



HAL
open science

Lossy compression of trees via linear directed acyclic graphs

Romain Azaïs, Jean-Baptiste Durand, Christophe Godin

► **To cite this version:**

Romain Azaïs, Jean-Baptiste Durand, Christophe Godin. Lossy compression of trees via linear directed acyclic graphs. 2016. hal-01294013v1

HAL Id: hal-01294013

<https://hal.science/hal-01294013v1>

Preprint submitted on 25 Mar 2016 (v1), last revised 26 Oct 2018 (v3)

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

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

Lossy compression of trees via linear directed acyclic graphs

Romain Azaïs*, Jean-Baptiste Durand†, and Christophe Godin‡

*Inria project-team BIGS
Institut Élie Cartan de Lorraine
Université de Lorraine
F-54 506 Vandœuvre-lès-Nancy, France
romain.azais@inria.fr

†Université Grenoble Alpes, Lab. J. Kuntzmann
Inria project-team MISTIS
51, Rue des Mathématiques
B.P. 53, F-38 041 Grenoble Cedex 9, France
jean-baptiste.durand@imag.fr

‡Inria project-team Virtual Plants
joint with CIRAD and INRA
Université Montpellier 2, Bâtiment 5, CC 06002
860, Rue de St Priest
34 095 Montpellier Cedex 5, France
christophe.godin@inria.fr

Abstract

A classical compression method for trees is to represent them by directed acyclic graphs. This approach exploits subtree repeats in the structure and is efficient only for trees with a high level of redundancy. The class of self-nested trees presents remarkable compression properties because of the systematic repetition of subtrees. In this paper, we provide a better understanding of this specific family of trees and we introduce a lossy compression method that consists in computing the reduction of a self-nested structure that closely approximates the initial data. We compare two versions of our algorithm and a competitive approach of the literature on a simulated dataset.

Keywords: Unordered trees, Self-nested trees, Directed acyclic graphs, Lossy compression

1 Introduction

Trees are commonly used to represent hierarchical data appearing in computer science as XML files, suffix trees, dictionaries, or language expression structures. Compression methods often take advantage of repeated substructures appearing in the tree. As it is explained in [4], one often considers the following two types of repeated substructures: subtree repeat and tree pattern repeat.

- *Subtree repeat.* A subtree (sometimes referred to as complete subtree in the literature) is a subgraph of a tree consisting of a vertex and all its descendants. A subtree repeat is an isomorphic occurrence of this pattern in the tree. DAG compression [5, 6, 13, 14] uses this type of repeated substructures.
- *Tree pattern repeat.* A tree pattern is any connected subgraph of a tree. A tree pattern repeat is an identical occurrence of a tree pattern appearing in the tree. Tree grammars [7, 17, 18] and top tree compression [4] use this type of repeated substructures.

A survey on this topic may be found in [19] in the context of XML files.

In this paper, we restrict ourselves to DAG compression, which consists in building a Directed Acyclic Graph (DAG) that represents a tree without the redundancy of its identical subtrees. Previous algorithms have been proposed to allow the computation of the DAG of an ordered tree with complexities ranging in $O(n^2)$ to $O(n)$ [10], where n is the number of vertices of the tree. In the case of unordered trees, two

different algorithms exist [14, 2.2 Computing Tree Reduction], which share the same time-complexity in $O(n^2 \times m \times \log(m))$, where n is the number of vertices of the tree and m denotes its outdegree. From now on, we limit ourselves to unordered trees and focus on the question of the compression rate achieved by DAG compression scheme. Of course, this is a complex problem that does not admit a simple answer because it depends on both the considered scenario and the chosen criterion. Here, we consider a compression rate that takes into account both the numbers of vertices and edges of the structures: ρ is defined as 1 minus the ratio of the number of vertices and edges of the DAG over the number of vertices and edges of the initial tree. Nevertheless, one may find in [5, Theorems 29 and 30] first theoretical elements to address this difficult question: the average numbers of vertices \bar{N}_n and of edges \bar{E}_n of the DAG related to a tree randomly chosen with uniform distribution among unlabeled unordered trees with n vertices behave as

$$\bar{N}_n = \sqrt{\frac{\ln(4)}{\pi}} \frac{n}{\sqrt{\ln(n)}} \left(1 + O\left(\frac{1}{\ln(n)}\right)\right) \quad \text{and} \quad \bar{E}_n = 3\sqrt{\frac{\ln(4)}{\pi}} \frac{n}{\sqrt{\ln(n)}} \left(1 + O\left(\frac{1}{\ln(n)}\right)\right).$$

As a consequence, the average compression rate $\bar{\rho}_n = 1 - (\bar{N}_n + \bar{E}_n)/(2n - 1)$ is of order

$$\bar{\rho}_n = 1 - 2\sqrt{\frac{\ln(4)}{\pi \ln(n)}} \left(1 + O\left(\frac{1}{\ln(n)}\right)\right),$$

which may be deemed to converge insufficiently fast to 1: for instance, one has $\bar{\rho}_{100} \simeq 38\%$ and $\bar{\rho}_{1000} \simeq 49\%$. In addition, these average results do not take into account the high dispersion of the potential compression rates among unordered trees. We shall present two opposite examples. The linear tree is defined as follows: each vertex has exactly one child, except the final leaf. This structure is not compressed at all by DAG procedure, that is to say $\rho = 0$. It should be remarked that the number of vertices of the DAG of a tree of height h is at least $h + 1$. In other words, the DAG only provides a width compression of the initial tree. Interestingly, a complementary strategy has been developed in [2] to compute height compression of a DAG in polynomial-time. Actually, only trees with a high level of redundancy in their subtrees are efficiently compressed by their DAG version. A good example of this phenomenon is provided by the topological structure of some plants. The authors of [14] model plants by tree graphs and propose in particular to compress a rice panicle with 843 vertices (see [14, Figure 19]): they obtain a complex DAG (see [14, Figure 20]) with 106 vertices and 162 edges that achieves a good compression rate $\rho \simeq 84\%$.

Trees that are the most compressed by DAG compression scheme present the highest level of redundancy in their subtrees: all the subtrees of a given height must be isomorphic. In this case, regardless of the number of vertices, the DAG related to a tree T has exactly $h + 1$ vertices, where h denotes the height of T , which is the minimal number of vertices that may be reached. This family of trees has been introduced in [14, Definition 7] under the name of self-nested trees. According to the previous remarks, a quite natural lossy compression method could be to approximate a tree by a self-nested structure that is next highly compressed by DAG method. If the self-nested approximation accurately represents the data to compress, one should obtain a lossy compression algorithm with high compression rates and low error rates. This idea has been developed in [14]: motivated by biological considerations, the authors propose to add a minimum number of vertices to a tree for obtaining a self-nested structure (called Nearest Embedding Self-nested Tree, NEST) that is next compressed. NEST estimate of a tree T may be computed in $O(h^2 \times m)$, where h denotes the height of T and m its outdegree.

In this article, we build on previous work and provide a better understanding of the class of self-nested trees that has never been studied in the literature. First of all, we investigate the combinatorics of this family under some natural conditions on the height and the outdegree (see Section 3). We then introduce a distance on the space of unordered trees in order to quantify the quality of a given self-nested estimate of a tree structure. We define in Subsection 4.1 an edit distance δ from the edit operations consisting in leaf insertion and deletion, with the same unit cost. We show that the computation of δ reduces to a minimum cost flow problem (see Subsection 4.3) which time-complexity is polynomial (see Proposition 10). We identify the least self-nested structure with respect to this edit distance, i.e., the structure that is the farthest to a

self-nested tree (see Proposition 12). These results show that self-nested trees are very rare, while still being relatively close to any unordered tree. Furthermore, we state through two examples that some quantities may be computed on self-nested trees faster than on general unordered trees (see Lemma 2: computation of the number of vertices of a tree and Proposition 11: computation of the edit distance). All these comments establish that self-nested trees provide a very simple structure that is well-adapted to approximate unordered trees.

The second aim of this paper is to propose an alternative solution to NEST approximation of a tree. Only adding vertices may appear restrictive in particular without a specific motivation coming for example from biology as in [14]. Thus one may expect better self-nested estimates of a tree than NEST solution. In this context, the best idea would be to compute a projection on self-nested trees of the tree structure that we want to compress. This question belongs to the class of NNS (Nearest Neighbor Search) problems, called post office problems in [15], referring to the question of assigning to a residence the nearest post office. More precisely, the problem on which we focus is an NNS in a non-ordered discrete data space. Limited work has been reported in the literature for these very complex queries [16]. The discrete state space considered in this work consists of unordered trees, which makes it necessary to use the adapted tree edit distance δ . Only exhaustive search would allow for computing the nearest self-nested tree of a given data. Here, we introduce two algorithms called RFC (Replace Forests by their Centroid) and RFC⁺ (RFC improved by local pruning) that provide accurate self-nested estimates of a tree (see Algorithms 2 and 4). Their time-complexity is investigated in Propositions 15 and 16, while their efficiency – in particular compared to NEST algorithm – is illustrated on simulated datasets in Section 6.

The paper is organized as follows. Section 2 is devoted to the presentation of the concepts of interest in this paper, namely unordered trees, self-nested trees and tree reduction. Combinatorics of self-nested trees is presented in Section 3. Section 4 deals with the edit distance δ on the space of unordered trees used in this article. Our approximation algorithms are presented in Section 5, while Section 6 gathers the numerical results. Finally, most of the long proofs have been deferred to Appendices A and B.

2 Preliminaries

This section is devoted to the precise formulation of the structures of interest in this paper, among which the class of non-plane rooted trees \mathbb{T} , the set of self-nested trees \mathbb{T}^{sn} and the concept of tree reduction.

2.1 Towards tree reduction

Directed graphs. A *finite directed graph* or *graph* is a pair $G = (V, E)$ where V denotes the finite set of *vertices*, and E denotes a finite set of ordered pairs of vertices called *edges*. If (x, y) is an edge of the graph G , x is a *parent* of y and y is a *child* of x . In all the sequel, $\text{child}(x)$ denotes the set of children of vertex x . A *path* from a vertex x to a vertex y is a sequence of edges $(\xi_k, \xi_{k+1})_{1 \leq k \leq M-1}$ such that $\xi_1 = x$ and $\xi_M = y$. x is called an *ancestor* of y if $x = y$ or if there exists a path from x to y , and y is then called a *descendant* of x . The ancestors of y that are different from y are referred to as *proper ancestors*.

Connected directed graphs. A *chain* from a vertex x to a vertex y is a sequence of vertex pairs $\{\xi_k, \xi_{k+1}\}_{1 \leq k \leq M-1}$ such that $\xi_1 = x$, $\xi_M = y$ and for any k , either (ξ_k, ξ_{k+1}) or (ξ_{k+1}, ξ_k) is an edge. Two vertices x and y are *connected* if there exists a chain from x to y . A graph G is *connected* if any pair of vertices are connected.

Rooted trees. τ is a *rooted tree* if τ is a connected graph containing no cycle, that is, without chain from any vertex x to itself, and such that there exists a unique vertex $\text{root}(\tau)$, 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. Their set is denoted $\text{leaves}(\tau)$. The *height* of a vertex x may be recursively defined as $\text{height}(x) = 0$ if x is a leaf of τ and $\text{height}(x) = 1 + \max_{y \in \text{child}(x)} \text{height}(y)$ otherwise. The height of the tree τ is defined

as the height of its root, $\text{height}(\tau) = \text{height}(\text{root}(\tau))$. The *outdegree* $\text{deg}(\tau)$ of τ is the maximal branching factor that can be found in τ , that is $\text{deg}(\tau) = \max_{x \in \tau} \#\text{child}(x)$ ¹.

Non-plane trees. In the present paper, we consider unordered 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 $\tau_1 = (V_1, G_1)$ and $\tau_2 = (V_2, G_2)$ two rooted trees. A one-to-one correspondence $\varphi : V_1 \rightarrow V_2$ is called a *tree isomorphism* if, for any edge $(x, y) \in E_1$, $(\varphi(x), \varphi(y)) \in E_2$. Structures τ_1 and τ_2 are called *isomorphic trees* (denoted $\tau_1 \equiv \tau_2$) whenever there exists a tree isomorphism between them. It should be noted that one may easily determine if two n -vertex trees are isomorphic in $O(n)$ time (see [1, Example 3.2 and Theorem 3.3]). The existence of a tree isomorphism \equiv defines an equivalence relation on the set of rooted trees. The class of *unordered* or *non-plane trees* may be defined as the equivalence classes for the relation \equiv , that is the quotient set of rooted trees by the existence of a tree isomorphism. One may refer the reader to [12, I.5.2. Non-plane trees] for more details on this combinatorial class.

Tree reduction. Let us now consider the equivalence relation \equiv on the set of the subtrees of a tree τ . We consider the quotient graph $\mathcal{Q}(\tau) = (V_{\equiv}, E_{\equiv})$ obtained from τ using this equivalence relation. V_{\equiv} is the set of equivalence classes on the subtrees of τ , while E_{\equiv} is a set of pairs of equivalence classes (C_1, C_2) such that C_2 is (isomorphic to) a subtree of C_1 and $\text{height}(C_1) = \text{height}(C_2) + 1$. In light of [14, Proposition 1], the graph $\mathcal{Q}(\tau)$ is a *directed acyclic graph* (DAG), that is a connected directed graph without path from any vertex x 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 as a subtree 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 (see [14, Definition 3 (Reduction of a tree)]) for more details). Intuitively, the graph $\mathcal{R}(\tau)$ represents the original tree τ without its structural redundancies (see Figure 1).

