Approximation of trees by self-nested trees

Romain Azaïs, Jean-Baptiste Durand, Christophe Godin

Approximation of trees by self-nested trees

Romain Azaïs[*] Jean-Baptiste Durand[†] Christophe Godin[*]

Abstract

The class of self-nested trees presents remarkable compression properties because of the systematic repetition of subtrees in their structure. In this paper, we provide a better combinatorial characterization of this specific family of trees. In particular, we show from both theoretical and practical viewpoints that complex queries can be quickly answered in self-nested trees compared to general trees. We also present an approximation algorithm of a tree by a self-nested one that can be used in fast prediction of edit distance between two trees.

1 Introduction

Trees form an expanded family of combinatorial objects that offers a wide range of application fields, from plant modeling to XML files analysis through study of RNA secondary structure. Complex queries on tree structures (e.g., computation of edit distance, finding common substructures, compression) are required to handle these models. A critical question is to control the complexity of the algorithms implemented to solve these queries. One way to address this issue is to approximate the original trees by simplified structures that achieve good algorithmic properties. One can expect good algorithmic properties from structures that present a high level of redundancy in their substructures. Indeed, one can take account these repetitions to avoid redundant computations on the whole structure.

Searching redundancies in the tree often allows to design efficient compression methods. As it is explained in [3], one often considers the following two types of repeated substructures: subtree repeat (used in DAG compression [4, 5, 13, 14]) and tree pattern repeat (exploited in tree grammars [6, 16, 17] and top tree compression [3]). A survey on this topic may be found in [18] 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 displaying 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], that share the same time-complexity in $O(n^2 \times d \times \log(d))$, where n is the number of vertices of the tree and d denotes its outdegree. From now on, we limit ourselves to unordered trees.

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 τ has exactly $H+1$ vertices, where H denotes the height of τ, which is the minimal number of vertices that may be reached. This family of trees has been introduced in [15] under the name of nested trees as an interesting class of trees for which the subtree isomorphism problem is in NC^2. Later, they have been called self-nested trees [14, Definition 7] to insist on their recursive structure and their proximity to the notion of self-similarity.

In this article, we prove the algorithmic efficiency of self-nested trees through different questions (compression, evaluation of recursive functions, evaluation of edit distance) and study their combinatorics. In particular, we establish that self-nested trees are roughly exponentially less frequent than general trees. This combinatorics can be an asset in exhaustive search problems. Nevertheless, this result also says that one can not always take advantage of the remarkable algorithmic properties of self-nested trees when working with general trees. Consequently, our aim is to investigate how general unordered trees can be approximated by simplified trees in the class

[*]Laboratoire Reproduction et Développement des Plantes, Univ Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRA, Inria, F-69342, Lyon, France.
[†]Laboratoire Jean Kuntzmann, MISTIS, INRIA Grenoble – Rhône-Alpes, Saint Ismier, France.

of self-nested trees from both theoretical and numerical perspectives. Our objective is to take advantage of the aforementioned qualities of self-nested trees even for a tree that is not self-nested. In particular, we show that our approximation algorithm can be used to very rapidly predict the edit distance between two trees, which is a usual but costly operation for comparing tree data in computational biology for instance [19].

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. Algorithmic efficiency of self-nested trees is investigated in Section 3. Combinatorial properties of self-nested trees are presented in Section 4. Our approximation algorithm is developed in Section 5. Section 6 is dedicated to an application to fast prediction of edit distance. All the proofs have been deferred to the supplementary file.

Note on numerical results We ran all the simulations of the paper in `Python3` on a Macbook Pro laptop running OSX High Sierra, with 2.9 GHz Intel Core i7 processors and 16 GB of RAM.

2 Preliminaries

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

2.1 Unordered rooted trees

A rooted tree $\tau = (V, E)$ is a connected digraph containing no cycle and such that there exists a unique vertex called root(τ) which has no parent. Any vertex different from the root has exactly one parent. For any vertex v of τ, children(v) denotes the set of vertices that have v as parent. The leaves of τ are all the vertices without children. The height of a vertex v may be recursively defined as height(v) = 0 if v is a leaf of τ and

$$\text{height}(v) = 1 + \max_{w\in\text{children}(v)} \text{height}(w)$$

otherwise. The height of the tree τ is defined as the height of its root, height(τ) = height(root(τ)). The outdegree deg(τ) of τ is the maximal branching factor that can be found in τ, that is

$$\deg(\tau) = \max_{v\in\tau} \#\,\text{children}(v),$$

where $\#A$ denotes the cardinality of the set A. With a slight abuse of notation, $\#\tau$ denotes the number of vertices of τ. A subtree $\tau[v] = (V[v], E[v])$ rooted in v is the connected subgraph of τ such that $V[v]$ is the set of the descendants of v in τ and $E[v]$ is defined as

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

In 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 $\tau_1 = (V_1, E_1)$ and $\tau_2 = (V_2, E_2)$ be two rooted trees. A one-to-one correspondence $\varphi : V_1 \to V_2$ is called a tree isomorphism if, for any edge $(v, w) \in E_1$, $(\varphi(v), \varphi(w)) \in E_2$. 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 $O(n)$ [1, Example 3.2 and Theorem 3.3]. The existence of a tree isomorphism defines an equivalence relation $\equiv$ 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. We refer the reader to [12, I.5.2. Non-plane trees] for more details on this combinatorial class. From now on, all the trees are unordered rooted trees.

Simulation of random trees We describe here the algorithm that we used in this paper to generate random trees of given size. From a tree with $n-1$ vertices, we construct a tree of size n by adding a child to a randomly chosen vertex (uniform distribution). We point out that the position of the new child is not significant since trees are unordered. Starting from the tree composed of a unique vertex, we repeat the procedure $n-1$ times to obtain a random tree of size n.

2.2 Tree reduction

Let us now consider the equivalence relation $\equiv$ on the set of the subtrees of a tree $\tau = (V, E)$. We consider the quotient graph $Q = (V_\equiv, E_\equiv)$ 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 the root of C_2 is a child of the root of C_1 (modulo isomorphism). In light of [14, Proposition 1], the graph Q is a Directed Acyclic Graph (DAG), that is a connected digraph without path from any vertex x to itself. Let (C_1, C_2) be an edge of the DAG Q. We define $N(C_1, C_2)$ as the number of occurrences of a tree of C_2 as child of root(C_1). The tree reduction $\mathscr{R}(\tau)$ is defined as the quotient graph Q augmented with labels $N(C_1, C_2)$ on its edges (see [14, Definition 3 (Reduction of a tree)] for more details). Intuitively, the labeled graph $\mathscr{R}(\tau)$ represents the original tree τ without its structural redundancies. Illustrations are presented in Figures 1 and 2. It should be noticed that a tree can be exactly reconstructed from its DAG reduction [14, Proposition 4], i.e., the application $\mathscr{R}$ is a one-to-one correspondence from unordered trees into DAG reductions space, which inverse is denoted $\mathscr{R}^{-1}$.

