ and $\mathcal{C}_\tau(w) = \{c\}$. Let θ be the tree obtained from τ by deleting the internal node w making its unique child c a child of v. Then,*

$$\forall\, u \in \mathcal{V}(\theta),\ \mathcal{H}(\theta[u]) = \mathcal{H}(\tau[u]) \iff \exists\, w' \in \mathcal{C}_\tau(v) \setminus \{w\},\ \mathcal{H}(\tau[w']) + 1 = \mathcal{H}(\tau[v]).$$

Proof. Deleting w may only decrease the height of v and the one of its ancestors in τ. If the height of v is not changed by deleting w, the height of its ancestors will not be modified. The height of v remains unchanged if and only if it has a child different of w of height $\mathcal{H}(\tau[v]) - 1$. □

Lemma 6. *Let τ be a tree, $v \in \mathcal{V}(\tau)$, $c \in \mathcal{C}_\tau(v)$. Let θ be the tree obtained from τ by deleting the subtree $\tau[c]$. Then,*

$$\forall\, u \in \mathcal{V}(\theta),\ \mathcal{H}(\theta[u]) = \mathcal{H}(\tau[u]) \iff \exists\, c' \in \mathcal{C}_\tau(v) \setminus \{c\},\ \mathcal{H}(\tau[c']) + 1 = \mathcal{H}(\tau[v]).$$

Proof. The proof follows the same reasoning as in the previous result. □

4.1.4. NEST and NeST

In view of the foregoing, we consider the set of inserting and deleting operations that fulfill the below requirements.

Adding operations (see Figure 7)

- Internal nodes (AI): adding w as a child of v making w the parent of the child c of v can be done only if $\mathcal{H}(\tau[c]) + 1 < \mathcal{H}(\tau[v])$.
- Subtrees (AS): adding t as a child of v can be done only if $\mathcal{H}(t) + 1 \leq \mathcal{H}(\tau[v])$.

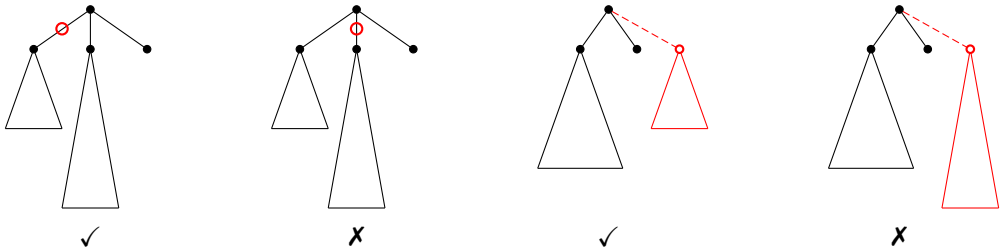

Figure 7. Allowed (✓) and forbidden (✗) inserting operations to construct the NEST of a tree.

Deleting operations (see Figure 8)

- Internal nodes (DI): deleting $v \in \mathcal{C}_\tau(u)$ (making the unique child w of v a child of u) can be done only if there exists $v' \in \mathcal{C}_\tau(u)$, $v \neq v'$, such that $\mathcal{H}(\tau[v']) \geq \mathcal{H}(\tau[v])$.
- Subtrees (DS): deleting the subtree $\tau[w]$, $w \in \mathcal{C}_\tau(v)$, of τ can be done if there exists $w' \in \mathcal{C}_\tau(v)$, $w' \neq w$, such that $\mathcal{H}(\tau[w']) + 1 = \mathcal{H}(\tau[v])$.

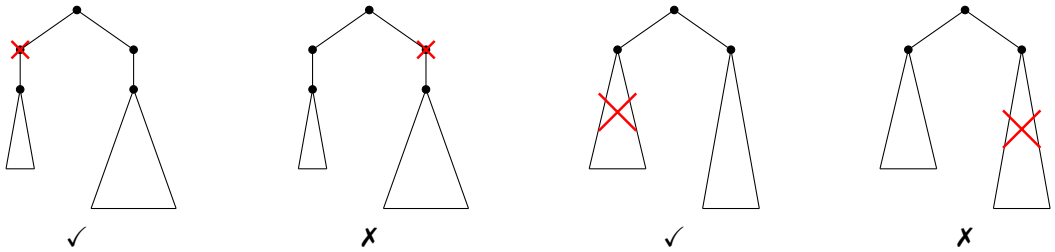

Figure 8. Allowed (✓) and forbidden (✗) deleting operations to construct the NeST of a tree.