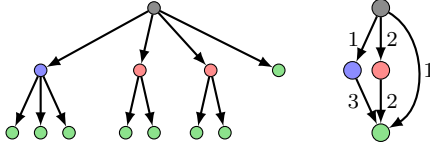


Figure 1: A rooted tree τ (left) and its reduction $\mathcal{R}(\tau)$ (right). In the tree, roots of isomorphic subtrees are colored identically. In the quotient graph, vertices are equivalence classes colored according to the class of isomorphic subtrees of τ that they represent.

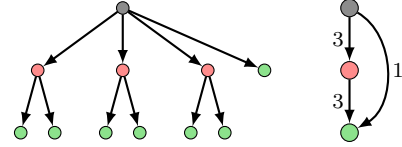


Figure 2: A self-nested tree τ (left) and its linear reduction $\mathcal{R}(\tau)$ (right). In the tree, all the subtrees of the same height are isomorphic and their roots are colored identically. The quotient graph is a linear DAG in which each vertex represents all the subtrees with the same height.

2.2 Self-nested trees

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

$$E[x] = \{(\xi, \xi') \in E : \xi \in V[x], \xi' \in V[x]\}.$$

A tree τ is called *self-nested* (see [14, III. Self-nested trees]) if for any pair of vertices x and y , either the subtrees $\tau[x]$ and $\tau[y]$ are isomorphic, $\tau[x] \equiv \tau[y]$, 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 x and y such that $\text{height}(x) = \text{height}(y)$, $\tau[x] \equiv \tau[y]$, i.e., all the subtrees of the same height are isomorphic.

A *linear DAG* is a DAG containing at least one path that goes through all its vertices. Linear DAGs are tightly connected with self-nested trees and thus play a central role in our investigations about self-nested

¹For the sake of simplicity, we often write $x \in \tau$ instead of $x \in V$, where $\tau = (V, E)$ is a rooted tree.

trees. We also introduce the notion of direct subtree of a vertex. Let τ be a rooted tree and x be one of its vertices. For any vertex $y \in \text{child}(x)$, the subtree $\tau[y]$ is called a *direct subtree* of x in τ .

Proposition 1 (Godin and Ferraro [14]) *A tree τ is self-nested if and only if its reduction $\mathcal{R}(\tau)$ is a linear DAG.*

Proof. Assume that $\mathcal{R}(\tau)$ is a linear DAG with $H + 1$ vertices. Thus $\text{height}(\tau) = H$ and τ contain subtrees of height h for any h between 0 (the leaves) and H (only τ). As a consequence, two vertices of $\mathcal{R}(\tau)$ cannot represent two equivalence classes of subtrees of the same height, and thus τ is self-nested. Reciprocally, if τ is self-nested, there exists only one equivalence class for subtrees of height h , $0 \leq h \leq \text{height}(\tau)$. Thus $\mathcal{R}(\tau)$ contains $\text{height}(\tau) + 1$ vertices. In addition, each subtree of height h in τ has at least one direct subtree of height $h - 1$. Consequently, there exists an edge between the vertex of $\mathcal{R}(\tau)$ representing the equivalence class of height h and the one representing the equivalence class of height $h - 1$, and $\mathcal{R}(\tau)$ is a linear DAG. ■

In light of Proposition 1, the structure of a self-nested tree τ is defined by the numbers n_{h_1, h_2} of direct subtrees of height h_2 rooted in the subtrees of τ of height h_1 , $0 \leq h_2 \leq h_1 - 1$, $1 \leq h_1 \leq \text{height}(\tau)$. It should be noted that the quantities n_{h_1, h_1-1} are constrained to be positive integers: a tree of height h_1 has at least one direct subtree of height $h_1 - 1$. We number the vertices of the linear DAG $\mathcal{R}(\tau)$ from 0 at the bottom (leaves of τ) to $\text{height}(\tau)$ at the top (root of τ) in such a way that there exists a path $\text{height}(\tau) \rightarrow \text{height}(\tau) - 1 \rightarrow \dots \rightarrow 0$. Thus edge $h_1 \rightarrow h_2$, $h_1 \geq h_2 + 1$, is labelled with n_{h_1, h_2} . In this context, the vertex with number h_1 in the DAG represents the unique (up to isomorphism) subtree of height h_1 in τ , which has n_{h_1, h_2} direct subtrees of height h_2 .

The set of admissible labels on the edges of the (linear) DAG of a self-nested tree of height H is given by

$$\mathcal{N}_H = \{(n_{h_1, h_2})_{0 \leq h_2 \leq h_1 - 1 \leq H - 1} : \forall 1 \leq h_1 \leq H, n_{h_1, h_1 - 1} \geq 1\}.$$

An element of \mathcal{N}_H is denoted by $\mathbf{n}_H = (n_{h_1, h_2})_{0 \leq h_2 \leq h_1 - 1 \leq H - 1}$ or $\mathbf{n}_H = (n_{h_1, h_2})$ in a more concise form and without confusion on the height. In this context, $\text{ST}(\mathbf{n}_H)$ denotes the unique self-nested tree of height H defined by labels $\mathbf{n}_H = (n_{h_1, h_2})$ in which subtrees of height h_1 have n_{h_1, h_2} direct subtrees of height h_2 (see Figure 3). The construction of a self-nested tree from the non-negative integers n_{h_1, h_2} is detailed in Algorithm 1. This notation plays a crucial role in our paper. By convention, the notation $\text{ST}(\mathbf{n}_0)$ refers to the tree composed of an isolated root \bullet .

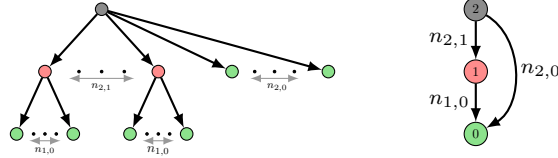


Figure 3: Representation of the self-nested tree $\text{ST}(\mathbf{n}_2)$ of height 2 (left) and its linear DAG (right) with our notation. All the subtrees of the same height are isomorphic and their roots are colored identically.

The number of vertices of a self-nested tree τ may be easily computed from the labels n_{h_1, h_2} on the edges of its linear DAG with complexity $O(\text{height}(\tau)^2)$ (see Lemma 2). This quite simple example shows that some computations may be easier in terms of complexity on a self-nested tree than on a general tree structure thanks to the elimination of redundant computations of duplicated structures. Indeed, the number of vertices of a tree t may be obtained by a depth-first search algorithm in $O(\#t)$ time, that is in the worst case $\#t = m^H$, where $m = \text{deg}(t)$ and $H = \text{height}(t)$. Another example will be given in Subsection 4.4.

Lemma 2 *For any $H \geq 1$, the number of vertices of the self-nested tree $\text{ST}(\mathbf{n}_H)$ may be computed in $O(H^2)$ from the formula*

$$\#\text{ST}(\mathbf{n}_H) = 1 + \sum_{h=0}^{H-1} n_{H, h} \times \#\text{ST}(\mathbf{n}_h),$$

initialized at height 0 by $\#\text{ST}(\mathbf{n}_0) = 1$.

Proof. The tree $\text{ST}(\mathbf{n}_H)$ is composed of a root and $n_{H,h}$ direct subtrees of height h all isomorphic to $\text{ST}(\mathbf{n}_h)$, for each $0 \leq h \leq H - 1$. Initialization is obvious because $\text{ST}(\mathbf{n}_0) = \bullet$ by convention. ■

Algorithm 1: Construction of the self-nested tree $\text{ST}(\mathbf{n}_H)$, $\mathbf{n} \in \mathcal{N}$ and $H \geq 1$.

```

1 Function SelfnestedTree( $\mathbf{n}_H, v = \bullet$ ):
   Data: triangle array  $\mathbf{n}_H \in \mathcal{N}_H$  and current vertex  $v$  (initially an isolated root)
   Result: self-nested tree  $\text{ST}(\mathbf{n}_H)$ 
2 for  $h_2$  in  $\{0, \dots, H - 1\}$  do
3   for  $i$  in  $\{1, \dots, n_{H,h_2}\}$  do
4     add a child  $c$  to  $v$ 
5     call SelfnestedTree( $\mathbf{n}_{h_2}, v = c$ )

```

3 Combinatorics of self-nested trees

We now investigate combinatorics of self-nested trees. This section gathers new results about this problem for trees that satisfy constraints on the height and the outdegree. All the proofs have been deferred into Appendix A.

3.1 Exact results

In this section, we restrict ourselves to finite classes of rooted non-plane trees that satisfy an equality or inequality constraint on the height and the outdegree. In particular, $\mathbb{T}_{=h, \leq m}$ ($\mathbb{T}_{\leq h, \leq m}$, respectively) stands for the set of non-plane trees of height h (height less than h , respectively) and an outdegree less than m . We use the similar notations $\mathbb{T}_{=h, < m}^{sn}$ and $\mathbb{T}_{\leq h, < m}^{sn}$ for the set of self-nested trees under the same conditions. Of course, we have the inclusion $\mathbb{T}_{\leq h, < m}^{sn} \subset \mathbb{T}_{\leq h, \leq m}$. Nevertheless, we would like to be more precise and characterize the relative size of the set of self-nested trees with respect to the size of the set of trees under the above conditions.

Proposition 3 For any $H \geq 1$ and $m \geq 1$,

$$\#\mathbb{T}_{\leq H, \leq m}^{sn} = \sum_{h=1}^H \prod_{i=1}^h (m + h - i).$$

Proof. The reader may find the proof in Appendix A.1. ■

Proposition 4 For any integer $m \geq 1$, let us define the sequence $(u_h(m))_{h \geq 0}$ by

$$u_0(m) = 1 \quad \text{and} \quad u_h(m) = \binom{u_{h-1}(m) + m}{m} \quad \text{for } h \geq 1.$$

Then, for any integer $H \geq 1$,

$$\#\mathbb{T}_{\leq H, \leq m} = u_H(m) - 1.$$

Proof. The reader may find the proof in Appendix A.2. ■

By virtue of Propositions 3 and 4, we can analyze the cardinality of the self-nested trees with respect to that of the non-planar trees. We compute the ratio $\#\mathbb{T}_{\leq H, < m}^{sn} / \#\mathbb{T}_{\leq H, \leq m}$ for values of H and m (see Table 1). An exhaustive enumeration of $\mathbb{T}_{\leq 3, \leq 2}$ is presented in Figure 4.

		outdegree		
		≤ 2	≤ 3	≤ 4
height	≤ 2	0.88	6.18×10^{-1}	3.52×10^{-1}
	≤ 3	0.49	3.38×10^{-2}	7.43×10^{-5}
	≤ 4	0.07	2.90×10^{-8}	4.16×10^{-23}
	≤ 5	3.36×10^{-4}	3.56×10^{-28}	1.66×10^{-100}

Table 1: Relative frequencies of self-nested trees with given maximal height and ramification number within the set of non-planar trees under the same constraint.

Remark 5 *A more traditional approach in the literature is to investigate combinatorics of trees with a given number of vertices. For example, exploiting the theory of ordinary generating functions, Flajolet and Sedgewick recursively obtained the cardinality of the set \mathbb{T}_n of unordered trees with n vertices (see [12, eq. (73)] and OEIS² A000081). In particular, the generating function associated with the non-plane trees is given by*

$$H(z) = z + z^2 + 2z^3 + 4z^4 + 9z^5 + 20z^6 + 48z^7 + 115z^8 + 286z^9 + \dots,$$

where the coefficient H_n of z^n in $H(z)$ is the cardinality of the set \mathbb{T}_n . It would be very interesting to investigate the cardinality of self-nested trees under the same constraint \mathbb{T}_n^{sn} , but it appears to be out of our reach. A strategy could be to remark that

$$\#\mathbb{T}_n^{sn} = \sum_{h=1}^n \#\{\mathbf{n}_h : \#\text{ST}(\mathbf{n}_h) = n\},$$

where $\#\{\mathbf{n}_h : \#\text{ST}(\mathbf{n}_h) = n\}$ denotes the number of solutions to the Diophantine equation $\#\text{ST}(\mathbf{n}_h) = n$. In light of Lemma 2, each of these Diophantine equations is polynomial of degree h in $h(h+1)/2$ unknown variables. Nevertheless, determining the number of solutions of such a Diophantine equation, even in this particular framework, remains a very difficult question.

3.2 Asymptotics

In light of Table 1, the number of non-plane trees seems to increase very much faster than the quantity of self-nested trees under the same constraint. For this, let us determine asymptotic equivalents for both cardinalities.

Corollary 6 *When h and m simultaneously go to infinity,*

$$\log \#\mathbb{T}_{=h, \leq m}^{sn} \sim \frac{(m+h)^2}{2} \log(m+h) - \frac{h^2}{2} \log h - \frac{m^2}{2} \log m - hm \log m.$$

Proof. The reader may find the proof in Appendix A.3. ■

Now, we focus on the cardinality of unordered trees.