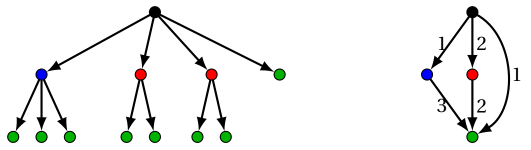


Figure 1: An unordered tree τ (left) and its reduction $\mathscr{R}(\tau)$ (right). In the tree, roots of isomorphic subtrees are colored identically. In the quotient graph, vertices are equivalence classes colored according to the class of isomorphic subtrees of τ that they represent.

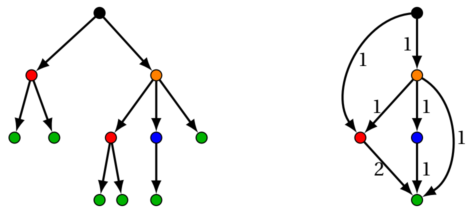


Figure 2: Another unordered tree τ (left) and its reduction $\mathscr{R}(\tau)$ (right).

Notation of DAGs The vertices of the quotient graph can be sorted by height, i.e., numbered as,

$$V_\equiv = \{(h, i) \,:\, 0 \le h \le \text{height}(\tau),\ 1 \le i \le M_h\},$$

where M_h denotes the number of vertices of height h appearing in Q. In other words, (h, i) is the vertex representing the i^{th} equivalence class of height h appearing in τ. We highlight that the order chosen to number the equivalence classes of a same height is not significant. The structure of $\mathscr{R}(\tau)$ is thus fully characterized by the array

$$\left[N((h_1, i), (h_2, j))\right]_{h_2, h_1, i, j},$$

with $0 \le h_2 < h_1 \le \text{height}(\tau)$, $1 \le i \le M_{h_1}$ and $1 \le j \le M_{h_2}$, and where

$$N((h_1, i), (h_2, j)) = 0 \quad \Leftrightarrow \quad ((h_1, i), (h_2, j)) \notin E_\equiv.$$

If $(h, i) \in V_\equiv$ and $D = \mathscr{R}(\tau)$, $D[(h, i)]$ denotes the sub-DAG rooted at (h, i). By construction of D, $D[(h, i)]$ is the reduction of a unique (up to an isomorphism) subtree of height h appearing in τ, which is denoted by $\mathscr{R}^{-1}(D[(h, i)])$.

2.3 Self-nested trees

A tree τ is called self-nested [14, III. Self-nested trees] if, for any pair of vertices v and w such that $\text{height}(\tau[v]) = \text{height}(\tau[w])$, $\tau[v]$ and $\tau[w]$ are isomorphic. It should be noted that this characterization is equivalent to the following statement: for any pair of vertices v and w such that $\text{height}(\tau[v]) \leq \text{height}(\tau[w])$, $\tau[v]$ is (isomorphic to) a subtree of $\tau[w]$. By definition, self-nested trees achieve the maximal presence of redundancies in their structure. Self-nested trees are tightly connected with linear DAGs, i.e., DAGs containing at least one path that goes through all their vertices.

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

We point out that a linear DAG reduction means that the compression is optimal, at least in terms of number of vertices. Indeed, the number of vertices of $\mathscr{R}(\tau)$ is always greater than $\text{height}(\tau) + 1$ by construction: there is at least one tree of height h appearing in τ for $0 \leq h \leq \text{height}(\tau)$. In addition, the inequality is saturated if and only if the DAG is linear. Proposition 1 evidences that DAG compression is optimal for self-nested trees. This is because (i) DAG compression removes repetitions of subtrees and (ii) self-nested trees present systematic redundancies in their subtrees. Two examples are displayed in Figures 3 and 4 for illustration purposes.

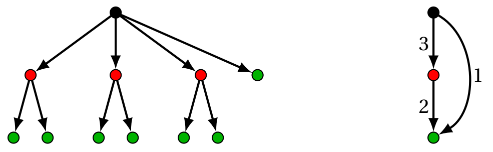


Figure 3: A self-nested tree τ (left) and its linear reduction $\mathscr{R}(\tau)$ (right). In the tree, all the subtrees of the same height are isomorphic and their roots are colored identically. The quotient graph is a linear DAG in which each vertex represents all the subtrees with the same height.

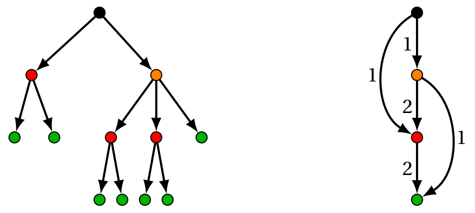


Figure 4: Another self-nested tree τ (left) and its linear reduction $\mathscr{R}(\tau)$ (right).

Notation of linear DAGs The vertices of a linear DAG $D = (V_\equiv, E_\equiv, N)$ of height H can be sorted by height and thus numbered as $V_\equiv = \{0, \dots, H\}$, i.e., h denotes the index of the unique equivalence class of height h appearing in the associated self-nested tree. The structure of D is fully characterized by the array

$$[N(h_1, h_2)]_{0 \leq h_2 < h_1 \leq H},$$

where $N(h_1, h_2) = 0$ if and only if $(h_1, h_2) \notin E_\equiv$. One can also notice that $N(h_1, h_1 - 1) \geq 1$.

Simulation of random self-nested trees As stated in Proposition 1, there is a one-to-one correspondence between self-nested trees and linear DAGs. To simulate a random self-nested tree, we generate a random linear DAG as follows. Given height H and maximal outdegree d, all the coefficients $N(h_1, h_2)$, $0 \leq h_2 < h_1 \leq H$, are chosen under the uniform distribution with constraints

$$\sum_{h_2=0}^{h_1-1} N(h_1, h_2) \leq d$$

and $N(h_1, h_1 - 1) \geq 1$, by rejection sampling. In this paper, we aim to compare the algorithmic efficiency of self-nested trees with respect to general trees. We have generated the datasets used in the numerical experiments as follows: first, generate a random general tree of given size, and then generate a random self-nested tree with the same height and outdegree.

3 Efficiency of self-nested trees

As aforementioned in Subsection 2.3, self-nested trees present the highest level of redundancy in their subtrees. Thus, one can expect good algorithmic properties from them. In this section we prove their computational efficiency through three questions: compression, evaluation of bottom-up functions, evaluation of edit distance.

3.1 Compression rates

Proposition 1 proves that self-nested trees achieve optimal compression rates among trees of the same height whatever their number of vertices. However, this statement does not take into account the number of edges, neither the presence of labels on the edges of the DAG reduction. We have estimated the average disk size being occupied by a tree and its DAG reduction from the simulation of 40 000 random trees (a half being self-nested). The data have been stored by using the `pickle` module in `Python3`. The results are presented in Figure 5 and Table 1. The simulations show that the compression rate (defined as the ratio of the compressed size over the uncompressed size) is around 10% for a random tree regardless of its size, while it is approximately 0.3% for a self-nested tree.