Proposition 5. *The editing operations AI and AS (DI and DS, respectively) are the only inserting (deleting, respectively) operations that ensure that (i) the induced mapping is a constrained mapping and that (ii) the height of all the pre-existing nodes is unchanged.*

Proof. This result is a direct corollary of Lemmas 2–6. □

The NEST (the NeST, respectively) of a tree τ is the self-nested tree obtained by the set of inserting operations AI and AS (of deleting operations DI and DS, respectively) of minimal cost, the cost of inserting a subtree being its number of nodes. Existence and uniqueness of the NEST are not obvious at this stage. The NeST exists because the (self-nested) tree composed of a unique root can be easily obtained by deleting operations from any tree, but its uniqueness is not evident.

4.2. NEST Algorithm

To present our NEST algorithm in a concise form in Algorithm 2, we need to define the following operations involving two vectors u and v of the same size n and a real number γ,

$$\begin{cases} u + v = (u_1 + v_1, \ldots, u_n + v_n), \\ u + \gamma = (u_1 + \gamma, \ldots, u_n + \gamma), \\ u \vee \gamma = (\max(u_1, \gamma), \ldots, \max(u_n, \gamma)). \end{cases}$$

In other words, these operations must be understood component by component. In addition, in a condition, $u = 0$ ($u \neq 0$, respectively) means that for all $1 \leq i \leq n$, $u_i = 0$ ($u_i \neq 0$, respectively). Finally, for $1 \leq i \leq j \leq n$, $u_{i\ldots j}$ denotes the vector $(u_i, \ldots, u_j)$ of length $j - i + 1$. This notation will also

be used in Algorithm 3 for calculating the NeST. It should be noticed that an illustrative example that can help the reader to follow the progress of the algorithm is provided in Section 6.

Algorithm 2: Construction of the nearest embedding self-nested tree.

```
Function NEST(τ):
    Data: the height profile ρ of an unordered tree τ
    Result: the nearest embedding self-nested tree of τ
    for h_1 from 1 to H(τ) do
        for h_2 from h_1 − 1 to 0 do
            Δ ← max ρ_{h_1,h_2} − ρ_{h_1,h_2}
            ρ_{h_1,h_2} ← max ρ_{h_1,h_2}
            i ← 1
            while Δ ≠ 0 and i ≤ h_2 do
                Δ , ρ_{h_1,h_2−i} ← (Δ − ρ_{h_1,h_2−i}) ∨ 0 , ρ_{h_1,h_2−i} − Δ
                i ← i + 1
    return SN(ρ)
```

The relation between the above algorithm and the NEST of a tree is provided in the following result, which states in particular the existence of the NEST.

Proposition 6. *For any tree τ, Algorithm 2 returns the unique NEST of τ in $O(\mathcal{H}(\tau)^2 \times \mathcal{D}(\tau))$.*

Proof. By definition of the NEST, the height of all the pre-existing nodes of τ cannot be modified. Thus, the number of nodes of height $h-1$ under a node of height h can only increase by inserting subtrees in the structure. Then we have

$$\rho_{\mathrm{NEST}(\tau)}(h, h-1) \geq \max \rho_\tau(h, h-1). \tag{3}$$

Let v be a vertex of height h in τ. We recall that $\gamma_i(v)$ denotes the number of subtrees of height i under v. Our objective is to understand the consequences for $\gamma_i(v)$ of inserting operations to obtain $\rho_{\mathrm{NEST}(\tau)}(h, h-1)$ subtrees of height $h-1$ under v. To this aim, we shall define a sequence $\gamma_i^{(h-1,j)}(v)$ starting from $\gamma_i^{(h-1,0)}(v) = \gamma_i(v)$ that corresponds to the modified versions of τ. The first exponent $h-1$ means that this sequence concerns editing operations used to get the good number of subtrees of height $h-1$ under v. □