Corollary 7 *For any integers $m \geq 1$ and $H \geq 3$,*

$$\exp \left[m^{H-1} \log \left(2 + \frac{1}{m} \right) - \left(\frac{m^{H-1} - 1}{m-1} - 1 \right) \log m \right] - 1 \leq \#\mathbb{T}_{\leq H, \leq m} \leq \exp \left[m^{H-1} \log 3 + \frac{m^H - 1}{m-1} - 1 \right].$$

Proof. The reader may find the proof in Appendix A.4. ■

By virtue of Corollary 7, the cardinality $\#\mathbb{T}_{\leq H, \leq m}$ roughly increases as $\exp(m^{H-1})$ for large parameters m and H , which is indeed very much faster than the rate obtained for self-nested trees in Corollary 6.

²On-line Encyclopedia of Integer Sequences

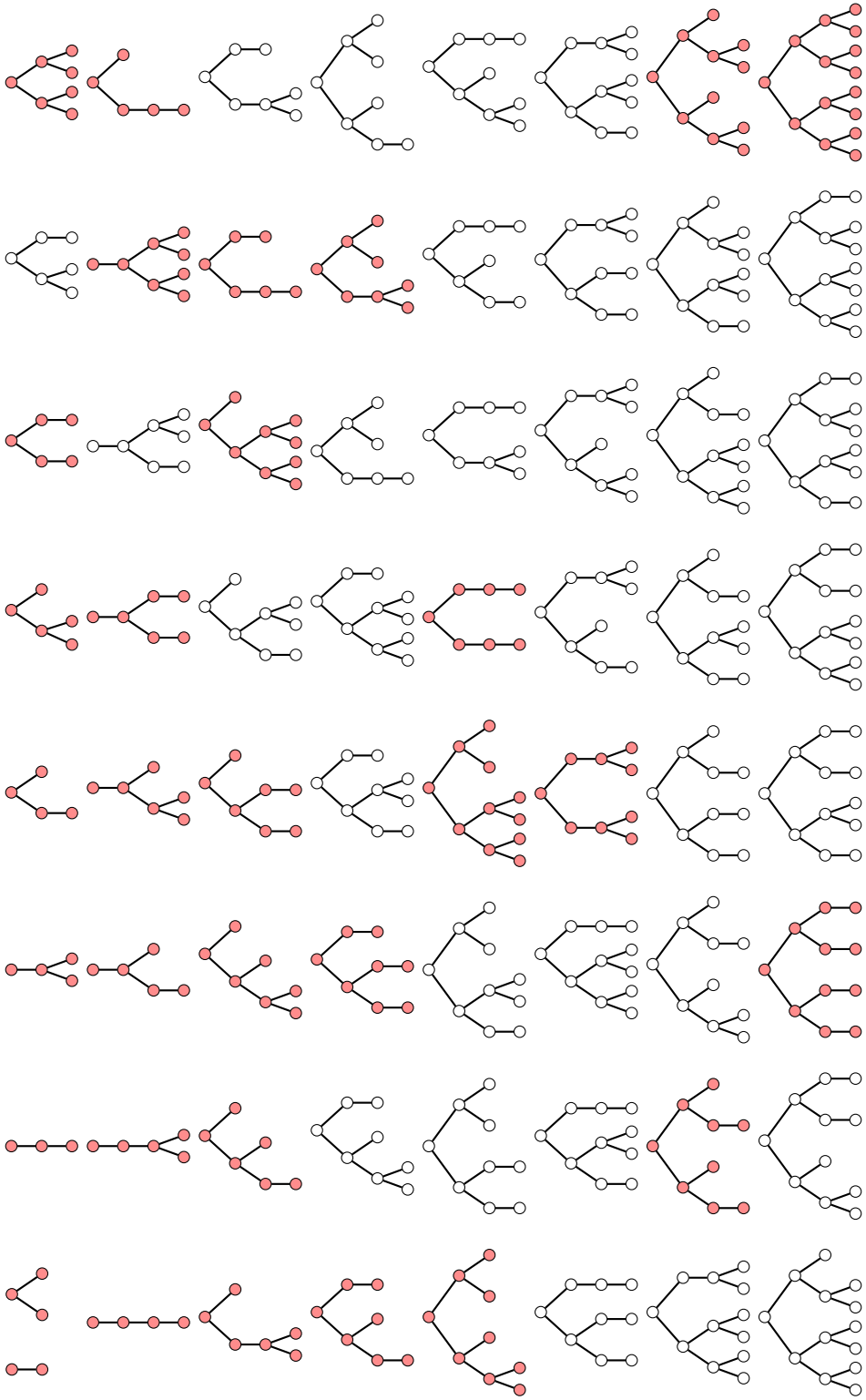


Figure 4: A representation of $\mathbb{T}_{\leq 3, \leq 2}$ with 32 colored self-nested trees among 65 unrooted rooted trees. The root is always drawn at the top.

4 Constrained edit distance

In this paper, our aim is to build approximation algorithms that provide self-nested estimates of trees. Here we introduce a distance on the space of unordered trees in order to quantify the quality of the estimates obtained from these algorithms. The problem of comparing trees occurs in several diverse areas such as computational biology and image analysis. We refer the reader to the survey [3] in which the author reviews the available results and presents, in detail, one or more of the central algorithms for solving the problem.

4.1 Definition

We consider a constrained edit distance between unordered rooted trees. This distance is based on the following *tree edit operations* [8]:

- *Insertion.* Let v be a vertex in a tree τ . The insertion operation inserts a new vertex in the list of children of v . In the transformed tree, the new vertex is necessarily a leaf.
- *Deletion.* Let l be a leaf vertex in a tree τ . The deletion operation results in removing l from τ . That is, if v is the parent of vertex l , the list of children of v in the transformed tree is $\text{child}(v) \setminus \{l\}$.

As in [8] for ordered trees, only adding and deleting a leaf vertex are allowed edit operations. An *edit script* is an ordered sequence of edit operations. The result of applying an edit script s to a tree τ is the tree τ^s obtained by applying the component edit operations to τ , in the order they appear in the script. The *cost* of an edit script s is only the number of edit operations $\#s$. In other words, we assign a unit cost to both allowed operations. Finally, given two unordered rooted trees τ_1 and τ_2 , the *constrained edit cost* $\delta(\tau_1, \tau_2)$ is the length of the minimum edit script that transforms τ_1 to a tree that is isomorphic to τ_2 ,

$$\delta(\tau_1, \tau_2) = \min_{\{s : \tau_1^s \equiv \tau_2\}} \#s.$$

We refer the reader to Figure 5 for an example of minimum-length edit script. We also point out that if $\hat{\tau}$ denotes the NEST of a tree τ and n the number of vertices that have been added to τ to obtain $\hat{\tau}$, one has $\delta(\tau, \hat{\tau}) = n$. The numerical results of this paper obtained from δ are thus fully comparable with those in [14] while the distance used is slightly different.

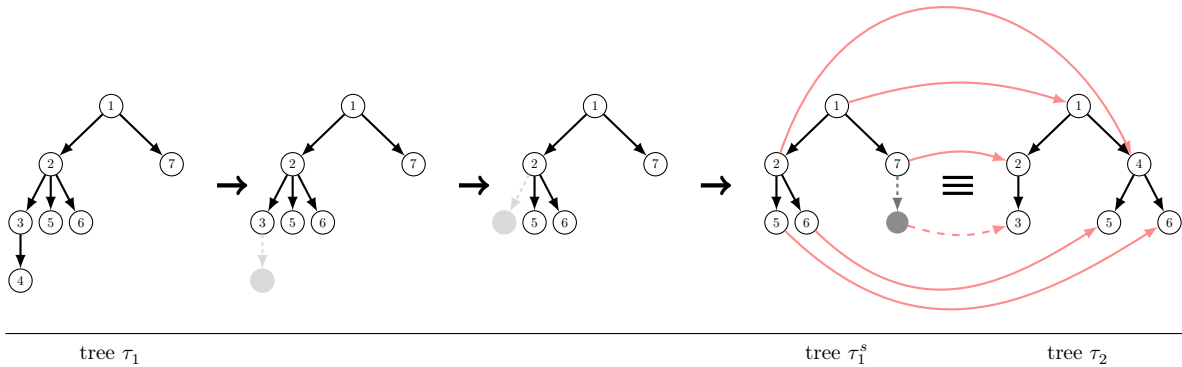


Figure 5: A minimum-cost edit script that transforms τ_1 (left) into τ_1^s that is a tree isomorphic to τ_2 (right) after 3 operations: delete leaf 4, delete leaf 3, add a child to vertex 7. The tree isomorphism illustrated by red arrows between the transformed version τ_1^s of τ_1 and τ_2 defines the tree mapping (only full red arrows) ($1 \rightarrow 1, 2 \rightarrow 4, 5 \rightarrow 6, 6 \rightarrow 5, 7 \rightarrow 2$) from τ_1 to τ_2 .

Proposition 8 δ defines a distance function on the space of unordered rooted trees \mathbb{T} . δ is now called constrained edit distance or simply edit distance.

Proof. The separation axiom is obviously satisfied by δ because of its definition as a cardinality. In addition, $\tau_1 \equiv \tau_2$ if and only if the empty script \emptyset satisfies $\tau_1^\emptyset \equiv \tau_2$, so that the coincidence axiom is checked. Symmetry is obvious by applying in the reverse order the reverse operations of a script s . Finally, if s (σ , respectively) denotes a minimum-length script to transform τ_1 into τ_2 (τ_2 into τ_3 , respectively), the script $s\sigma$ obtained as the concatenation of both these scripts transforms τ_1 into τ_3 . The triangle inequality is thus satisfied,

$$\delta(\tau_1, \tau_3) \leq \#(s\sigma) = \#s + \#\sigma = \delta(\tau_1, \tau_2) + \delta(\tau_2, \tau_3),$$

which yields the expected result. ■

4.2 Relation with tree mappings

Here we address the issue of equivalence between edit distance and tree mapping cost using the particular edit distance δ . Such equivalence has been discussed in [20, 22] in the context of other edit distances.

Mapping. Let τ_1 and τ_2 be two trees. Suppose that we have a numbering of the vertices for each tree. Since we are concerned with unordered trees, we can fix an arbitrary order for each of the vertex in the tree and then use left-to-right postorder numbering or left-to-right preorder numbering. A *mapping* \mathcal{M} from τ_1 to τ_2 is a set of couples $i \rightarrow j$, $1 \leq i \leq \#\tau_1$ and $1 \leq j \leq \#\tau_2$, satisfying (see [22, 2.3.2 Editing Distance Mappings]), for any $i_1 \rightarrow j_1$ and $i_2 \rightarrow j_2$ in \mathcal{M} , the following assumptions:

- $i_1 = i_2$ if and only if $j_1 = j_2$.
- vertex i_1 in τ_1 is an ancestor of vertex i_2 in τ_1 if and only if vertex j_1 in τ_2 is an ancestor of vertex j_2 in τ_2 .

Constrained tree mapping. Let τ_1 and τ_2 be two trees, s be a script such that $\tau_1^s \equiv \tau_2$ and φ a tree isomorphism between τ_1^s and τ_2 . The graph $\tau_1 \cap \tau_1^s$ defines a tree embedded in τ_1 because script s only added and deleted leaves. As a consequence, the function $\widehat{\varphi}$ defined as φ restricted to $\tau_1 \cap \tau_1^s$ provides a tree mapping from τ_1 to τ_2 with $i \rightarrow j$ if and only if $\widehat{\varphi}(i) = j$. Of course, this is a particular tree mapping since it has been obtained from very special conditions. The main additional condition is the following: for any $i_1 \rightarrow j_1$ and $i_2 \rightarrow j_2$,

- vertex i_1 is the parent of vertex i_2 in τ_1 if and only if vertex j_1 is the parent of vertex j_2 in τ_2 .

It is easy to see that this assumption is actually the only required additional constraint to define the class of constrained tree mappings cTM involved in the computation of our constrained edit distance δ (see the example presented in Figure 5). The equivalence between constrained mappings of cTM and δ may be stated as follows:

$$\delta(\tau_1, \tau_2) = \min_{\mathcal{M} \in \text{cTM}} \#\{i : \nexists j \text{ s.t. } i \rightarrow j \in \mathcal{M}\} + \#\{j : \nexists i \text{ s.t. } i \rightarrow j \in \mathcal{M}\}.$$

The equivalence between tree mapping cost and edit distance is a classical property used in the computation of the edit distance.

The mappings involved in our constrained edit distance have other properties related to some previous works of the literature. We present additional definitions, namely, the lowest common ancestor of two vertices and the constrained mappings presented in [22].

Lowest common ancestor. The lowest common ancestor (LCA) of two vertices v and w in a same tree is the lowest (i.e., least height) vertex that has both v and w as descendants. In other words, the LCA is the shared ancestor that is located farthest from the root. It should be noted that if v is a descendant of w , w is the LCA.

Constrained mapping with Zhang’s distance. Tanaka and Tanaka proposed in [20] the following condition for mapping ordered labeled trees: disjoint subtrees should be mapped to disjoint subtrees. They showed that in some applications (e.g., classification tree comparison) this kind of mapping is more meaningful than more general edit distance mappings. Zhang investigated in [22] the problem of computing the edit distance associated with this kind of constrained mapping between unordered labeled trees. Precisely, a *constrained mapping* \mathcal{M} between trees τ_1 and τ_2 is a mapping satisfying the additional condition (see [22, 3.1. Constrained Edit Distance Mappings]):

- Assume that $i_1 \rightarrow j_1, i_2 \rightarrow j_2$ and $i_3 \rightarrow j_3$ are in \mathcal{M} . Let v (w , respectively) be the LCA of vertices i_1 and i_2 in τ_1 (of vertices j_1 and j_2 in τ_2 , respectively). v is a proper ancestor of vertex i_3 in τ_1 if and only if w is a proper ancestor of vertex j_3 in τ_2 .