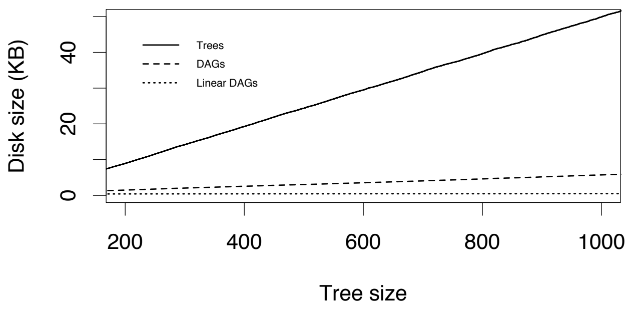


Figure 5: Estimation of the disk space occupied by a tree and its DAG reduction for trees and self-nested trees.

	Disk size (KB)
Trees	$5.044 \times 10^{-2} \# \tau$
DAGs	$5.433 \times 10^{-3} \# \tau$
Linear DAGs	$1.913 \times 10^{-4} \# \tau$

Table 1: Estimation of the slope of the three curves of Figure 5. For example, the disk size being occupied by a tree of size 100 is 5.044 KB on average, while it is approximately 0.02 KB for the DAG reduction of a self-nested tree of the same size.

3.2 Bottom-up recursive functions

Given a tree $\tau = (V, E)$, we consider bottom-up recursive functions $f : V \to \mathcal{X}$, defined by

$$f(v) = \begin{cases} \Phi\big((f(c))_{c \in \text{children}(v)} \big) & \text{if children}(v) \neq \emptyset \\ f_0 & \text{else,} \end{cases}$$

where f_0 is the value of f on the leaves of τ and

$$\Phi : \bigcup_{n\geq 1} \mathscr{X}^n \to \mathscr{X}$$

is invariant under permutation of arguments. The value of $f(\tau)$ is defined as $f(\tau) = f(\text{root}(\tau))$. Bottom-up recursive functions play a central role in the study of trees. For instance, the number of vertices, the number of leaves, the height and the Strahler number (that measures the branching complexity) are useful bottom-up recursive functions.

Proposition 2. *$f(\tau)$ can be computed in:*

- *$O(\#\tau)$-time from the tree τ;*
- *$O(\#D\,\text{deg}(\tau))$-time from the DAG reduction $D = \mathscr{R}(\tau)$.*

Consequently, if τ is self-nested, $f(\tau)$ can be computed in $O(\text{height}(\tau)\,\text{deg}(\tau))$-time.

For example, the number of vertices of a tree τ can be computed from its DAG reduction through the recursive formula

$$\#\mathscr{R}^{-1}(D[(h_1,i)]) = 1 + \sum_{h_2=0}^{h_1-1}\sum_{j=1}^{M_{h_2}} N((h_1,i),(h_2,j))\,\#\mathscr{R}^{-1}(D[(h_2,j]).$$

If τ is self-nested, its DAG reduction is linear and the recursion becomes

$$\#\mathscr{R}^{-1}(D[h_1]) = 1 + \sum_{h_2=0}^{h_1-1} N(h_1,h_2)\,\#\mathscr{R}^{-1}(D[h_2]), \tag{1}$$

where the number of positive terms in the sum is bounded by $\text{deg}(\tau)$. The complexities stated in Proposition 2 have been illustrated through the numerical simulation of 40 000 random trees (see Figure 6 and Table 2). The simulations state that, whatever the size of the tree, computing a bottom-up function is 4 times faster from a linear DAG than from the DAG reduction of a random tree, and nearly 20 times faster than from the tree.

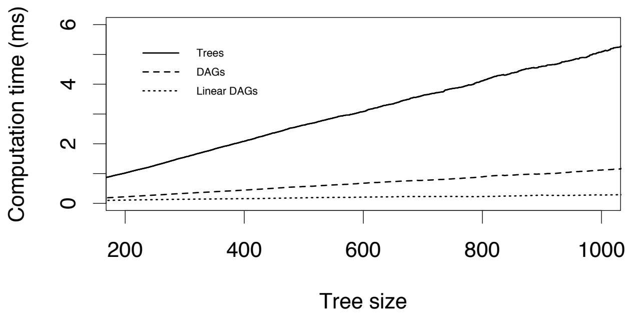


Figure 6: Estimation of the computation time of the number of vertices from trees, DAGs and linear DAGs.

3.3 Edit distance

The problem of comparing trees occurs in several diverse areas such as computational biology and image analysis. We refer the reader to the survey [2] in which the author reviews the available results and presents, in detail, one or more of the central algorithms for solving the problem. 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.

	Computation time (ms)
Trees	$5.094 \times 10^{-3}\,\#\tau$
DAGs	$1.112 \times 10^{-3}\,\#\tau$
Linear DAGs	$2.652 \times 10^{-4}\,\#\tau$

Table 2: Estimation of the slope of the three curves of Figure 6. For example, the computation time of the number of vertices from a tree of size 1 000 is 5.094 ms on average, while it is approximately 0.27 ms from the DAG reduction of a self-nested tree of the same size.

- *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 set of children of v in the transformed tree is $\text{children}(v) \setminus \{l\}$.

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 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 show in Lemma 1 in the supplementary document that δ defines a distance on the space of unordered trees.

Proposition 3. *The edit distance $\delta(\tau_1, \tau_2)$ can be computed in:*

- *$O(\#\tau_1 \#\tau_2\, \psi(\tau_1, \tau_2))$-time from the trees τ_1 and τ_2;*
- *$O(\#D_1 \#D_2 \deg(D_1) \deg(D_2)\, \psi(\tau_1, \tau_2))$-time from the DAG reductions $D_1 = \mathscr{R}(\tau_1)$ and $D_2 = \mathscr{R}(\tau_1)$;*

with

$$\psi(\tau_1, \tau_2) = (\deg \tau_1 + \deg \tau_2) \log_2(\deg \tau_1 + \deg \tau_2).$$

Consequently, if τ_1 and τ_2 are self-nested, the time-complexity of $\delta(\tau_1, \tau_2)$ is

$$O(\text{height}(\tau_1)\, \text{height}(\tau_2)\, \deg(D_1) \deg(D_2)\, \psi(\tau_1, \tau_2)).$$

The complexities stated in Proposition 2 have been illustrated through the numerical simulation of 20 000 pairs of random trees (see Figure 7). Again, the simulations show that this complex query can be answered much faster for self-nested trees than for general trees. We highlight that the computational gain is even more substantial for this quadratic operation than for computing bottom-up functions.

4 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. In this context, $\mathbb{T}_{=H,\leq d}$ ($\mathbb{T}_{\leq H,\leq d}$, respectively) denotes the set of unordered trees of height H (of height bounded by H, respectively) and outdegree bounded by d. The same notation lies for self-nested trees with the exponent sn. We give an explicit formula for the cardinality of self-nested trees in the following proposition.