Let $\Delta_{h-1}^{(0)}(v) = \rho_{\mathrm{NEST}(\tau)}(h, h-1) - \gamma_{h-1}^{(0)}(v)$ be the number of subtrees of height $h-1$ that must be added under v to obtain the height profile of the NEST under v, i.e.,

$$\gamma_{h-1}^{(h-1,1)}(v) = \rho_{\mathrm{NEST}(\tau)}(h, h-1).$$

Implicitly, it means that $\gamma_i^{(h-1,1)}(v) = \gamma_i(0)(v)$ for $i \neq h-1$. The subtrees of height $h-1$ that we must add are isomorphic, self-nested and embed all the subtrees of height $h-2$ appearing in τ by definition of the NEST. In particular, they can be obtained by the allowed inserting operations from the subtrees of height $h-2$ under v, by first adding an internal node to increase their height to $h-1$. In addition, it is less costly in terms of editing operations to construct the subtrees of height $h-1$ from the subtrees of height $h-2$ available under v than to directly add these subtrees under v. If all the subtrees of height $h-2$ under v must be reconstructed later, it will be possible to insert them and the total cost will be same as by directly adding the subtrees of height $h-1$ under v. Consequently, all the available subtrees of height $h-2$ are used to construct subtrees of height $h-1$ under v and it remains

$$\Delta_{h-1}^{(1)} = \left(\Delta_{h-1}^{(0)}(v) - \gamma_{h-2}^{(h-1,1)}\right) \vee 0$$

subtrees of height $h-1$ to be built under v. Furthermore, in the new version of τ, we have

$$\gamma_{h-2}^{(h-1,2)}(v) = \gamma_{h-2}^{(h-1,1)}(v) - \Delta_{h-1}^{(1)}(v).$$

The $\Delta_{h-1}^{(1)}$ subtrees of height $h-1$ can be constructed from subtrees of height $h-3$ (with a larger cost than from subtrees of height $h-2$), and so on. To this aim, we define the sequence of the modified versions of τ by, for $0 \leq j \leq h-2$,

$$\begin{cases} \Delta_{h-1}^{(j+1)}(v) & = & \left(\Delta_{h-1}^{(j)}(v) - \gamma_{h-1-(j+1)}^{(h-1,j+1)}(v)\right) \vee 0, \\ \gamma_{h-(j+2)}^{(h-1,j+2)}(v) & = & \gamma_{h-(j+2)}^{(h-1,j+1)}(v) - \Delta_{h-1}^{(j+1)}(v). \end{cases}$$

At the final step $j = h-2$, the $\Delta_{h-1}^{(0)}(v)$ subtrees of height $h-1$ have been constructed from all the available subtrees appearing under v, starting from subtrees of height $h-2$, then $h-3$, etc., and then have been added if necessary.

From now on, the number of subtrees of height $h-2$ under v will not decrease. Indeed, it would mean that an internal node has been added between v and the root of a subtree of height $h-2$. This would have the consequence to increase of one unit the number of subtrees of height $h-1$ in subtrees of height h, which cost is (strictly) larger than adding a subtree of height $h-2$ in all the subtrees of height h. Consequently, we obtain

$$\rho_{\mathrm{NEST}(\tau)}(h, h-2) \geq \max_{\{v \in \mathcal{V}(\tau) \,:\, \mathcal{H}(\tau[v])=h\}} \gamma_{h-2}^{(h-1,h)}(v).$$

We can reproduce the above reasoning to construct under v subtrees of height $h-i$, i from 2 to $h-1$, from subtrees with a smaller height, which defines a sequence $\gamma_i^{(h-i,j)}$ of modified versions of τ, which size is $h-i+1$, and we get the following inequality,

$$\forall\, 2 \leq i \leq h,\ \rho_{\mathrm{NEST}(\tau)}(h, h-i) \geq \max_{\{v \in \mathcal{V}(\tau) \,:\, \mathcal{H}(\tau[v])=h\}} \gamma_{h-i}^{(h-i+1,h-i+2)}(v). \tag{4}$$