Let τ_1 and τ_2 be two trees and $\mathcal{M} \in \text{cTM}$. First, one may remark that the roots are necessarily mapped together. In addition, \mathcal{M} satisfies all the conditions of constrained mappings imposed by Zhang in [22] and presented above (see again the example of Figure 5).

4.3 Distance computation and reduction to the minimum cost flow problem

The edit distance between two trees T_1 and T_2 may be obtained from the recursive formula presented in Proposition 9 hereafter. In the sequel, the forest of direct subtrees of the root of a tree t is denoted by \mathcal{F}_t . Furthermore, $S(n)$ denotes the set of permutations of $\{1, \dots, n\}$ and $\binom{A}{n}$ the set of subsets of A with cardinality n .

Proposition 9 *Let T_1 and T_2 be two trees and $n = \min(\#\mathcal{F}_{T_1}, \#\mathcal{F}_{T_2})$. The edit distance between T_1 and T_2 satisfies the following induction formula,*

$$\delta(T_1, T_2) = \min_{\{t_1, \dots, t_n\} \in \binom{\mathcal{F}_{T_1}}{n}} \min_{\{\tau_1, \dots, \tau_n\} \in \binom{\mathcal{F}_{T_2}}{n}} \min_{\sigma \in S(n)} \sum_{i=1}^n \delta(t_i, \tau_{\sigma(i)}) + \sum_{\theta \in \mathcal{F}_{T_1} \setminus \{t_1, \dots, t_n\}} \delta(\theta, \emptyset) + \sum_{\theta \in \mathcal{F}_{T_2} \setminus \{\tau_1, \dots, \tau_n\}} \delta(\emptyset, \theta),$$

initialized with

$$\delta(T_1, \emptyset) = \#T_1 \quad \text{and} \quad \delta(\emptyset, T_2) = \#T_2,$$

where the symbol \emptyset stands for the empty tree.

Proof. First, let us remark that a maximum number of direct subtrees of T_1 should be mapped to direct subtrees of T_2 , because $\delta(\theta_1, \theta_2) < \delta(\theta_1, \emptyset) + \delta(\emptyset, \theta_2)$, for any trees θ_1 and θ_2 . This maximum number is $n = \min(\#\mathcal{F}_{T_1}, \#\mathcal{F}_{T_2})$. As a consequence, the minimal editing cost is obtained by considering all the possible mappings between n direct subtrees of T_1 and n direct subtrees of T_2 . The direct subtrees that are not involved in a mapping are either deleted or added. We refer the reader to Figure 6. ■

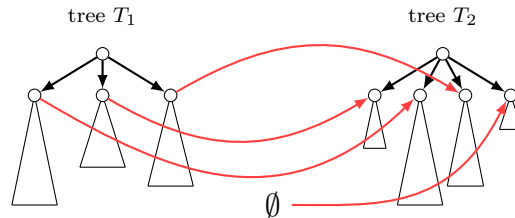


Figure 6: Schematic illustration of the recursive formula to compute the constrained edit distance δ between trees T_1 and T_2 . Edit script to transform T_1 into T_2 : the three direct subtrees of T_1 are mapped to three direct subtrees of T_2 , while the empty tree \emptyset is mapped to the fourth direct subtree of T_2 (all the vertices are added).

In light of Proposition 9 and Figure 6 and as in [22, 5. Algorithm and complexity], each step in the recursive computation of the edit distance $\delta(T_1, T_2)$ between trees T_1 and T_2 reduces to the minimum cost maximum flow problem on a graph $G = (V, E)$ constructed as follows. First the set of vertices V of G is defined by

$$V = \{\text{source}, \text{sink}, \emptyset_{T_1}, \emptyset_{T_2}\} \cup \mathcal{F}_{T_1} \cup \mathcal{F}_{T_2}.$$

The set E of edges of G is defined from:

- edge $\text{source} \rightarrow t_i, t_i \in \mathcal{F}_{T_1}$: capacity 1 and cost 0;
- edge $\text{source} \rightarrow \emptyset_{T_1}$: capacity $\#\mathcal{F}_{T_1} - \min(\#\mathcal{F}_{T_1}, \#\mathcal{F}_{T_2})$ and cost 0;
- edge $t_i \rightarrow \tau_j, t_i \in \mathcal{F}_{T_1}, \tau_j \in \mathcal{F}_{T_2}$: capacity 1 and cost $\delta(t_i, \tau_j)$;
- edge $t_i \rightarrow \emptyset_{T_2}, t_i \in \mathcal{F}_{T_1}$: capacity 1 and cost $\delta(t_i, \emptyset) = \#t_i$;
- edge $\emptyset_{T_1} \rightarrow \tau_j, \tau_j \in \mathcal{F}_{T_2}$: capacity 1 and cost $\delta(\emptyset, \tau_j) = \#\tau_j$;
- edge $\tau_j \rightarrow \text{sink}, \tau_j \in \mathcal{F}_{T_2}$: capacity 1 and cost 0;
- edge $\emptyset_{T_2} \rightarrow \text{sink}$: capacity $\#\mathcal{F}_{T_2} - \min(\#\mathcal{F}_{T_1}, \#\mathcal{F}_{T_2})$ and cost 0.

We obtain a network G augmented with integer capacities and nonnegative costs. A representation of G is given in Figure 7. By construction and as explained in [22, Lemma 8], one has $C(G) = \delta(T_1, T_2)$ where $C(G)$ denotes the cost of the minimum cost maximum flow on G . As a consequence, $\delta(T_1, T_2)$ may directly be computed from a minimum cost maximum flow algorithm presented for example in [21, 8.4 Minimum cost flows]. The related complexity is given in Proposition 10.

Proposition 10 $\delta(T_1, T_2)$ may be computed in $O(\#T_1 \times \#T_2 \times [\deg(T_1) + \deg(T_2)] \times \log_2(\deg(T_1) + \deg(T_2)))$.

Proof. In light of [21, Theorem 8.13], the complexity of finding the cost of the minimum cost maximum flow on the network G defined in Figure 7 may be directly obtained from its characteristics and is $O(N \times |f^*| \times \log_2(n))$, where n, N and $|f^*|$ respectively denote the number of vertices, the number of edges and the maximum flow of G . It is quite obvious that:

- $N = O(\#\text{child}(T_1) \times \#\text{child}(T_2) + \#\text{child}(T_1) + \#\text{child}(T_2))$;
- $|f^*| = O(\#\text{child}(T_1) + \#\text{child}(T_2))$;
- $n = O(\#\text{child}(T_1) + \#\text{child}(T_2))$.

Thus, the total complexity to compute the recursive formula of $\delta(T_1, T_2)$ presented in Proposition 9 is

$$\begin{aligned} & \sum_{t \in T_1} \sum_{\tau \in T_2} O(\#\text{child}(t) \times \#\text{child}(\tau) \times [\#\text{child}(t) + \#\text{child}(\tau)] \times \log_2(\#\text{child}(t) + \#\text{child}(\tau))) \\ & \leq O([\deg(T_1) + \deg(T_2)] \times \log_2(\deg(T_1) + \deg(T_2)) \times \sum_{t \in T_1} \sum_{\tau \in T_2} \#\text{child}(t) \times \#\text{child}(\tau)) \\ & \leq O(\#T_1 \times \#T_2 \times [\deg(T_1) + \deg(T_2)] \times \log_2(\deg(T_1) + \deg(T_2))), \end{aligned}$$

which yields the expected result. ■

Not surprisingly, the time-complexity of computing the edit distance δ is the same as in [22] for another kind of constrained edit distance. It should be noted that this algorithm does not take into account the possible presence of redundant substructures, which should reduce the complexity. We tackle this question in the following part.

4.4 Complexity of distance computation for self-nested trees

The compression methods that we will present in the sequel require to compute the edit distance between a tree and its self-nested estimates. Consequently, the complexity of our algorithms highly depends on the computation of the edit distance involving self-nested trees. The tree-to-tree comparison problem has already been considered for quotiented trees (an adaptation of Zhang's algorithm [22] to quotiented trees is presented in [11]), but never in the specific framework of self-nested trees. Because of the systematic presence of redundancies, the edit distance between two self-nested trees or between one tree and a self-nested one should be computed with a time-complexity smaller than in Proposition 10. We investigate this question in particular for the sake of reducing the complexities of the approximation algorithms presented in Section 5.

As a first step, we only deal with the edit distance between two self-nested trees $\text{ST}(\mathbf{n}_H)$ and $\text{ST}(\mathbf{n}'_{H'})$. The computational complexity in the case of distances between self-nested and non-self-nested trees may be addressed in a similar way, also using the previous result of Subsection 4.3. The computation of the edit distance $\delta(\text{ST}(\mathbf{n}_H), \text{ST}(\mathbf{n}'_{H'}))$ reduces to a minimum cost flow problem but the network graph that we consider takes into account the number of appearances of a given pattern among the lists of direct subtrees of $\text{ST}(\mathbf{n}_H)$ and $\text{ST}(\mathbf{n}'_{H'})$. We construct a graph $G = (V, E)$ as follows. The set of vertices V of G is given by

$$V = \{\text{source}, \text{sink}, \emptyset, \emptyset'\} \cup \bigcup_{0 \leq h \leq H-1} \{\text{ST}(\mathbf{n}_h)\} \cup \bigcup_{0 \leq h \leq H'-1} \{\text{ST}(\mathbf{n}'_h)\}.$$

The set E of edges of G is defined from:

- edge $\text{source} \rightarrow \text{ST}(\mathbf{n}_h)$, $0 \leq h \leq H-1$: capacity $n_{H,h}$ and cost 0;
- edge $\text{source} \rightarrow \emptyset$: capacity $\sum_{0 \leq h \leq H-1} n_{H,h} - \min(\sum_{0 \leq h \leq H-1} n_{H,h}, \sum_{0 \leq h \leq H'-1} n'_{H',h})$ and cost 0;
- edge $\text{ST}(\mathbf{n}_h) \rightarrow \text{ST}(\mathbf{n}'_{h'})$, $0 \leq h \leq H-1$, $0 \leq h' \leq H'-1$: capacity $n_{H,h}$ and cost $\delta(\text{ST}(\mathbf{n}_h), \text{ST}(\mathbf{n}'_{h'}))$;
- edge $\text{ST}(\mathbf{n}_h) \rightarrow \emptyset'$, $0 \leq h \leq H-1$: capacity $n_{H,h}$ and cost $\delta(\text{ST}(\mathbf{n}_h), \emptyset) = \#\text{ST}(\mathbf{n}_h)$;
- edge $\emptyset \rightarrow \text{ST}(\mathbf{n}'_h)$, $0 \leq h \leq H'-1$: capacity $n'_{H',h}$ and cost $\delta(\emptyset, \text{ST}(\mathbf{n}'_h)) = \#\text{ST}(\mathbf{n}'_h)$;
- edge $\text{ST}(\mathbf{n}'_h) \rightarrow \text{sink}$, $0 \leq h \leq H'-1$: capacity $n'_{H',h}$ and cost 0;
- edge $\emptyset' \rightarrow \text{sink}$: capacity $\sum_{0 \leq h \leq H'-1} n'_{H',h} - \min(\sum_{0 \leq h \leq H-1} n_{H,h}, \sum_{0 \leq h \leq H'-1} n'_{H',h})$ and cost 0.

As in Subsection 4.3, the graph G has integer capacities and nonnegative costs on its edges (see Figure 8). By construction, the cost $C(G)$ of the minimum cost maximum flow on the graph G is equal to the expected edit distance $\delta(\text{ST}(\mathbf{n}_H), \text{ST}(\mathbf{n}'_{H'}))$. The related complexity is presented in Proposition 11. The computation of the edit distance between a tree and a self-nested structure also reduces to a minimum cost flow problem but we skip this case which may be easily derived from graphs presented in Figures 7 and 8. Nevertheless, we give the related complexity in Proposition 11.

Proposition 11

- $\delta(\text{ST}(\mathbf{n}_H), \text{ST}(\mathbf{n}'_{H'}))$ may be computed in $O(H^2 \times H'^2 \times [\deg(\text{ST}(\mathbf{n}_H)) + \deg(\text{ST}(\mathbf{n}'_{H'}))] \times \log_2(H + H'))$.
- $\delta(T, \text{ST}(\mathbf{n}_H))$ may be computed in $O(\#T \times H^2 \times [\deg(T) + \deg(\text{ST}(\mathbf{n}_H))] \times \log_2(\deg(T) + H))$.

Proof. We only state the first item. The proof is quite similar to the demonstration of Proposition 10. The characteristics n (number of vertices), $|f^*|$ (maximum flow) and N (number of edges) of the network G may be easily found from Figure 8:

- $N = O(H \times H' + H + H')$;
- $|f^*| = O(\deg(\text{ST}(\mathbf{n}_H)) + \deg(\text{ST}(\mathbf{n}'_{H'})))$;
- $n = O(H + H')$.