Proposition 4. *For any $H \geq 1$ and $d \geq 1$,*

$$\#\mathbb{T}^{sn}_{=H,\leq d} = \prod_{i=1}^{H} \binom{d + H - i}{H - i + 1}.$$

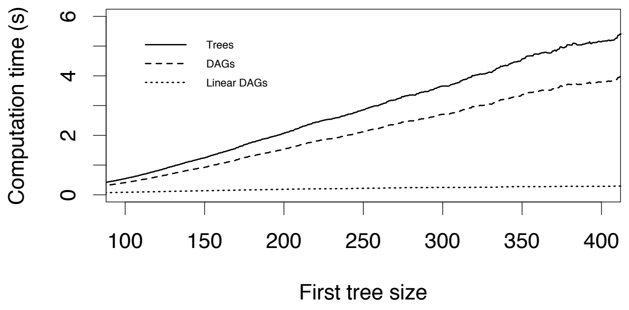


Figure 7: Estimation of the computation time of the edit distance between two trees from the trees and from their DAG reduction, for trees and self-nested trees. The computation time is displayed as a function of the first tree size since the edit distance is a symmetric function.

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 OEIS [1] A000081). In particular, the generating function associated with the unordered trees is given by

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

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 the set $\mathbb{T}_n^{sn}$ of self-nested trees of size n. A strategy could be to remark that

$$\#\mathbb{T}_n^{sn} = \sum_{H=1}^{n} \#\{\tau \in \mathbb{T}_n^{sn} \ : \ \text{height}(\tau) = H\},$$

where $\#\{\tau \in \mathbb{T}_n^{sn} \ : \ \text{height}(\tau) = H\}$ is a polynomial equation of degree H in $H(H+1)/2$ unknown variables in light of (1). Determining the number of solutions of such a Diophantine equation, even in this particular framework, remains a very difficult question.

Nevertheless, thanks to Proposition 4, we can numerically evaluate the frequency of self-nested trees (see Table 3). We have also derived an asymptotic equivalent when both the height and the outdegree go to infinity that can be compared to the cardinality of unordered trees.

		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 3: Relative frequencies of self-nested trees with given maximal height and ramification number within the set of unordered trees under the same constraint.

Proposition 5. *When H and d simultaneously go to infinity,*

$$\log \#\mathbb{T}^{sn}_{=H,\leq d} \sim \frac{(d+H)^2}{2} \log(d+H) - \frac{H^2}{2} \log H - \frac{d^2}{2} \log d - Hd \log d.$$

For the sake of comparison,

$$\log \#\mathbb{T}_{\leq H, \leq d} = \Theta(d^{H-1}).$$

[1] On-line Encyclopedia of Integer Sequences

Consequently, self-nested trees are very rare among unordered trees, since they are roughly exponentially less frequent. Exploring the space of self-nested trees is thus exponentially easier than for general trees. This is a very good point in NP problems for which exhaustive search is often chosen to determine a solution.

5 Approximation algorithm

In Section 3, we have established the numerical efficiency of self-nested trees. In addition, they are very unfrequent as seen in Proposition 5, which can be an interesting asset in exhaustive search problems. Our aim is to develop an approximation algorithm of trees by self-nested trees in order to take advantage of these remarkable algorithmic properties for any unordered tree. An application of this algorithm will be presented in Section 6.

5.1 Characterization of self-nested trees

Let τ be a tree of height H and $D = \mathscr{R}(\tau)$ its DAG reduction. With the notation of Subsection 2.2, for each vertex (h, i) of D, we define

$$\nu(h_1, i, h_2) = \sum_{j=1}^{M_{h_2}} N((h_1, i), (h_2, j))$$

that counts the number of subtrees of height h_2 which root is a child of (h_1, i). This quantity typically represents the height profile of the tree τ: it gives the distribution of the number of subtrees of height h_2 under vertices of height h_1 in τ. This feature makes us able to derive a new characterization of self-nested trees.

Proposition 6. *τ is self-nested if and only if, for any $0 \le h_2 < h_1 \le H$ and $1 \le i, j \le M_{h_1}$,*

$$\nu(h_1, i, h_2) = \nu(h_1, j, h_2). \tag{2}$$

In other words, a tree is self-nested if and only if its height profile is reduced to an array of Dirac masses. Furthermore, it should be remarked that, if (2) holds, the self-nested tree τ can be reconstructed from the array

$$[\nu(h_1, 1, h_2)]_{0 \le h_2 < h_1 \le H}.$$

Indeed, with the notation of Subsection 2.3 for self-nested trees, the label on edge (h_1, h_2) in D is $N(h_1, h_2) = \nu(h_1, 1, h_2)$.

5.2 Averaging

Given a tree τ, we construct the self-nested tree $\widehat{\tau}$ that optimally approximates, for each possible value of (h_1, h_2), the quantity $\nu(h_1, \cdot, h_2)$ in weighted $\mathbb{L}^2$-norm taking into account the multiplicity of the vertices (h_1, i) of D, $1 \le i \le M_{h_1}$. The multiplicity $\mu((h_1, i))$ of a vertex (h_1, i) of the DAG reduction D of τ is the number of occurrences of the tree $\mathscr{R}^{-1}(D[(h_1, i)])$ in τ. It is easy to see that $\mu((h_1, i))$ can be recursively computed from D as

$$\mu((h_1, i)) = \prod_{\substack{h > h_1 \\ 1 \le j \le M_h}} N((h, j), (h_1, i)) \times \mu((h, j)). \tag{3}$$

The linear DAG $\widehat{D}$ of $\widehat{\tau}$ is defined (with the notation of Subsection 2.3) by, for any $0 \le h_2 < h_1 \le \text{height}(\tau)$,

$$\begin{aligned} \widehat{N}(h_1, h_2) &= \operatorname*{arg\,min}_{x \in \mathbb{N}} \sum_{1 \le i \le M_{h_1}} \mu((h_1, i)) \left[x - \nu(h_1, i, h_2) \right]^2 \\ &= \pi \left(\frac{\displaystyle\sum_{1 \le i \le M_{h_1}} \mu((h_1, i)) \sum_{1 \le j \le M_{h_2}} N((h_1, i), (h_2, j))}{\displaystyle\sum_{1 \le i \le M_{h_1}} \mu((h_1, i))} \right), \end{aligned}$$

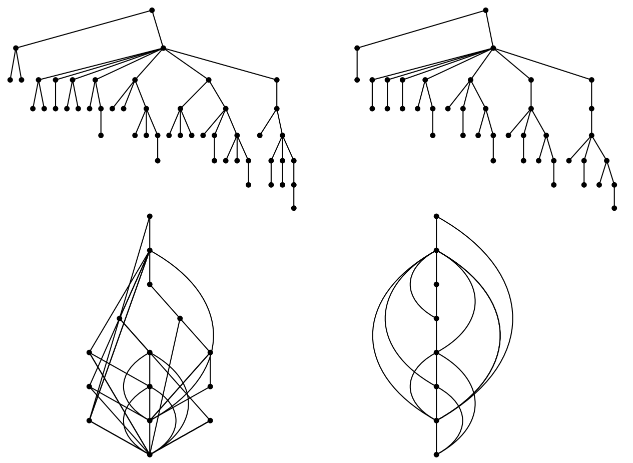


Figure 8: A random tree of size 50 (top left), its self-nested approximation tree of size 41 (top right), the nonlinear DAG reduction of the initial tree (bottom left), and the linear DAG reduction of the self-nested approximation (bottom right).

where π denotes the projection on $\mathbb{N}$, i.e., the distribution of the number of subtrees of height h_2 under vertices of height h_1 in τ is approximated by its weighted mean. By construction, $\widehat{\tau}$ is the best self-nested approximation of the height profile of τ. An example is presented in Figure 8. The complexity of this algorithm is stated in Proposition 7.