The tree returned by Algorithm 2 is self-nested and its height profile saturates the inequalities (3) and (4) for all the possible values of h and i by construction. In addition, we have shown that this tree can be obtained from τ by the allowed inserting operations. Since increasing of one unit the height profile at (h_1, h_2) has a (strictly) positive cost, this tree is thus the (unique) NEST of τ. As seen previously, the number of iterations of the while loop at line 7 is the number of subtrees of height $h_2 < h_1$ available to construct a tree of height h_1, i.e., the degree of τ in the worst case, which states the complexity.

4.3. NeST Algorithm

This section is devoted to the presentation of the calculation of the NeST in Algorithm 3. An illustrative example that can help the reader to follow the progress of the algorithm is provided in Section 6.

Algorithm 3: Construction of the nearest embedded self-nested tree.

Function `NeST(`τ`)`**:**
  Data: the height profile ρ of an unordered tree τ
  Result: the nearest embedded self-nested tree of τ
  for h_1 **from** 1 **to** $\mathcal{H}(\tau)$ **do**
    for h_2 **from** $h_1 - 1$ **to** 0 **do**
      $\Delta \leftarrow \rho_{h_1,h_2} - \min \rho_{h_1,h_2}$
      $\rho_{h_1,h_2} \leftarrow \min \rho_{h_1,h_2}$
      if $\rho_{h_1-1,0\dots h_1-3} = 0$ **and** $\rho_{h_1-1,h_1-2} = 1$ **then**
        $\rho_{h_1,h_2-1} \leftarrow \rho_{h_1,h_2-1} + \Delta$
  return `SN(`ρ`)`

Proposition 7. *For any tree τ, Algorithm 3 returns the unique NeST of τ in $O(\mathcal{H}(\tau)^2)$.*

Proof. The proof follows the same reasoning as the proof of Proposition 6. First, one may remark that

$$\rho_{\mathrm{NeST}(\tau)}(h, h-1) \leq \min \rho_\tau(h, h-1), \tag{5}$$

because the number of subtrees of height $h-1$ under a node v of height h can only decrease by the allowed deleting operations. Let v be a node of height h in τ and $\gamma_i(v)$ the number of subtrees of height i under v. If a subtree of height $h-i$ under v that must be deleted is not self-nested, one can first modify it to get a self-nested tree and then remove it with the same overall cost. Thus, we can assume without loss of generality that all the subtrees under v are self-nested. $\Delta_{h-1}(v) = \gamma_{h-1}(v) - \rho_{\mathrm{NeST}(\tau)}(h, h-1)$ denotes the number of subtrees of height $h-1$ that have to be removed from v. Let $\gamma_i^{(j)}(v)$ the sequence of the modifications to obtain $\rho_{\mathrm{NeST}(\tau)}(h, h-1)$ subtrees of height $h-1$ under v, with $\gamma_i^{(0)}(v) = \gamma_i(v)$. Instead of deleting a subtree of height $h-1$, it is always less costly to decrease its height of one unit by deleting its root. However it is possible only if this internal node has only one child, i.e., if $\rho_\tau(h-1, h-2) = 1$ and $\rho_\tau(h-1, i) = 0$ for $0 \leq i < h-2$. If this new tree of height $h-2$ must be deleted in the sequel, it will be done with the same global cost as by directly deleting the subtree of height $h-1$. Consequently,

$$\begin{cases} \gamma_{h-1}^{(1)}(v) &= \rho_{\mathrm{NeST}(\tau)}(h, h-1), \\ \gamma_{h-2}^{(1)}(v) &= \gamma_{h-2}^{(0)}(v) + \Delta_{h-1}(v)\mathbb{I}_{\{\rho_\tau(h-1,h-2)=1,\, \forall\, 3\leq i\leq h,\, \rho_\tau(h-1,h-i)=0\}}. \end{cases}$$