Together with [21, Theorem 8.13], this states the result. ■

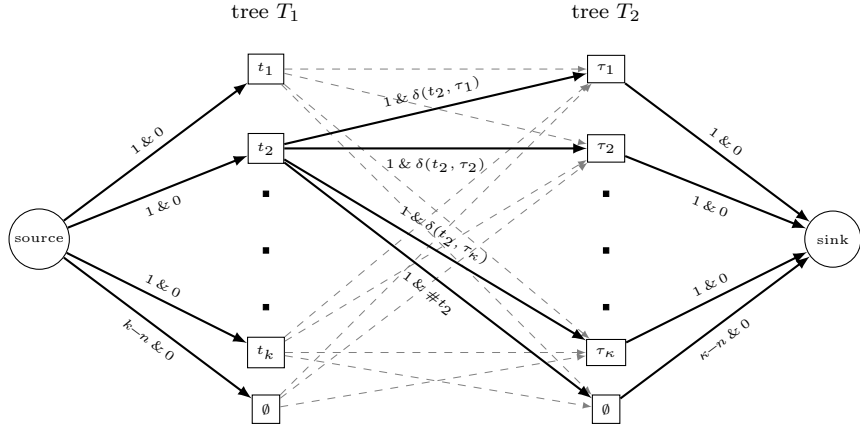


Figure 7: Reduction of the computation of edit distance $\delta(T_1, T_2)$ presented in Proposition 9 and in Figure 6 to the minimum cost flow problem. Each edge is augmented with two labels separated by the symbol $\&$: its capacity (left) and its cost (right). For the sake of simplicity, k (κ , respectively) denotes $\#\mathcal{F}_{T_1}$ ($\#\mathcal{F}_{T_2}$, respectively), and $n = \min(k, \kappa)$.

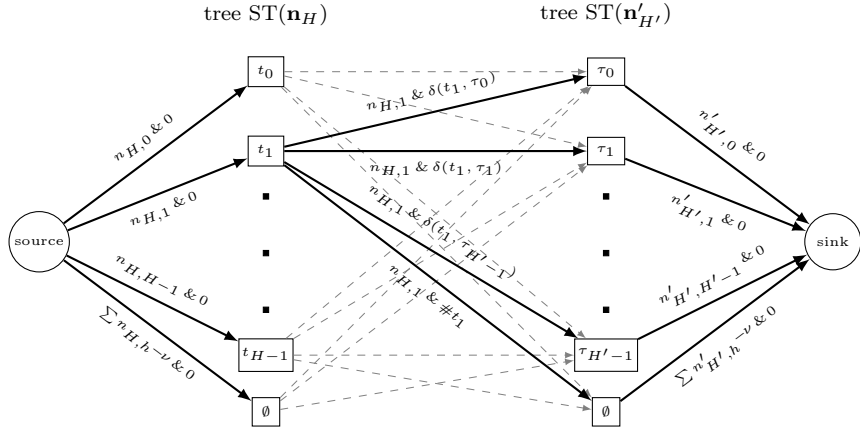


Figure 8: Reduction of the computation of the edit distance $\delta(\text{ST}(\mathbf{n}_H), \text{ST}(\mathbf{n}'_{H'}))$ between two self-nested trees to the minimum cost flow problem. As in Figure 7 in the general case, each edge is augmented with an integer capacity and a cost. For the sake of simplicity, we use the following notations: $t_h = \text{ST}(\mathbf{n}_h)$, $\tau_h = \text{ST}(\mathbf{n}'_h)$ and $\nu = \min(\sum_{0 \leq h \leq H-1} n_{H,h}, \sum_{0 \leq h \leq H'-1} n'_{H',h})$.

5 Self-nested approximation

Having defined the distance δ on the space of unordered trees, this section is devoted to the presentation of two algorithms to compute an accurate self-nested approximation of a tree T that next will be highly compressed by DAG method. We also investigate the worst approximation error that may be obtained by such an algorithm.

5.1 Theoretical considerations on the worst case

Let us consider a tree T to be compressed. Our strategy consists in finding a self-nested tree \hat{T} that approximates T . To achieve this goal, we would like to minimize the function $\tau \mapsto \delta(T, \tau)$. We investigate the worst-case approximation error that may be achieved, i.e., we search among trees of height H and maximal outdegree m a tree T that is the farthest from its best self-nested approximation \hat{T} . We state in Proposition 12 that such a tree is $\Theta_{H,m}$ defined by its DAG in Figure 9. One of its best self-nested estimates is $\mathcal{T}_{H,m}$, also defined in Figure 9. Two examples are displayed in Figure 10.

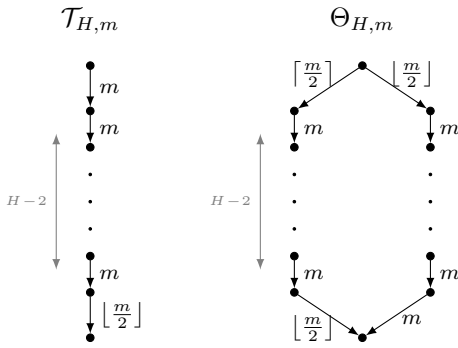


Figure 9: Definition of the trees $\mathcal{T}_{H,m}$ (left) and $\Theta_{H,m}$ (right) from their DAG. $\Theta_{H,m}$ is one of the least self-nested trees among $\mathbb{T}_{\leq H, \leq m}$ and one of its nearest self-nested trees is given by $\mathcal{T}_{H,m}$.

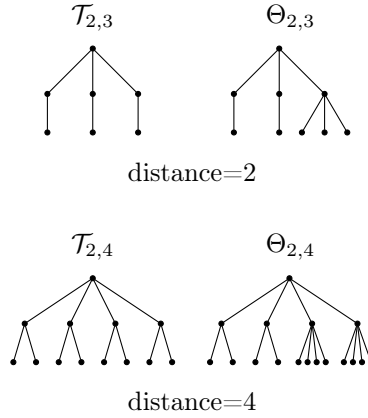


Figure 10: Trees $\mathcal{T}_{H,m}$ and $\Theta_{H,m}$ for $H = 2$ and $m = 3$ or $m = 4$.

Proposition 12 For any $H \geq 2$ and m large enough (greater than a constant depending on H),

$$\begin{aligned} \max_{t \in \mathbb{T}_{\leq H, \leq m}} \min_{\tau \in \mathbb{T}_{\leq H, \leq m}^n} \delta(t, \tau) &= \delta(\Theta_{H,m}, \mathcal{T}_{H,m}) \\ &= \left\lfloor \frac{m}{2} \right\rfloor \times \left\lceil \frac{m}{2} \right\rceil \times m^{H-2}, \end{aligned}$$

where the trees $\mathcal{T}_{H,m}$ and $\Theta_{H,m}$ are defined in Figure 9.

Proof. The reader may find the proof in Appendix B. ■

The diameter of the state space $\mathbb{T}_{\leq H, \leq m}$ is of order m^H (indeed, the largest tree of this family is the full m -tree, while the smallest tree is reduced to a unique root vertex). As a consequence and in light of Proposition 12, the largest area without any self-nested tree is a ball with relative radius

$$\begin{aligned} \frac{\left\lfloor \frac{m}{2} \right\rfloor \times \left\lceil \frac{m}{2} \right\rceil \times m^{H-2}}{m^H} &= \frac{\left\lfloor \frac{m}{2} \right\rfloor \times \left\lceil \frac{m}{2} \right\rceil}{m^2} \\ &= \frac{1}{4} + \frac{1}{4m^2} \mathbb{1}_{2\mathbb{N}+1}(m) \simeq \frac{1}{4}. \end{aligned}$$

This establishes a remarkable property of the space of self-nested trees: it is impossible to approximate a tree by a self-nested one with a relative error less than $1/4$. This result is especially noteworthy considering the very low frequency of self-nested trees compared to unordered trees (see Table 1 and Subsection 3.2).

5.2 Replace forests by their centroid

NEST algorithm only adds vertices to get a self-nested structure from a given tree. In many cases, the number of vertices to add is large compared to the size of the tree and other solutions may be preferable (see the example given in Figure 11). All the subtrees of a same height appearing in a self-nested tree are isomorphic. Consequently, instead of only adding vertices, our strategy consists in replacing all the subtrees of a same height by a same structure. In other words, we replace some internal structures by their self-nested centroid (i.e., the self-nested tree minimizing the distance to these structures). In particular, this allows us to delete some vertices and thus to gain in flexibility with respect to NEST.

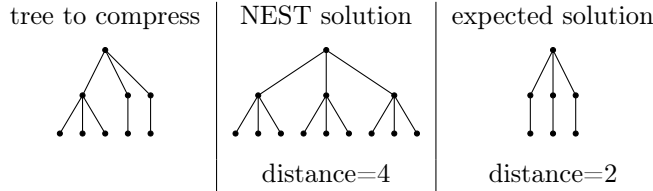


Figure 11: The tree to compress is given in the left column. NEST algorithm adds 4 vertices to get a self-nested structure, whereas a better solution may be expected by deleting only 2 leaves (tree in the right column).

Let M_{h_1} be the number of subtrees of height h_1 in the tree T , $1 \leq i \leq M_{h_1}$ be the index of one of these subtrees and r_i be its root vertex. For $h_2 < h_1$, $T[r_i]$ has $\mu_{h_1, h_2}^{(i)}$ direct subtrees of height h_2 . Without loss of generality, we assume that the sequence $(\mu_{h_1, h_2}^{(i)})_{1 \leq i \leq M_{h_1}}$ is sorted in increasing order. Our compression algorithm relies on the following result that is a direct consequence of Proposition 1.

Lemma 13 *The tree T is self-nested if and only if, for any heights $h_1 \geq h_2 + 1$, the multiset*

$$\mathcal{M}_{h_1, h_2} = \{\mu_{h_1, h_2}^{(i)} : 1 \leq i \leq M_{h_1}\}$$

has only one element with multiplicity M_{h_1} .

If T is not a self-nested tree, some of the multisets \mathcal{M}_{h_1, h_2} are not reduced to a singleton. In this case, we propose to approximate T by replacing the $\mu_{h_1, h_2}^{(i)}$ direct subtrees of height h_2 appearing in the i^{th} subtree of height h_1 by $\bar{\mu}_{h_1, h_2}$ subtrees of height h_2 , where $\bar{\mu}_{h_1, h_2}$ is one centroid of the multiset \mathcal{M}_{h_1, h_2} . Of course, there may exist several combinations of centroids. We choose the best possibility in terms of edit distance. In other words, the self-nested trees RFC(T) (Replace Forests by their Centroid) that we propose to approximate T with are

$$\text{RFC}(T) = \text{ST}(\mathbf{n}_H^*) \quad \text{with} \quad \mathbf{n}_H^* \in \arg \min_{\{\mathbf{n}_H : n_{h_1, h_2} \in \Lambda_{h_1, h_2}\}} \delta(T, \text{ST}(\mathbf{n}_H)),$$

where $H = \text{height}(T)$ and Λ_{h_1, h_2} is the set of centroids $\bar{\mu}_{h_1, h_2}$ of \mathcal{M}_{h_1, h_2} . There remains the question of how to find the set Λ_{h_1, h_2} of centroids of the multiset \mathcal{M}_{h_1, h_2} . Formally, this is equivalent to minimizing the cost function

$$\varphi_{h_1, h_2} : n_{h_1, h_2} \mapsto \sum_{i=1}^{M_{h_1}} |\mu_{h_1, h_2}^{(i)} - n_{h_1, h_2}|.$$

Lemma 14 *There are two possibilities:*

- φ_{h_1, h_2} has only one absolute minimum $\mu_{h_1, h_2}^{(i^*)}$.
- There exists an index i^* such that $\mu_{h_1, h_2}^{(i^*)}$ and $\mu_{h_1, h_2}^{(i^*+1)}$ minimize φ_{h_1, h_2} and thus all the integers between $\mu_{h_1, h_2}^{(i^*)}$ and $\mu_{h_1, h_2}^{(i^*+1)}$ also minimize φ_{h_1, h_2} .

Proof. The function φ_{h_1, h_2} is convex and piecewise-linear because it is obtained as a sum of convex piecewise-linear functions. In addition, slope changing may occur only at values $\mu_{h_1, h_2}^{(i)}$. The expected result follows. ■

In light of Lemma 14, an exhaustive search among the $\mu_{h_1, h_2}^{(i)}$ enables all centroids $\bar{\mu}_{h_1, h_2}$ – and thus $\text{RFC}(T)$ – to be found. This procedure provides first self-nested estimates of the tree T among which we choose those that are closest to T (see Algorithm 2, the time-complexity is given in Proposition 15).

Algorithm 2: Computation of self-nested estimates $\text{RFC}(T)$ of a tree T .

```

1 Function RFC():
   Data: an unordered rooted tree  $T$  of height  $H$ 
   Result: list of compressed versions of  $T$  augmented with their editing cost
2 for  $h_1$  in  $\{1, \dots, H\}$  do
3   for  $h_2$  in  $\{0, \dots, h_1 - 1\}$  do
4      $\lfloor$  compute the set  $\Lambda_{h_1, h_2}$  of absolute minima of  $\varphi_{h_1, h_2}$ 
5    $L \leftarrow []$ 
6    $c \leftarrow +\infty$ 
7   for  $\mathbf{n}_H$  in  $\Lambda_{1,0} \times \Lambda_{2,0} \times \Lambda_{2,1} \times \dots \times \Lambda_{H,H-1}$  do
8      $\tau \leftarrow \text{SelfnestedTree}(\mathbf{n}_H)$ 
9      $d \leftarrow \delta(\tau, T)$ 
10    if  $d = c$  then
11       $\lfloor$  append  $\tau$  to  $L$ 
12    else if  $d < c$  then
13       $\lfloor$   $L \leftarrow [\tau]$ 
14       $\lfloor$   $c \leftarrow d$ 
15  return  $L, c$ 

```

Proposition 15 $\text{RFC}(T)$ may be computed in

$$O(\alpha(T) \times \#T \times \text{height}(T)^3 \times \text{deg}(T) \times \log_2(\text{deg}(T) + \text{height}(T))),$$

where $\alpha(T)$ is the cardinality of the product set $\Lambda_{1,0} \times \Lambda_{2,0} \times \Lambda_{2,1} \times \dots \times \Lambda_{\text{height}(T), \text{height}(T)-1}$.