Proposition 7. *$\widehat{D}$ can be computed in $O(\#E_\equiv)$-time from the DAG reduction $D = (V_\equiv, E_\equiv, N)$ of τ.*

In [14], the authors propose to approximate a tree by its Nearest Embedding Self-nested Tree (NEST), i.e., the self-nested tree obtained from the initial data by adding a minimal number of vertices. The NEST of the tree of Figure 8 (top left) is displayed in Figure 9 for illustration purposes. They establish in [14, Theorem 1] that the NEST can be computed in $O(\text{height}(\tau)^2 \deg(\tau))$. We prove from numerical simulations that our averaging algorithm achieves better approximation errors (on average approximately 30 times lower for a tree of size 400, see Figure 10) while it requires much less computation time (on average approximately 50 times lower for a tree of size 800, see Figure 11).

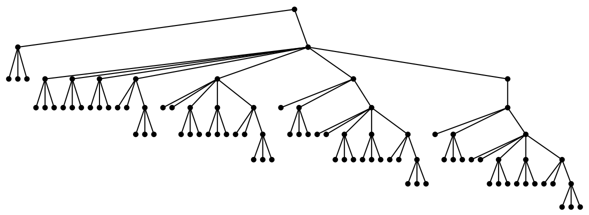


Figure 9: The NEST of the tree of Figure 8 (top left). It has 92 vertices, which corresponds to an increase of 84% of the size of the initial tree.

5.3 Error upper bound in approximation

We establish the optimal bound of the approximation error of a tree by a self-nested one in terms of edit distance δ in the following result.

Proposition 8. *For any $H \geq 2$ and d large enough (greater than a constant depending on H),*

$$\max_{t \in \mathbb{T}_{\leq H, \leq d}} \min_{\tau \in \mathbb{T}^{sn}} \delta(t, \tau) = \left\lfloor \frac{d}{2} \right\rfloor \times \left\lceil \frac{d}{2} \right\rceil \times d^{H-2}.$$

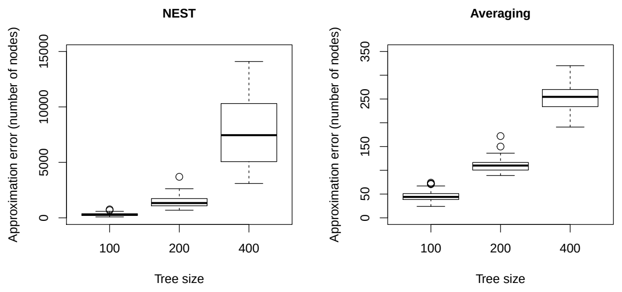


Figure 10: Comparison between NEST approximation algorithm (left) and the averaging method proposed in this paper (right) in terms of approximation error.

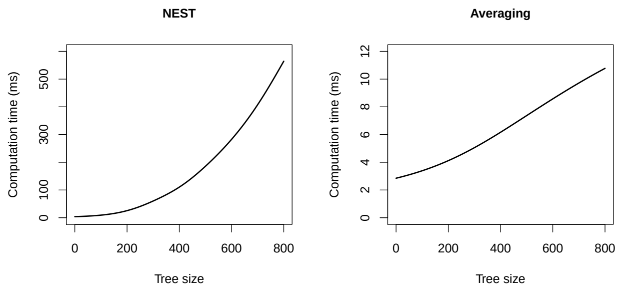


Figure 11: Comparison between NEST approximation algorithm (left) and the averaging method proposed in this paper (right) in terms of computation time.

In addition, this worst case is reached for the nonlinear DAG of Figure 12 (left).

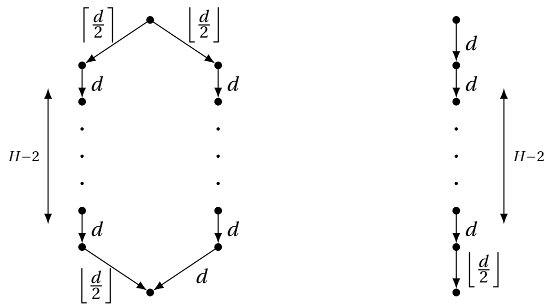
The diameter of the state space $\mathbb{T}_{\leq H, \leq d}$ is of order d^H (indeed, the largest tree of this family is the full d-tree, while the smallest tree is reduced to a unique root vertex). As a consequence and in light of Proposition 8, the largest area without any self-nested tree is a ball with relative radius

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

This result is especially noteworthy considering the very low frequency of self-nested trees compared to unordered trees (see Table 3 and Proposition 5). Nevertheless, we would like to emphasize that this negative result (exponential upper bound of the error) is far from representing the average behavior of the averaging algorithm presented in Figure 10 (right).

6 Fast prediction of edit distance

This section is dedicated to an example of application of the averaging algorithm: we show that it can be used for fast prediction of edit distance between two trees. Given two trees τ_1 and τ_2, we propose to estimate $\delta(\tau_1, \tau_2)$ by $\delta(\widehat{\tau}_1, \widehat{\tau}_2)$, where $\widehat{\tau}_i$ is the self-nested approximation of τ_i. Including the approximation step, computing the edit distance between $\widehat{\tau}_1$ and $\widehat{\tau}_2$ is on average 10 times faster than computing $\delta(\tau_1, \tau_2)$ from trees with 1 000 vertices (see Figure 13 for more details on computation times), which represents a significant gain even for trees of reasonable size.



$\left\lceil \frac{d}{2} \right\rceil$
$\left\lfloor \frac{d}{2} \right\rfloor$
d
$H-2$
d
$\left\lfloor \frac{d}{2} \right\rfloor$

Figure 12: Nonlinear DAG of height H and outdegree d (left) for which the best linear DAG approximation (right) achieves the worst error in terms of edit distance δ.

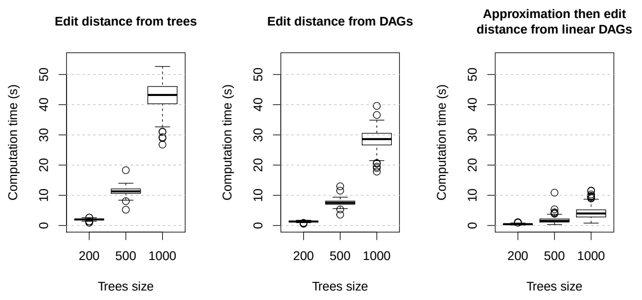


Figure 13: Estimation of the computation time of the edit distance between two trees from the trees (left), from their DAG reduction (middle), and from their linear DAG approximation (including computation of the approximations, right).