From now on, the number of subtrees of height $h-2$ under v will thus not increase and we obtain

$$\rho_{\mathrm{NeST}(\tau)}(h, h-2) \leq \min_{\{v\in\mathcal{V}(\tau)\,:\,\mathcal{H}(\tau[v])=h\}} \gamma_{h-2}^{(1)}(v).$$

There are $\Delta_{h-2}(v) = \gamma_{h-2}^{(1)}(v) - \rho_{\mathrm{NeST}(\tau)}(h, h-2)$ subtrees of height $h-2$ to be deleted under v. We can repeat the previous reasoning and delete the root of subtrees of height $h-2$ if possible rather than delete the whole structure, and so on for any height. Thus, the sequence $\gamma_i^{(j)}$ is defined from

$$\begin{cases} \Delta_{h-1-i}(v) &= \gamma_{h-1-i}^{(i)}(v) - \rho_{\mathrm{NeST}(\tau)}(h, h-1-i), \\ \gamma_{h-1-i}^{(i+1)}(v) &= \rho_{\mathrm{NeST}(\tau)}(h, h-1-i), \\ \gamma_{h-2-i}^{(i+1)}(v) &= \gamma_{h-2-i}^{(i)}(v) + \Delta_{h-1-i}(v)\mathbb{I}_{\{\rho_\tau(h-1,h-2)=1,\, \forall\, i+2\leq j\leq h,\, \rho_\tau(h-i,h-j)=0\}}, \end{cases}$$

and we have

$$\forall\, 0 \leq i \leq h-2,\ \rho_{\mathrm{NeST}(\tau}(h, h-2-i) \leq \min_{\{v \in \mathcal{V}(\tau) : \mathcal{H}(\tau[v])=h\}} \gamma_{h-2-i}^{(i+1)}(v). \tag{6}$$

The tree returned by Algorithm 3 saturates the inequalities (5) and (6) for all the possible values of h and i. Decreasing of one unit the height profile at (h_1, h_2) has a (strictly) positive cost. Thus, this tree is the (unique) NeST of τ. The time-complexity is given by the size of the height profile array. □

5. Numerical Illustration

5.1. Random Trees

The aim of this section is to illustrate the behavior of the NEST and of the NeST on a set of simulated random trees regarding both the quality of the approximation and the computation time. We have simulated 3000 random trees of size 10, 20, 30, 40, 50, 75, 100, 150, 200, and 250. For each tree, we have calculated the NEST and the NeST. The number of nodes of these approximations is displayed in Figure 9. We can observe that the number of nodes of the NEST is very large in regards with the size of the initial tree: approximately one thousand nodes on average for a tree of 150 nodes, which is to say an approximation error of 750 vertices. Remarkably, the NEST has never been a better approximation than the NeST on the set of simulated trees.

The computation time required to compute the NEST or the NeST of one tree on a 2.8 GHz Intel Core i7 has also been estimated on the set of simulated trees and is presented in Figure 10. As predicted by the theoretical complexities given in Propositions 6 and 7, the NeST algorithm requires less computation time than the NEST. Consequently, the NeST provides a much better and faster approximation of the initial data than the NEST.

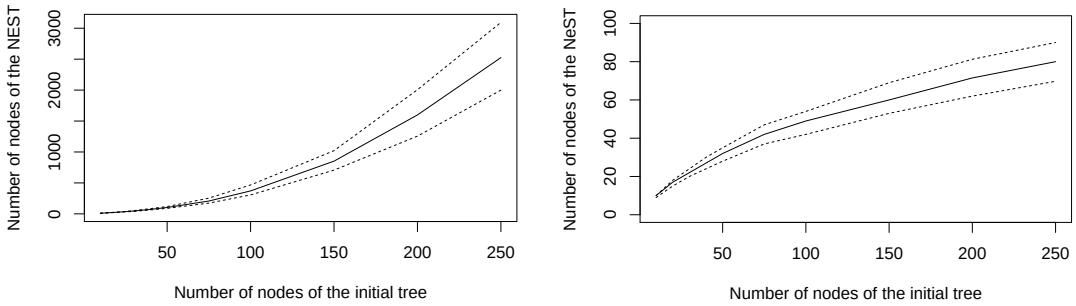

Figure 9. Number of nodes of the NEST (**left**) and of the NeST (**right**) estimated from 3000 random trees: average (full lines) and first and third quartiles (dashed lines).