Proof. The complexity of determining the sets Λ_{h_1, h_2} for all h_1, h_2 (first loop, lines 2–4 of Algorithm 2) is only $O(\text{height}(T)^2 \times \#T)$. The main step is thus to compute the $\alpha(T)$ edit distances involving a self-nested tree (line 9) that appear in the procedure and which complexity is given in Proposition 11. ■

5.3 Local pruning and second approximation algorithm

The preceding procedure only allows us to modify some subtrees and not to delete them. The second strategy that we consider exploits local pruning of T , i.e., consists in checking if deleting a subtree is a good operation to transform T into a self-nested tree. Indeed, it may be more efficient to prune subtrees with only few nodes rather than to transform them (see the example of Figure 12).

$F_1(T)$ denotes the forest of the (not necessarily direct) subtrees of height 1 appearing in T . In addition, $\theta_1(l)$ denotes the tree of height 1 with l leaves. As a consequence, one has

$$F_1(T) = \{\theta_1(l) \text{ with multiplicity } m(l) : l \in \mathcal{L}_T\},$$

for some finite set $\mathcal{L}_T \subset \mathbb{N}^*$. It should be noted that $F_1(T) = \emptyset$ if and only if T is composed of an isolated root. The locally pruned versions of T are the trees obtained by deleting all the leaves from subtrees of $F_1(T)$

in T . More precisely, for any integers l_1, \dots, l_k , $1 \leq k \leq \#\mathcal{L}_T$, let $s(l_1, \dots, l_k)$ be the script that deletes all the leaves from the $m(l_i)$ subtrees $\theta_1(l_i)$ appearing in T , for any $1 \leq i \leq k$. $S_T(k)$ denotes the set of such editing scripts. The forest of all the locally pruned versions of T is the set defined by

$$\text{LP}(T) = \bigcup_{k=1}^{\#\mathcal{L}_T} \{T^s : s \in S_T(k)\}.$$

The edit distance between T and one of its locally pruned versions $T^s \in \text{LP}(T)$ is the number of leaves that have been deleted, $\delta(T, T^s) = \#s$. The construction of this set is given in Algorithm 3. It should be noted that, in the worst case, all the subtrees of height 1 appearing in T are different. The cardinality of $\text{LP}(T)$ is thus of order $O(2^{\text{deg}(T)})$.

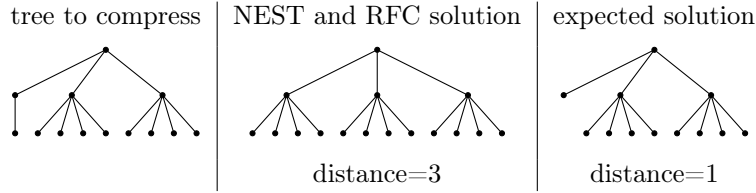


Figure 12: NEST and RFC algorithms add 3 vertices to the left subtree of height 1 in the tree to compress given in the left column. It would be more efficient to delete this subtree (right column).

Algorithm 3: Computation of the locally pruned versions $\text{LP}(t)$ of a tree t . Each element of the output is a couple (t^s, c) , where $t^s \in \text{LP}(t)$ and $c = \delta(t, t^s)$.

```

1 Function LocPruning( $t$ ):
   Data: a tree  $t$ 
   Result: forest  $\text{LP}(t)$  augmented with edit distances to  $t$ 
2    $L \leftarrow []$ 
3   for  $n$  in  $\{1, \dots, \#\mathcal{L}_t\}$  do
4     for  $(i_1, \dots, i_n)$  in  $\binom{\mathcal{L}_t}{n}$  do
5        $c \leftarrow 0$ 
6       for  $k$  in  $\{1, \dots, n\}$  do
7         delete all the leaves of all subtrees  $\theta_1(i_k)$  appearing in  $t$ 
8          $c \leftarrow c + i_k$ 
9       add the edited tree and the editing cost  $c$  to  $L$ 
10  return  $L$ 

```

Local pruning is motivated by the following idea: if two subtrees $T[x]$ and $T[y]$ are isomorphic, and if the deletion of $T[x]$ is a good operation to find a self-nested estimate of T , thus $T[y]$ should be also deleted. Our algorithm $\text{RFC}^+(T)$ (RFC improved by local pruning) exploits this scheme: approximate T by the self-nested trees $\text{RFC}(t)$ for any $t \in \text{LP}^\gamma(T)$, where $\text{LP}^\gamma(T)$ denotes the recursive application of local pruning to T γ times³ for any $\gamma \geq 0$. Of course it is not useful to investigate all the locally pruned versions of T , particularly whenever the cost of pruning exceeds the cost of matching the subforest, as highlighted in the pseudocode given in Algorithm 4. The complexity of this algorithm is presented in Proposition 16. Two typical examples are presented in Figures 13 and 14.

³It should be remarked that $\text{LP}^0(T)$ is only the singleton $\{T\}$.

Proposition 16 $\text{RFC}^+(T)$ may be computed in

$$O(\alpha(T) \times 2^{\deg(T)} \times \#T^2 \times \text{height}(T)^3 \times \deg(T) \times \log_2(\deg(T) + \text{height}(T))),$$

where $\alpha(T)$ is given by

$$\alpha(T) = \max_{\gamma \geq 0} \max_{t \in \text{LP}^\gamma(T)} \# [\Lambda_{1,0} \times \Lambda_{2,0} \times \Lambda_{2,1} \times \cdots \times \Lambda_{\text{height}(t), \text{height}(t)-1}].$$

Proof. In the worst case, the cardinality of $\text{LP}(T)$ is $O(2^{\deg(T)})$. As a consequence, one has to apply RFC algorithm on at most $O(\#T \times 2^{\deg(T)})$ recursive locally pruned versions of T . The result follows from Proposition 15. ■

Algorithm 4: Computation of self-nested estimates $\text{RFC}^+(T)$ of a tree T .

```

1 Function  $\text{RFC}^+(T)$ :
   Data: an unordered rooted tree  $T$ 
   Result: list of compressed versions of  $T$  augmented with their editing cost
2 T2C  $\leftarrow [T]$ 
3 res  $\leftarrow []$ 
4 cost  $\leftarrow +\infty$ 
5 while T2C  $\neq []$  do
6   newT2C  $\leftarrow []$ 
7   for  $\tau$  in T2C do
8      $L, c \leftarrow \text{RFC}(\tau)$ 
9     if cost =  $c$  then
10       $\lfloor$  extend res with  $L$  without redundancies
11    else if cost >  $c$  then
12       $\lfloor$  res  $\leftarrow L$ 
13       $\lfloor$  cost  $\leftarrow c$ 
14       $P \leftarrow \text{LocPruning}(\tau)$ 
15      for  $(\theta, \gamma)$  in  $P$  do
16         $\lfloor$  if  $\gamma \leq$  cost then
17           $\lfloor$   $\lfloor$  append  $\theta$  to newT2C without redundancies
18      T2C  $\leftarrow$  newT2C
19 return res, cost

```

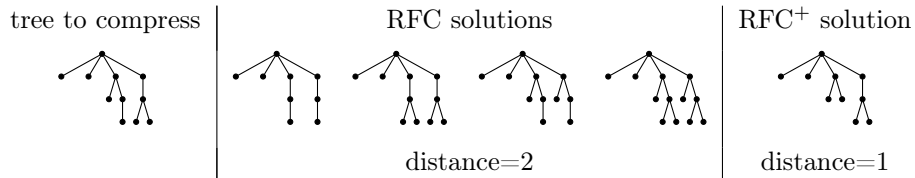


Figure 13: The tree to compress is given in the left column. RFC algorithm finds 4 solutions that are at distance 2 of the initial tree and presented in the middle column. Local pruning is useful here: there exists a better solution at distance 1 found by RFC^+ algorithm (right column).

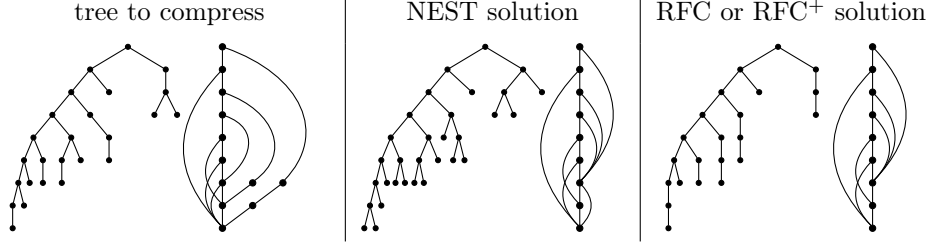


Figure 14: The tree to compress and its DAG reduction are given in the left column. NEST algorithm adds 6 vertices to obtain a self-nested tree (middle), whereas RFC and RFC⁺ only delete 3 vertices to obtain a self-nested tree (right).

6 Numerical illustration

We illustrate the behavior of both algorithms RFC and RFC⁺ on simulated binary trees. We compare these three algorithms by considering their compression rate ρ and their error rate e defined from

$$\rho = 1 - \frac{\#V(\mathcal{R}(C(T))) + \#E(\mathcal{R}(C(T)))}{\#V(T) + \#E(T)} \quad \text{and} \quad e = \frac{\delta(T, C(T))}{\#V(T)},$$

where T denotes the initial data to compress and $C(T)$ stands for its self-nested approximate version. It should be noted that e may be greater than 1. In addition, we highlight that the tree $\Theta_{H,m}$ introduced in Subsection 5.1 is approximated by $\mathcal{T}_{H,m}$ (see Figure 9 and Proposition 12) with the error rate

$$e = \frac{\delta(\Theta_{H,m}, \mathcal{T}_{H,m})}{\#\Theta_{H,m}} \simeq 1/3.$$

Indeed, the number of vertices of $\Theta_{H,m}$ is of order $3m^H/4$. This is not the maximal error rate but it gives an idea of the incompressible error that must be expected from any approximation algorithm.

6.1 A large binary tree

To assess our algorithms, we first construct a large tree T sampled from the uniform distribution among binary trees with 100 vertices. The error and compression rates of these algorithms are presented in Figure 15, while statistics on the numbers of vertices are given in Figure 16. The topological structure of T , together with the results provided by the different algorithms, are displayed in Figure 17.

First of all, one may remark that the compression rate of the classical DAG compression scheme is satisfactory on this example compared to the average rate $\bar{\rho}_{100} \simeq 38\%$ expected for trees with 100 vertices (see Section 1 and [5, Theorems 29 and 30]): the DAG reduction has 32 vertices and 55 edges (see Figure 17), thus $\rho \simeq 56\%$. NEST estimate is obtained by adding 107 vertices to the target. It is more than two times larger than the initial tree and thus is not visually similar to it (see Figures 16 and 17). RFC algorithm provides 4 self-nested estimates that all are at distance 43 to the target. With no additional information, none can be considered as a better compression than the others, except by choosing the solution which number of vertices is the closest to the target. However, it should be noted that this criterion seems somewhat arbitrary, in the sense that in particular applications other criteria, as the height or outdegree, could make more sense. The only solution provided by RFC⁺ algorithm is at distance 35 to the target, that is to say a substantial gain of around 19% compared to RFC. Nevertheless, all these trees are good visual estimates of the initial tree structure. The compression rates given in Figure 15 are very similar for the three approximation algorithms. The only reason why RFC⁺ has a better compression rate than its alternatives is that local pruning may shorten the height of the tree, and thus makes the number of vertices of the DAG reduction decrease.

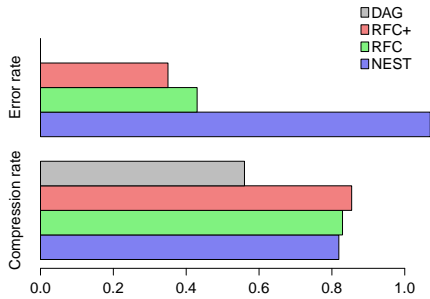


Figure 15: Error (top) and compression (bottom) rates for DAG compression applied to the initial data and its three self-nested estimates.

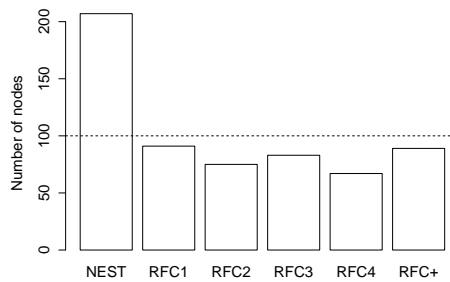


Figure 16: Sizes of trees obtained from NEST, RFC and RFC⁺ algorithms. The NEST solution has twice as many vertices as in the initial data (dashed line).