Nevertheless, $\delta(\widehat{\tau}_1,\widehat{\tau}_2)$ does not always provide a good estimate of $\delta(\tau_1,\tau_2)$: on average, the relative error is of order -50% (see Figure 14 (left)), i.e., $\delta(\widehat{\tau}_1,\widehat{\tau}_2)$ strongly underestimates $\delta(\tau_1,\tau_2)$. To improve this relative error, we correct the predictor via a learning algorithm. First, we generate from random trees (t_1,t_2) a training dataset containing 2500 replicates of $\delta(t_1,t_2)$ and $\delta(\widehat{t}_1,\widehat{t}_2)$. Then we learn a prediction rule of $\delta(t_1,t_2)$ from $\delta(\widehat{t}_1,\widehat{t}_2)$ with a linear model as implemented in the `lm` function in `R` [7]. Given two trees τ_1 and τ_2, we first compute $\delta(\widehat{\tau}_1,\widehat{\tau}_2)$ and then we predict $\delta(\tau_1,\tau_2)$ with the aforementioned prediction rule. The results on a test dataset of size 1500 are presented in Figure 14 (middle). The error is null on average but presents a large variance making the prediction not reliable.

This can be corrected by adding explanatory variables to the learning step. In the learning dataset, we add the following features: size, height, outdegree and Strahler number of t_i and $\widehat{t}_i$. It should be noted that these quantities can be computed without adding any computation time to the whole procedure during the computation of the DAG reductions. Thus, this does not affect the speed of our prediction algorithm. The results are presented in Figure 14 (right). The error is null on average with a small variance: in 50% of cases, the prediction error is between -6.5% and 6.9%. The most significant variables (p-values less than 2.10^{-16}) are $\delta(\widehat{t}_1,\widehat{t}_2)$ and the sizes of the 4 trees. In addition, the prediction rule learned from the same training dataset without $\delta(\widehat{t}_1,\widehat{t}_2)$ and the additional informations on the self-nested approximations (size, height, outdegree and Strahler number) achieve worse results in 66.9% of cases. This shows that self-nested approximations can be used to obtain a fast and accurate prediction of the edit distance between two trees.

Acknowledgments The authors thank the reviewers for their careful reading of the manuscript and their insightful comments. R.A. and C.G. were supported by the European Union's Horizon 2020 project ROMI.

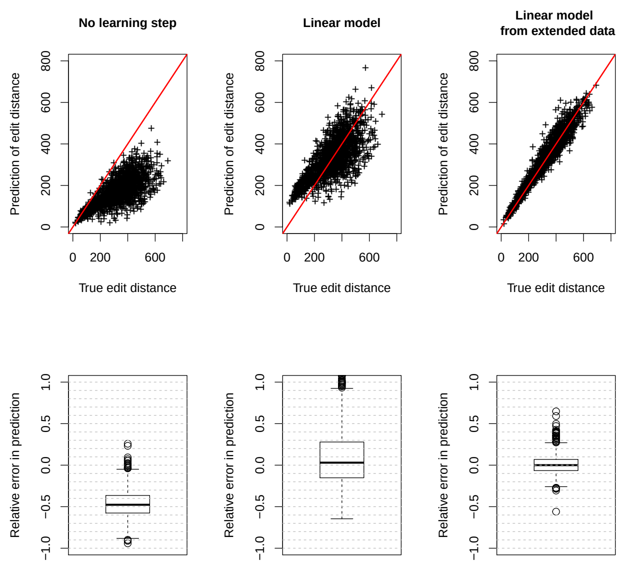


Figure 14: Prediction (top) and relative error in prediction (bottom) of the edit distance between two trees from their linear DAG approximations (left), from a linear model learned with edit distances between linear DAG approximations (middle), and from a linear model learned with edit distances between linear DAG approximations and additional informations on the trees (right).

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] Bille, P. A survey on tree edit distance and related problems. *Theoretical Computer Science 337*, 1-3 (2005), 217 – 239.

[3] 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.

[4] Bousquet-Mélou, M., Lohrey, M., Maneth, S., and Noeth, E. XML compression via directed acyclic graphs. *Theory of Computing Systems* (2014), 1–50.

[5] Buneman, P., Grohe, M., and Koch, C. Path queries on compressed XML. In *Proceedings of the 29th International Conference on Very Large Data Bases* (2003), pp. 141–152.

[6] Busatto, G., Lohrey, M., and Maneth, S. Efficient memory representation of xml document trees. *Inf. Syst. 33*, 4-5 (June 2008), 456–474.

[7] Chambers, J. M. *Statistical Models in S*. Wadsworth & Brooks/Cole, 1992, ch. Linear models.

[8] Chawathe, S. S. Comparing hierarchical data in external memory. In *Proceedings of the 25th International Conference on Very Large Data Bases* (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] GREENLAW, R. Subtree isomorphism is in dlog for nested trees. *International Journal of Foundations of Computer Science 07*, 02 (1996), 161–167.

[16] 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.

[17] LOHREY, M., MANETH, S., AND MENNICKE, R. Tree structure compression with repair. In *Data Compression Conference (DCC), 2011* (March 2011), pp. 353–362.

[18] SAKR, S. XML compression techniques: A survey and comparison. *Journal of Computer and System Sciences 75*, 5 (2009), 303 – 322.

[19] SHAPIRO, B. A., AND ZHANG, K. Comparing multiple rna secondary structures using tree comparisons. *Bioinformatics 6*, 4 (1990), 309–318.

[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.

Appendix to the paper

Approximation of trees by self-nested trees

Romain Azaïs, Jean-Baptiste Durand, and Christophe Godin

This supplementary file contains the proofs of all the original theoretical results presented in the main paper.

A Proof of Proposition 2

Computing a bottom-up function requires to traverse the tree τ from the leaves to the root in $O(\#\tau)$. From the DAG, we have, with the notation of Subsection 2.2,

$$f\big(\mathscr{R}^{-1}(D[(h_1,i)])\big) = \Phi\Bigg(\Big(\underbrace{f(\mathscr{R}^{-1}(D[(h_2,j)]))}_{N((h_1,i),(h_2,j))\text{ times}}\Big)_{(h_2,j)}\Bigg). \tag{4}$$

It should be noted that (4) holds only if Φ is invariant under permutation of arguments. In addition,

$$\sum_{h_2=0}^{h_1-1}\sum_{j=1}^{M_{h_2}} N((h_1,i),(h_2,j)) \leq \deg(\tau).$$

As a consequence, computing $f(\tau)$ from its DAG reduction requires to traverse the DAG with at most $\deg(\tau)$ operations on each vertex, which states the result.

B Proof of Proposition 3

B.1 Preliminaries

Lemma 1. *δ defines a distance function on the space of unordered rooted trees $\mathbb{T}$.*

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. □

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 $\mathscr{M}$ from τ_1 to τ_2 is a set of couples $i \to j$, $1 \leq i \leq \#\tau_1$ and $1 \leq j \leq \#\tau_2$, satisfying (see [22, 2.3.2 Editing Distance Mappings]), for any $i_1 \to j_1$ and $i_2 \to j_2$ in $\mathscr{M}$, 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 $i \to j$ 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 $i_1 \to j_1$ and $i_2 \to j_2$,

- 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 involved in the computation of our constrained edit distance δ. The equivalence between constrained mappings and δ may be stated as follows:

$$\delta(\tau_1,\tau_2) = \min_{\mathscr{M}} \left[\#\{i : \nexists j \text{ s.t. } i \to j \in \mathscr{M}\} + \#\{j : \nexists i \text{ s.t. } i \to j \in \mathscr{M}\} \right],$$

where the minimum is taken over the set of all the possible constrained tree mappings. 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 least common ancestor of two vertices and the constrained mappings presented in [22].