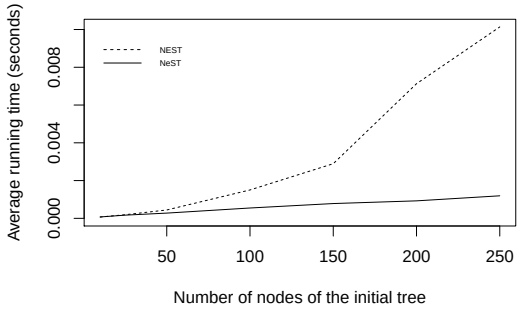

Figure 10. Average running time required to compute the NEST (dashed line) or the NeST (full line) estimated from 3000 simulated trees.

5.2. Structural Analysis of a Rice Panicle

Considering [5], we propose to quantify the degree of self-nestedness of a tree τ by the following indicator based on the calculation of $\mathrm{NEST}(\tau)$,

$$\delta_{\mathrm{NEST}}(\tau) = 1 - \frac{D_Z(\mathrm{NEST}(\tau), \tau)}{\#\mathcal{V}(\tau)} = \frac{2\#\mathcal{V}(\tau) - \#\mathcal{V}(\mathrm{NEST}(\tau))}{\#\mathcal{V}(\tau)}, \tag{7}$$

where D_Z stands for Zhang's edit distance [11]. In [5] (Equation (6)), the degree of self-nestedness of a plant is defined as in (7) but normalizing by the number of nodes of the NEST and not the size of the initial data, which avoids the indicator to be negative. In the present paper, we prefer normalizing by the number of nodes of τ to obtain the following comparable self-nestedness measure based on the calculation of $\mathrm{NeST}(\tau)$,

$$\delta_{\mathrm{NeST}}(\tau) = 1 - \frac{D_Z(\mathrm{NeST}(\tau), \tau)}{\#\mathcal{V}(\tau)} = \frac{\#\mathcal{V}(\mathrm{NeST}(\tau))}{\#\mathcal{V}(\tau)}.$$

The main advantage of this normalization is that if the NEST and the NeST offer equally good approximations, i.e., $D_Z(\mathrm{NEST}(\tau), \tau) = D_Z(\mathrm{NeST}(\tau), \tau)$, then the degree of self-nestedness does not depend on the chosen approximation scheme, $\delta_{\mathrm{NEST}}(\tau) = \delta_{\mathrm{NeST}}(\tau)$.

We propose to investigate the degree of structural self-similarity of the topological structure of the rice panicle studied in [5] (4.2 Analysis of a Real Plant) through these self-nested approximations. The rice panicle V_1 is made of a main axis bearing a main inflorescence P_1 and lateral systems V_i, $2 \leq i \leq 5$, each composed of inflorescences P_j, $2 \leq j \leq 8$ (see Figure 11). We have computed the indicators of self-nestedness $\delta_{\mathrm{NEST}} \vee 0$ and δ_{NeST} for each substructure composing the whole panicle (see Figure 12). The numerical values and the shape of these indicators are similar. However, δ_{NeST} is always greater than δ_{NEST}, in particular for the largest structures V_i. Based on a better approximation procedure as highlighted in the previous section, the NeST better captures the self-nestedness of the rice panicle.

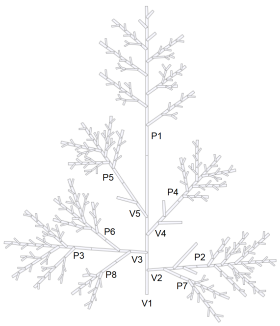

Figure 11. The rice panicle is composed of a main axis and lateral systems V_i, each made of one or several inflorescences P_j.