6.2 Small binary trees

Our simulations are performed on a stochastic model of binary trees. Given a tree t , we randomly choose a vertex of t with uniform distribution and add a child to it if it has 0 or 1 child. Beginning with the tree composed of an isolated root and recursively repeating this operation n times, one obtains a random binary tree with at most n vertices. Our dataset is composed of 500 trees simulated according to this model for different values of n between 20 and 50. Descriptive statistics of the dataset are given in Figure 18 and Table 2. Numerical results are provided in Figure 20.

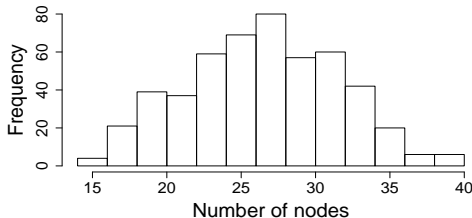


Figure 18: Histogram of simulated trees.

number of vertices			
mean	min	max	std
26.9	15	40	5.1

Table 2: Statistics of simulated trees.

The three algorithms are equivalent in terms of compression rates, which was expected for the same reason as in Subsection 6.1. The key parameter is thus the error rate that is much better for RFC and RFC⁺ algorithms than for NEST procedure. On average, one obtains a substantial gain of around 20% for both our algorithms (see Figure 19). Local pruning is useful in RFC 24.2% of the time and makes the error decrease of only 1.8% (see Table 3). However, when local pruning is not useless, the error is improved of 7.3%, which is not negligible. Despite the fact that small binary trees framework is the most favourable to NEST solution, our compression procedures perform better than this algorithm. As a conclusion, the RFC and RFC⁺ algorithms have a higher time-complexity than NEST that is in $O(\text{height}(T)^2 \times \text{deg}(T))$ time, but provide very much improved compression properties.

$e_{\text{RFC}} - e_{\text{RFC}^+} > 0$		$e_{\text{RFC}} - e_{\text{RFC}^+}$
freq.	mean	mean
24.2%	7.3%	1.8%

Table 3: Comparison of error rates e_{RFC} and e_{RFC^+} .

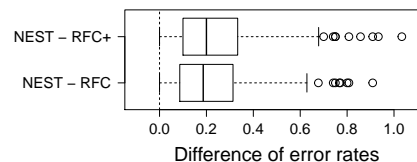


Figure 19: Comparison of error rates: $e_{\text{NEST}} - e_{\text{RFC}^+}$ (top) and $e_{\text{NEST}} - e_{\text{RFC}}$ (bottom).

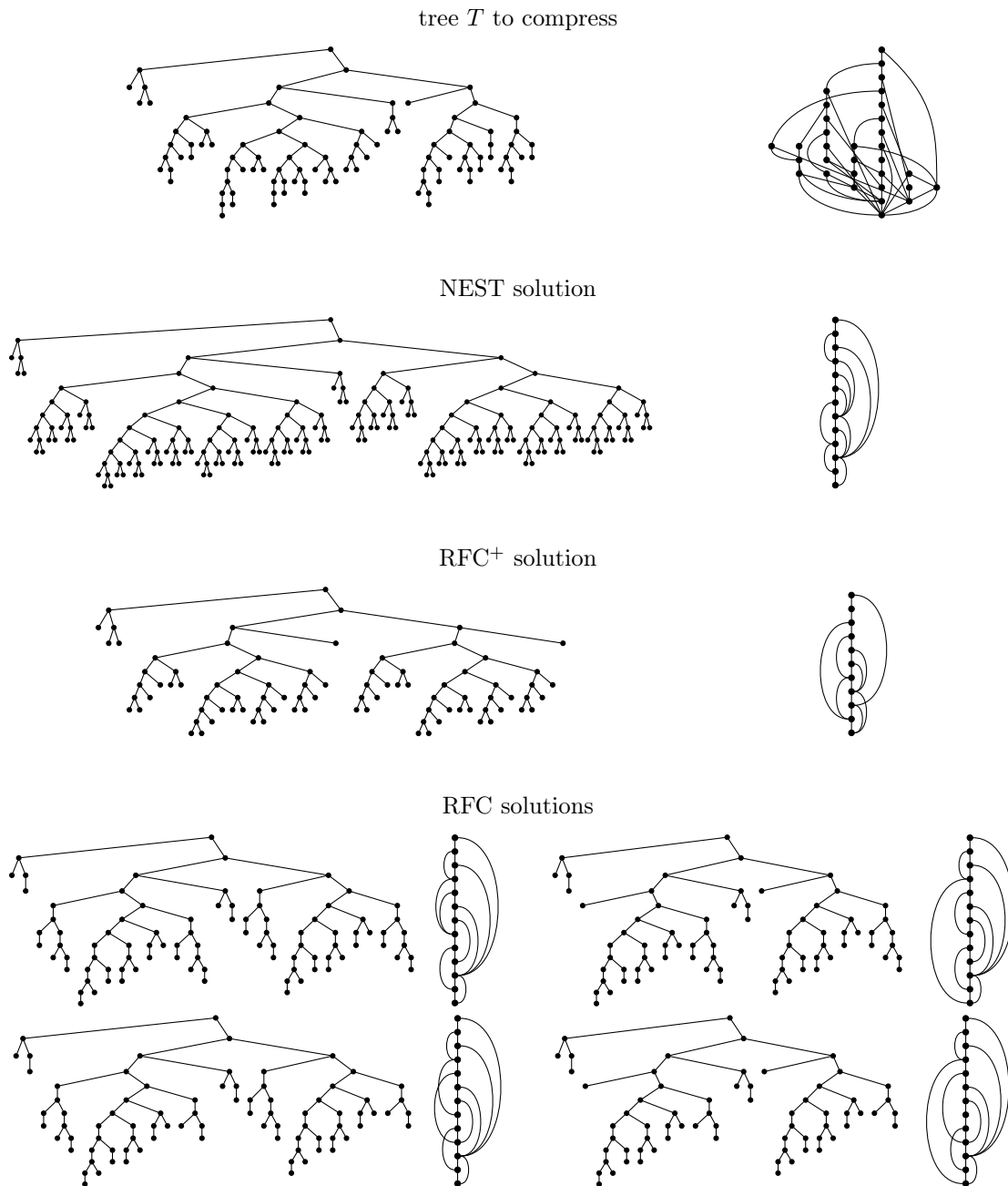


Figure 17: The initial data to compress, its DAG version and the solutions provided by the different lossy compression algorithms presented in this paper. One may observe that the RFC+ solution is visually very close to the initial tree, which is confirmed by the error rate of 35%. All the solutions provided by the RFC algorithm are at distance 43 to the tree to compress and are also good visual self-nested estimates. The NEST solution has too many vertices for looking like the initial data: the algorithm added 107 vertices to obtain a self-nested tree.

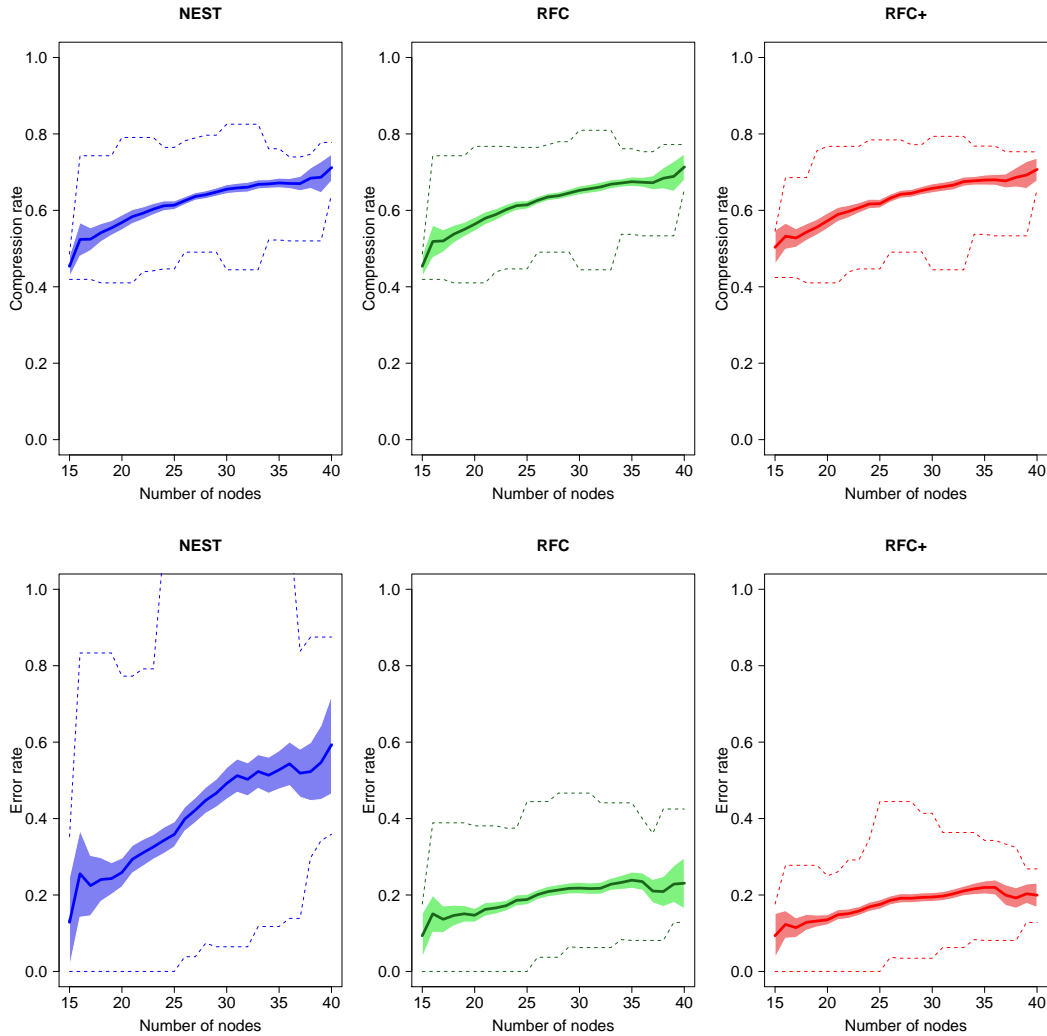


Figure 20: Compression rates (top) and error rates (bottom) for NEST (blue, left), RFC (green, middle) and RFC⁺ (red, right) algorithms estimated from 500 simulations of random binary trees: average rates (bold lines), 95% confidence intervals (colored areas), minimum and maximum rates (dashed lines).

References

- [1] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1974.
- [2] BEN-NAOUM, F., AND GODIN, C. Algorithmic height compression of unordered trees. *Journal of Theoretical Biology* 389 (2016), 237 – 252.
- [3] BILLE, P. A survey on tree edit distance and related problems. *Theoretical Computer Science* 337, 1-3 (2005), 217 – 239.
- [4] BILLE, P., GØRTZ, I. L., LANDAU, G. M., AND WEIMANN, O. Tree compression with top trees. *Information and Computation* 243 (2015), 166 – 177. 40th International Colloquium on Automata, Languages and Programming (ICALP 2013).

- [5] BOUSQUET-MÉLOU, M., LOHREY, M., MANETH, S., AND NOETH, E. XML compression via directed acyclic graphs. *Theory of Computing Systems* (2014), 1–50.
- [6] BUNEMAN, P., GROHE, M., AND KOCH, C. Path queries on compressed XML. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29* (2003), VLDB '03, VLDB Endowment, pp. 141–152.
- [7] BUSATTO, G., LOHREY, M., AND MANETH, S. Efficient memory representation of xml document trees. *Inf. Syst.* 33, 4-5 (June 2008), 456–474.
- [8] CHAWATHE, S. S. Comparing hierarchical data in external memory. In *VLDB* (Edinburgh, Scotland, sep 1999), pp. 90–101.
- [9] COSTELLO, J. On the number of points in regular discrete simplex (corresp.). *IEEE Transactions on Information Theory* 17, 2 (Mar 1971), 211–212.
- [10] DOWNEY, P. J., SETHI, R., AND TARJAN, R. E. Variations on the common subexpression problem. *J. ACM* 27, 4 (Oct. 1980), 758–771.
- [11] FERRARO, P., AND GODIN, C. An edit distance between quotiented trees. *Algorithmica* 36 (2003), 1–39.
- [12] FLAJOLET, P., AND SEDGEWICK, R. *Analytic Combinatorics*, 1st ed. Cambridge University Press, New York, USA, 2009.
- [13] FRICK, M., GROHE, M., AND KOCH, C. Query evaluation on compressed trees. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on* (2003), IEEE, pp. 188–197.
- [14] GODIN, C., AND FERRARO, P. Quantifying the degree of self-nestedness of trees. Application to the structural analysis of plants. *IEEE TCBB* 7, 4 (Oct. 2010), 688–703.
- [15] KNUTH, D. E. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.
- [16] KOLBE, D., ZHU, Q., AND PRAMANIK, S. On k-nearest neighbor searching in non-ordered discrete data spaces. In *IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007.* (April 2007), pp. 426–435.
- [17] LOHREY, M., AND MANETH, S. The complexity of tree automata and xpath on grammar-compressed trees. *Theor. Comput. Sci.* 363, 2 (Oct. 2006), 196–210.
- [18] LOHREY, M., MANETH, S., AND MENNICKE, R. Tree structure compression with repair. In *Data Compression Conference (DCC), 2011* (March 2011), pp. 353–362.
- [19] SAKR, S. XML compression techniques: A survey and comparison. *Journal of Computer and System Sciences* 75, 5 (2009), 303 – 322.
- [20] TANAKA, E., AND TANAKA, K. The tree-to-tree editing problem. *International Journal of Pattern Recognition and Artificial Intelligence* 02, 02 (1988), 221–240.
- [21] TARJAN, R. E. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.
- [22] ZHANG, K. A constrained edit distance between unordered labeled trees. *Algorithmica* 15, 3 (1996), 205–222.