Least common ancestor The Least 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 in 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 $\mathscr{M}$ between trees τ_1 and τ_2 in sense of Zhang is a mapping satisfying the additional condition [22, 3.1. Constrained Edit Distance Mappings]:

- Assume that $i_1 \to j_1$, $i_2 \to j_2$ and $i_3 \to j_3$ are in $\mathscr{M}$. 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 $\mathscr{M}$ a constrained tree mapping in sense of this paper. First, one may remark that the roots are necessarily mapped together. In addition, $\mathscr{M}$ satisfies all the conditions of constrained mappings imposed by Zhang in [22] and presented above.

B.2 Reduction to the minimum cost flow problem

The edit distance between two trees τ_1 and τ_2 may be obtained from the recursive formula presented in Proposition 9 hereafter. In the sequel, $\mathscr{F}_t$ denotes the forest of all the subtrees of t which root is a child of root(t), i.e., $\mathscr{F}_t$ is the list of the subtrees of t that can be found just under its root. Furthermore, $S(n)$ denotes the set of permutations of $\{1,\ldots,n\}$ and $\binom{A}{n}$ the set of subsets of A with cardinality n.

Proposition 9. *Let τ_1 and τ_2 be two trees and $n = \min(\#\mathscr{F}_{\tau_1}, \#\mathscr{F}_{\tau_2})$. The edit distance between τ_1 and τ_2 satisfies the following induction formula,*

$$\delta(\tau_1,\tau_2) = \min_{\substack{\{t_1^1,\ldots,t_n^1\} \\ \{t_1^2,\ldots,t_n^2\} \\ \sigma}} \left[\sum_{i=1}^{n} \delta(t_i^1, t_{\sigma(i)}^2) + \sum_{\theta \notin (t_1^1,\ldots,t_n^1)} \delta(\theta,\emptyset) + \sum_{\theta \notin (t_1^2,\ldots,t_n^2)} \delta(\emptyset,\theta) \right],$$

where the minimum is taken over $\{t^1_1,\ldots,t^1_n\}\in\binom{\mathscr{F}_{T_1}}{n}$, $\{t^2_1,\ldots,t^2_n\}\in\binom{\mathscr{F}_{T_2}}{n}$ *and* $\sigma\in S(n)$, *and the symbol* $\emptyset$ *stands for the empty tree. The formula is initialized with*

$$\delta(\tau_1,\emptyset)=\#\tau_1 \qquad \textit{and} \qquad \delta(\emptyset,\tau_2)=\#\tau_2.$$

Proof. First, let us remark that a maximum number of subtrees of $\mathscr{F}_{\tau_1}$ should be mapped to subtrees of $\mathscr{F}_{\tau_2}$, 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 $n=\min(\#\mathscr{F}_{\tau_1},\#\mathscr{F}_{\tau_2})$. As a consequence, the minimal editing cost is obtained by considering all the possible mappings between n subtrees of $\mathscr{F}_{\tau_1}$ and n subtrees of $\mathscr{F}_{\tau_2}$. The subtrees that are not involved in a mapping are either deleted or added. We refer the reader to Figure 15. □

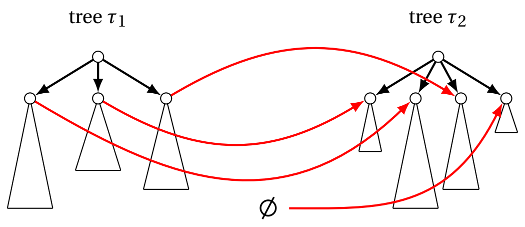


Figure 15: Schematic illustration of the recursive formula to compute the constrained edit distance δ between trees τ_1 and τ_2: the three subtrees of $\mathscr{F}_{\tau_1}$ are mapped to three subtrees of $\mathscr{F}_{\tau_2}$, while the empty tree $\emptyset$ is mapped to the fourth subtree of $\mathscr{F}_{\tau_2}$, i.e., all the vertices are added.

In light of Proposition 9 and Figure 15 and as in [22, 5. Algorithm and complexity], each step in the recursive computation of the edit distance $\delta(\tau_1,\tau_2)$ between trees τ_1 and τ_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

$$V=\{\text{source},\text{sink},\emptyset_{\tau_1},\emptyset_{\tau_2}\}\cup\mathscr{F}_{\tau_1}\cup\mathscr{F}_{\tau_2}.$$

The set E of edges of G is defined from:

- source $\to t^1_i$, $t^1_i\in\mathscr{F}_{\tau_1}$
  - capacity: 1
  - cost: 0
- source $\to \emptyset_{\tau_1}$
  - capacity: $\#\mathscr{F}_{\tau_1}-\min(\#\mathscr{F}_{\tau_1},\#\mathscr{F}_{\tau_2})$
  - cost: 0
- $t^1_i\to t^2_j$, $t^1_i\in\mathscr{F}_{\tau_1}$, $t^2_j\in\mathscr{F}_{\tau_2}$
  - capacity: 1
  - cost: $\delta(t^1_i,t^2_j)$
- $t^1_i\to\emptyset_{\tau_2}$, $t^1_i\in\mathscr{F}_{\tau_1}$
  - capacity: 1
  - cost: $\delta(t^1_i,\emptyset)=\#t^1_i$
- $\emptyset_{\tau_1}\to t^2_j$, $t^2_j\in\mathscr{F}_{\tau_2}$
  - capacity: 1

  - cost: $\delta(\emptyset, t_j^2) = \# t_j^2$

- $t_j^2 \to \text{sink}$, $t_j^2 \in \mathscr{F}_{\tau_2}$
  - capacity: 1
  - cost: 0

- $\emptyset_{\tau_2} \to \text{sink}$
  - capacity: $\#\mathscr{F}_{\tau_2} - \min(\#\mathscr{F}_{\tau_1}, \#\mathscr{F}_{\tau_2})$
  - cost: 0

We obtain a network G augmented with integer capacities and nonnegative costs. A representation of G is given in Figure 16. By construction and as explained in [22, Lemma 8], one has $C(G) = \delta(\tau_1, \tau_2)$ where $C(G)$ denotes the cost of the minimum cost maximum flow on G. As a consequence, $\delta(\tau_1, \tau_2)$ 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 3.