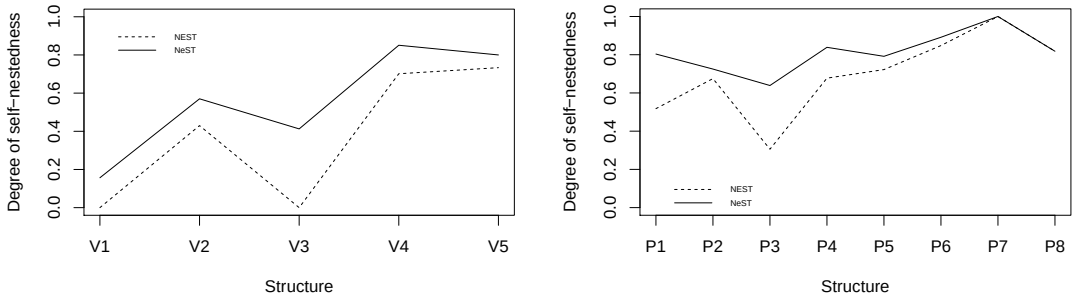

Figure 12. Degree of self-nestedness measured by $\delta_{\mathrm{NEST}} \vee 0$ (dashed lines) and δ_{NeST} (full lines) of the different substructures appearing in the rice panicle.

6. Summary and Concluding Remarks

Self-nested trees are unordered rooted trees that are the most compressed by DAG compression. Since DAG compression takes advantage of subtree repetitions, they present the highest level of redundancy in their subtrees. In this paper, we have developed a new algorithm for computing the Nearest Embedding Self-nested Tree (NEST) of a tree τ in $O(\mathcal{H}(\tau)^2 \times \mathcal{D}(\tau))$, as well as the first algorithm for determining its Nearest embedded Self-nested Tree (NeST) with time-complexity $O(\mathcal{H}(\tau)^2)$.

To this end, we have introduced the notion of height profile of a tree. Roughly speaking, the height profile is a triangular array which component (h_1, h_2), with $h_2 < h_1$, is the list of the numbers of direct subtrees of height h_2 in subtrees of height h_1, where a subtree is said direct if it is attached to the root. We have shown in Proposition 3 that self-nested trees are characterized by their height profile. While the first NEST algorithm [5] was based on edition of the DAG related to the tree to be compressed, the two approximation algorithms developed in the present paper take as input the height profile of any tree τ, which can be computed in $O(\#\mathcal{V}(\tau) \times \mathcal{D}(\tau))$-time (see Proposition 2), and modify it from top to bottom and from right to left, to return the self-nested height profile of the expected estimate (see Algorithms 2 and 3). Figures 13 and 14 illustrate the progress of the algorithms on a simple example. They should be examined in relation to the corresponding algorithms. We would like to emphasize that our paper also states the uniqueness of the NEST and of the NeST, and studies the link with edit operations admitted in Zhang's distance.

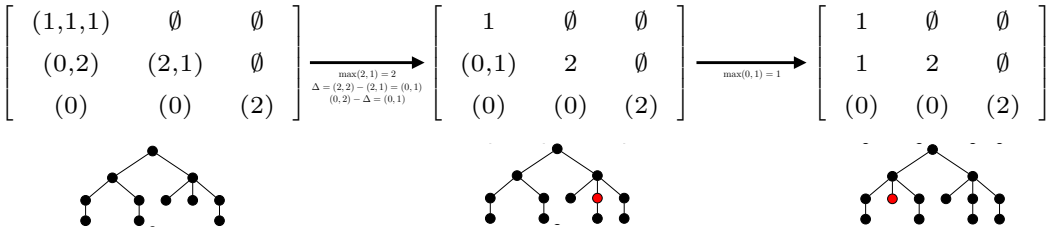

Figure 13. Progress of Algorithm 2 to compute the NEST of the left tree from its height profile. Only the second line must be edited to get the correct output. Editions of the height profile are associated with addition of vertices in red. The output tree is self-nested and has been constructed by adding a minimal number of nodes to the initial tree.