A Proofs of the combinatorics results

A.1 Proof of Proposition 3

Using the notation of Subsection 2.2, a self-nested tree of height h is represented by a linear DAG with $h + 1$ vertices numbered from 0 to h (top) in such a way that there exists a path $h \rightarrow \dots \rightarrow 1 \rightarrow 0$. One recalls that this graph is augmented with integer-valued label $n_{i,j}$ on edge $i \rightarrow j$ for any $i > j$ with the constraint $n_{i,i-1} > 0$. In this context, the outdegree of a self-nested tree is less than m if and only if, for any i ,

$$\sum_{j=0}^{i-1} n_{i,j} \leq m.$$

We propose to write $n_{i,i-1} = 1 + n'_{i,i-1}$ and, for $j \leq i - 2$, $n'_{i,j} = n_{i,j}$. As a consequence, all the labels are parametrized by the $n'_{i,j}$'s which satisfy,

$$\forall 1 \leq i \leq h, \quad \forall 0 \leq j \leq i - 1, \quad n'_{i,j} \geq 0 \quad \text{and} \quad \sum_{j=0}^{i-1} n'_{i,j} \leq m - 1.$$

Thus, the number of self-nested trees of height h is obtained as

$$\#\mathbb{T}_{=h, \leq m}^{sn} = \prod_{i=1}^h \#\left\{ n'_{i,j} : n'_{i,j} \geq 0 \quad \text{and} \quad \sum_{j=0}^{i-1} n'_{i,j} \leq m - 1 \right\}. \quad (1)$$

Furthermore, the set $\{n'_{i,j} : n'_{i,j} \geq 0 \quad \text{and} \quad \sum_{j=0}^{i-1} n'_{i,j} \leq m - 1\}$ is only the regular discrete simplex of dimension i having m points on an edge. The cardinality of this set has been studied by Costello in [9]. Thus, by virtue of [9, Theorem 2], one has

$$\#\left\{ n'_{i,j} : n'_{i,j} \geq 0 \quad \text{and} \quad \sum_{j=0}^{i-1} n'_{i,j} \leq m - 1 \right\} = \binom{m + i - 1}{i}. \quad (2)$$

Together with

$$\#\mathbb{T}_{\leq H, \leq m}^{sn} = \sum_{h=1}^H \#\mathbb{T}_{=h, \leq m}^{sn},$$

this yields the expected result.

A.2 Proof of Proposition 4

Roughly speaking, an unordered tree with maximal height H and maximal outdegree m may be obtained by adding at most m trees of height less than $H - 1$ to an isolated root. More precisely, one has to choose m elements with repetitions among the set $\mathbb{T}_{\leq H-1, \leq m} \cup \{\bullet\} \cup \{\emptyset\}$ and add them to the list of direct subtrees (initially empty) of a same vertex. It should be noted that no subtree is added when \emptyset is picked.

One obtains either an isolated root (if and only if one draws m times the symbol \emptyset), or a tree with maximal height H . As a consequence, one has the formula,

$$\#\left[\mathbb{T}_{\leq H, \leq m} \cup \{\bullet\}\right] = \binom{\#\left[\mathbb{T}_{\leq H-1, \leq m} \cup \{\bullet\} \cup \{\emptyset\}\right] + m - 1}{m},$$

which shows the result.

A.3 Proof of Corollary 6

In the proof of Proposition 3, we have already shown that

$$\#\mathbb{T}_{=h, \leq m}^{sn} = \prod_{i=1}^h \binom{m+h-i}{h-i+1},$$

see (1) and (2). Substituting the binomial coefficients by their value, we get

$$\begin{aligned} \#\mathbb{T}_{=h, \leq m}^{sn} &= \Gamma(m)^{-h} \prod_{i=1}^h \frac{\Gamma(m+h-i+1)}{\Gamma(h-i+2)} \\ &= \Gamma(m)^{-h} \prod_{i=1}^h (h-i+2) \times (h-i+3) \times \cdots \times (h-i+m), \end{aligned}$$

where Γ denotes the Euler function such that $\Gamma(n+1) = n!$ for any integer n . As a consequence,

$$\begin{aligned} \log \#\mathbb{T}_{=h, \leq m}^{sn} &= -h \log \Gamma(m) + \sum_{\substack{1 \leq i \leq h \\ 2 \leq k \leq m}} \log(h-i+k) \\ &= -h \log \Gamma(m) + \sum_{\substack{0 \leq j \leq h-1 \\ 2 \leq k \leq m}} \log(j+k), \end{aligned} \quad (3)$$

by substituting $h-i$ by j . First, according to Stirling's approximation, we have

$$-h \log \Gamma(m) \sim -hm \log m. \quad (4)$$

Now, we focus on the second term. In order to simplify, we are looking for an equivalent of the same double sum but indexed on $1 \leq j \leq h$ and $1 \leq k \leq m$. We have

$$\begin{aligned} \sum_{j=1}^h \sum_{k=1}^m \log(j+k) &= \sum_{j=1}^h \sum_{k=1}^m \int_1^{j+k} \frac{dx}{x} \\ &= \sum_{j=1}^h \left[\sum_{l=0}^{m-1} (m-l) \int_{j+l}^{j+l+1} \frac{dx}{x} + \int_1^j \frac{dx}{x} \right] \\ &= \sum_{l=0}^{m-1} (m-l) \left[\sum_{j=1}^h \int_{j+l}^{j+l+1} \frac{dx}{x} + \log j \right] \\ &= \sum_{l=0}^{m-1} (m-l) \int_{l+1}^{l+h+1} \frac{dx}{x} + m \sum_{j=1}^h \log j \\ &= \sum_{l=0}^{m-1} (m-l) \log \left(1 + \frac{h}{l+1} \right) + m \sum_{j=1}^h \log j. \end{aligned} \quad (5)$$

As usually, we find an equivalent of this term by using an integral comparison test. We establish by a conscientious calculus that

$$\sum_{l=0}^{m-1} (m-l) \log \left(1 + \frac{h}{l+1} \right) + m \sum_{j=1}^h \log j \sim \frac{(m+h)^2}{2} \log(m+h) - \frac{h^2}{2} \log h - \frac{m^2}{2} \log m + R(m, h), \quad (6)$$

where the rest $R(m, h)$ is neglectable with respect to the other terms and to $hm \log m$. Let us remark that the expression of the equivalent is symmetric in h and m as expected. Finally, (3), (4), (5) and (6) show the result.

A.4 Proof of Corollary 7

The proof is based on the classical bounds on binomial coefficients,

$$\left(\frac{n \times e}{k}\right)^k \geq \binom{n}{k} \geq \left(\frac{n}{k}\right)^k.$$

Using Proposition 4, we have $u_1(m) = \binom{1+m}{m} = m + 1$ and

$$u_2(m) = \binom{u_1(m) + m}{m} = \binom{2m + 1}{m} \geq \left(\frac{2m + 1}{m}\right)^m = \left(2 + \frac{1}{m}\right)^m.$$

The lower bound is obtained by induction on $h > 2$: assuming that

$$u_h(m) \geq \frac{\left(2 + \frac{1}{m}\right)^{m^{h-1}}}{m^{\frac{m^{h-1}-1}{m-1}}},$$

we have

$$u_{h+1}(m) = \binom{u_h(m) + m}{m} \geq \left(\frac{u_h(m)}{m} + 1\right)^m \geq \left(\frac{u_h(m)}{m}\right)^m \geq \left(\frac{\left(2 + \frac{1}{m}\right)^{m^{h-1}}}{m^{\frac{m^{h-1}-1}{m-1}}}\right)^m$$

by the induction hypothesis. Using

$$m \left(\frac{m^{h-1} - 1}{m - 1}\right) = \frac{m^h - 1}{m - 1} - 1,$$

we obtain

$$u_{h+1}(m) \geq \frac{\left(2 + \frac{1}{m}\right)^{m^h}}{m^{\frac{m^h-1}{m-1}-1}}.$$

Moreover,

$$u_2(m) = \binom{2m + 1}{m} \leq \left(\frac{2m + 1}{m} e\right)^m \leq (3e)^m.$$

The upper bound is obtained by induction on $h \geq 2$: assuming that

$$u_h(m) \leq 3^{m^{h-1}} e^{\frac{m^{h-1}-1}{m-1}},$$

we obtain

$$u_{h+1}(m) = \binom{u_h(m) + m}{m} \leq \left(\left(\frac{u_h(m)}{m} + 1\right) e\right)^m \leq \left(\left(\frac{3^{m^{h-1}} e^{\frac{m^{h-1}-1}{m-1}-1}}{m} + 1\right) e\right)^m$$

by the induction hypothesis. Using the inequality

$$\frac{k^x}{x} + 1 \leq k^x,$$



satisfied whenever k and x are both greater than the critical value 1.693... obtained by numerical methods, we obtain

$$u_{h+1}(m) \leq \left(3^{m^{h-1}} e^{\frac{m^{h-1}-1}{m-1}-1} e\right)^m = 3^{m^h} e^{\frac{m^{h+1}-1}{m-1}-1}.$$

This shows the expected result.

B Proof of Proposition 12

The main difficulty is to establish that the worst case is given by $\Theta_{H,m}$. We propose to begin with trees of height 2, and we shall state in two steps the expected result.

First of all, let us remark that the DAG of any tree of $\mathbb{T}_{2,\leq m}$ is of the form . Nevertheless, leaves attached to the root do not impact the self-nestedness of the tree and deletes some degrees of freedom in our research of the worst case. As a consequence, we only consider DAGs of the form  with M intermediate vertices (that is to say M different subtrees of height 1) labeled from I_1 to I_M , $M \leq m$. Of course, $M = 1$ ensures that the corresponding tree is self-nested: we exclude this case. Let p_k (l_k , respectively) denote the number of appearances (the number of leaves, respectively) of I_k , for $1 \leq k \leq M$.

We shall investigate the worst case for a given value of M . First, it should be noted that if an operation is optimal for an equivalence class I_k , it is also optimal for all the subtrees of this class. In addition, there are only two possible scripts to transform I_k : either one deletes all the leaves of I_k (with a cost $p_k l_k$), or one adds or deletes some leaves to transform I_k into a given subtree of height 1 with, say, x leaves (with a cost $p_k |l_k - x|$). As a consequence, the total editing cost (to transform the initial tree into a self-nested tree in which trees of height 1 have x leaves) is given by

$$C_2 = \sum_{k \in A} p_k l_k + \sum_{k \notin A} p_k |l_k - x|,$$

where A denotes the set of indices k for which one deletes all the leaves of I_k .

The worst case has the maximum entropy and thus a uniform repartition of its leaves in the tree. For the sake of clarity, one assumes in the sequel that m is even and M divides m . The explicit solution of the problem is thus $p_k = \frac{m}{M}$, $l_k = \frac{km}{M}$, $x = \frac{m}{2}$ and $A = \emptyset$. The remarkable fact is that the corresponding cost is given by

$$\begin{aligned} C_2 &= \frac{m}{M} \sum_{k=1}^M \left| \frac{m}{2} - \frac{km}{M} \right| \\ &= \frac{2m}{M} \sum_{k=1}^{\frac{M}{2}-1} \left(\frac{m}{2} - \frac{km}{M} \right) \\ &= \frac{m^2}{4}. \end{aligned}$$

This means that the worst case may be obtained from any value of M whenever it divides m . Actually, the case M does not divide m leads to a worst case better than when M divides m . One concludes that one of the worst cases is obtained from $M = 2$, $p_1 = p_2 = \frac{m}{2}$, $l_1 = \frac{m}{2}$, $l_2 = m$ and $C_2 = \frac{m^2}{4}$. When m is an odd integer, one observes the same phenomenon: the worst case is obtained from $M = 2$, $p_1 = \lceil \frac{m}{2} \rceil$, $p_2 = \lfloor \frac{m}{2} \rfloor$, $l_1 = \lfloor \frac{m}{2} \rfloor$, $l_2 = m$ and $C_2 = \lfloor \frac{m}{2} \rfloor \times \lceil \frac{m}{2} \rceil$. This yields the expected result for any integer m .

We shall use the preceding idea to show the result for any height H . Among trees of height at most H , it is quite obvious that the worst case appears in trees of height H . We assume that there are M different patterns I_1, \dots, I_M appearing p_1, \dots, p_M times under the root. The cost of editing operations (adding or deleting leaves) at distance h to the root is in the worst case $p_k \times m^{h-1}$. As a consequence, at least for m large enough, $\text{height}(I_k) = H - 1$ and the only difference with the other patterns is on the fringe: all the vertices of I_k have m children except vertices at height $H - 2$ that have l_k leaves. If A denotes the set of indices k for which one deletes all the leaves of I_k , the editing cost to transform the tree into the self-nested tree in which subtrees of height 1 have x leaves is given by

$$C_H = m^{H-2} \left[\sum_{k \in A} p_k l_k + \sum_{k \notin A} p_k |l_k - x| \right].$$

In light of the previous reasoning, this states the expected result.