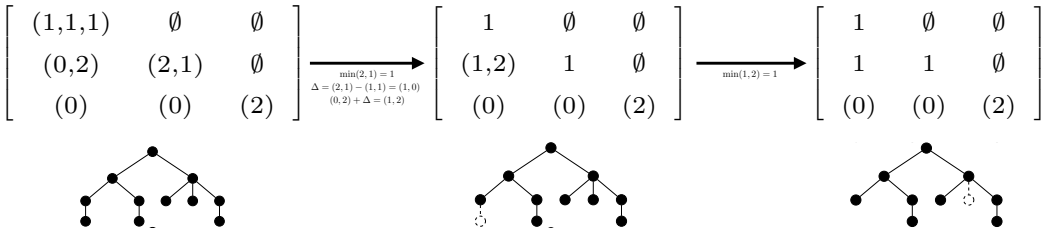

Figure 14. Progress of Algorithm 3 to compute the NeST of the left tree from its height profile. Only the second line must be edited to get the correct output. Editions of the height profile are associated with deletion of vertices in dashed lines. The output tree is self-nested and has been constructed by removing a minimal number of nodes from the initial tree.

Remarkably, estimations performed on a dataset of random trees establish that the NeST is a more accurate approximation of the initial tree than the NEST. This observation could be investigated from a theoretical perspective. In addition, we have shown that the NeST better captures the degree of structural self-similarity of a rice panicle than the NEST.

The algorithms developed in this paper are available in the last version of the `Python` library `treex` [9].

Funding: This research received no external funding.

Acknowledgments: The author would like to show his gratitude to two anonymous reviewers for their relevant comments on a first version of the manuscript.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Bille, P.; Gørtz, I.L.; Landau, G.M.; Weimann, O. Tree compression with top trees. *Inf. Comput.* **2015**, *243*, 166–177. [CrossRef]
2. Bousquet-Mélou, M.; Lohrey, M.; Maneth, S.; Noeth, E. XML Compression via Directed Acyclic Graphs. *Theory Comput. Syst.* **2014**, *57*, 1322–1371. [CrossRef]
3. Buneman, P.; Grohe, M.; Koch, C. Path Queries on Compressed XML. In Proceedings of the 29th International Conference on Very Large Data Bases, VLDB'03, Berlin, Germany, 9–12 September 2003; Volume 29, pp. 141–152.
4. Frick, M.; Grohe, M.; Koch, C. Query evaluation on compressed trees. In Proceedings of the 18th Annual IEEE Symposium of Logic in Computer Science, Ottawa, ON, Canada, 22–25 June 2003; pp. 188–197.
5. Godin, C.; Ferraro, P. Quantifying the degree of self-nestedness of trees. Application to the structural analysis of plants. *IEEE Trans. Comput. Biol. Bioinform.* **2010**, *7*, 688–703. [CrossRef] [PubMed]
6. Busatto, G.; Lohrey, M.; Maneth, S. Efficient Memory Representation of XML Document Trees. *Inf. Syst.* **2008**, *33*, 456–474. [CrossRef]
7. Lohrey, M.; Maneth, S. The Complexity of Tree Automata and XPath on Grammar-compressed Trees. *Theor. Comput. Sci.* **2006**, *363*, 196–210. [CrossRef]
8. Greenlaw, R. Subtree Isomorphism is in DLOG for Nested Trees. *Int. J. Found. Comput. Sci.* **1996**, *7*, 161–167. [CrossRef]
9. Azaïs, R.; Cerutti, G.; Gemmerlé, D.; Ingels, F. treex: A Python package for manipulating rooted trees. *J. Open Source Softw.* **2019**, *4*, 1351. [CrossRef]
10. Aho, A.V.; Hopcroft, J.E.; Ullman, J.D. *The Design and Analysis of Computer Algorithms*, 1st ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1974.
11. Zhang, K. A constrained edit distance between unordered labeled trees. *Algorithmica* **1996**, *15*, 205–222. [CrossRef]