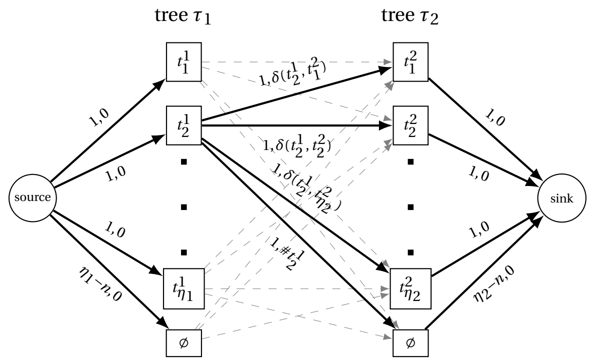


Figure 16: Reduction of the computation of edit distance $\delta(\tau_1, \tau_2)$ presented in Proposition 9 and in Figure 15 to the minimum cost flow problem. Each edge is augmented with two labels separated by a comma: its capacity (left) and its cost (right). For the sake of simplicity, $\eta_i = \#\mathscr{F}_{\tau_i}$ and $n = \min(\eta_1, \eta_2)$.

B.3 Complexity from trees

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 16 may be directly obtained from its characteristics and is $O(N \times |f^\star| \times \log_2(n))$, where n, N and $|f^\star|$ respectively denote the number of vertices, the number of edges and the maximum flow of G. It is quite obvious that

$$N = O\big(\#\text{children}(\text{root}(\tau_1))\#\text{children}(\text{root}(\tau_2)) + \#\text{children}(\text{root}(\tau_1)) + \#\text{children}(\text{root}(\tau_2))\big).$$

In addition,

$$\begin{aligned} |f^\star| &= O\big(\#\text{children}(\text{root}(\tau_1)) + \#\text{children}(\text{root}(\tau_2))\big) \\ &= O(\deg \tau_1 + \deg \tau_2). \end{aligned}$$

And,

$$\begin{aligned} n &= O\big(\#\text{children}(\text{root}(\tau_1)) + \#\text{children}(\text{root}(\tau_2))\big) \\ &= O(\deg \tau_1 + \deg \tau_2). \end{aligned}$$

Thus, the total complexity to compute the recursive formula of $\delta(\tau_1, \tau_2)$ presented in Proposition 9 is

$$O\Big([\deg(\tau_1) + \deg(\tau_2)] \log_2(\deg(\tau_1) + \deg(\tau_2)) \times \sum_{v\in\tau_1} \sum_{w\in\tau_2} \#\mathrm{child}(v) \times \#\mathrm{child}(w)\Big) = O(\#\tau_1 \#\tau_2\, \psi(\tau_1, \tau_2)),$$

with $\psi(\tau_1, \tau_2) = (\deg \tau_1 + \deg \tau_2) \log_2(\deg \tau_1 + \deg \tau_2)$, 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.

B.4 Complexity from DAG reductions

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 for DAG reductions of trees. Taking into account the redundancies, the edit distance obtained from DAG reductions is expected to be computed with a time-complexity smaller than from trees.

In the sequel, D_i denotes the DAG reduction of the tree τ_i of height H_i. With the notation of Subsection 2.2, the vertices of D_i are denoted by $(h, j)^i$, $0 \le h \le H_i$ and $1 \le j \le M_h^i$. In addition, the DAG D_i is characterized by the array

$$\big[N^i((h_1, j)^i, (h_2, k)^i)\big].$$

We also define

$$\eta_i = \sum_{\substack{0\le h<H_i\\ 1\le j\le M_h^i}} N^i((H_i, 1)^i, (h, j)^i),$$

which counts the number of children of $\mathrm{root}(\tau_i)$. As in Subsection B.2, the computation of the edit distance $\delta(\tau_1, \tau_2)$ from the DAG reductions D_1 and D_2 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 subtrees of $\mathscr{F}_{\tau_1}$ and $\mathscr{F}_{\tau_2}$. We construct the network graph $G = (V, E)$ as follows. The set of vertices V of G is given by

$$V = \{\mathrm{source}, \mathrm{sink}, \emptyset_1, \emptyset_2\} \cup \bigcup_{\substack{0\le h<H_1\\ 1\le j\le M_h^1}} \big\{(h, j)^1\big\} \cup \bigcup_{\substack{0\le l<H_2\\ 1\le k\le M_l^2}} \big\{(l, k)^2\big\}.$$

The set E of edges of G is defined from:

- source $\to (h, j)^1$
  - capacity: $N^1((H_1, 1)^1, (h, j)^1)$
  - cost: 0
- source $\to \emptyset_1$
  - capacity: $\eta_1 - \min(\eta_1, \eta_2)$
  - cost: 0
- $(h, j)^1 \to (l, k)^2$
  - capacity: $N^1((H_1, 1)^1, (h, j)^1)$
  - cost: $\delta(\mathscr{R}^{-1}(D_1[(h, j)^1]), \mathscr{R}^{-1}(D_2[(l, k)^2]))$
- $(h, j)^1 \to \emptyset_2$
  - capacity: $N^1((H_1, 1)^1, (h, j)^1)$
  - cost: $\#\mathscr{R}^{-1}(D_1[(h, j)^1])$

- $\emptyset_1 \to (l,k)^2$
  - capacity: $N^2((H_2,1)^2,(l,k)^2)$
  - cost: $\#\mathscr{R}^{-1}(D_2[(l,k)^2])$
- $(l,k)^2 \to$ sink
  - capacity: $N^2((H_2,1)^2,(l,k)^2)$
  - cost: 0
- $\emptyset_2 \to$ sink
  - capacity: $\eta_2 - \min(\eta_1,\eta_2)$
  - cost: 0

As in Subsection B.2, the graph G has integer capacities and nonnegative costs on its edges. By construction, the cost $C(G)$ of the minimum cost maximum flow on the graph G is equal to the expected edit distance

$$\delta(\tau_1,\tau_2) = \delta\big(\mathscr{R}^{-1}(D_1[(H_1,1)]),\mathscr{R}^{-1}(D_2[(H_2,1)])\big).$$

The characteristics n (number of vertices), $|f^\star|$ (maximum flow) and N (number of edges) of the network G satisfy:

- $N = O(\deg D_1 \times \deg D_2 + \deg D_1 + \deg D_2)$;
- $|f^\star| = O(\deg \tau_1 + \deg \tau_2)$;
- $n = O(\deg D_1 + \deg D_2) = O(\deg \tau_1 + \deg \tau_2)$.

Summing over the vertices of D_1 and D_2 (number of terms bounded by $\#D_1 \times \#D_2$), together with [21, Theorem 8.13], states the result.

C Proof of Proposition 4

Using the notation of Subsection 2.3, a self-nested tree of height H is represented by a linear DAG with $H+1$ vertices numbered from 0 (bottom, leaves of the tree) to H (top, root of the tree) in such a way that there exists a path $H \to \cdots \to 1 \to 0$. One recalls that this graph is augmented with integer-valued label $N(i,j)$ on edge $i \to j$ 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 d if and only if, for any i,

$$\sum_{j=0}^{i-1} N(i,j) \leq d.$$

We propose to write $n_{i,i-1} = N(i,i-1) - 1$ 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, for any $i > j$, $n_{i,j} \geq 0$ and, for any $i \geq 1$,

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

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

$$\#\mathbb{T}^{sn}_{=H,\leq d} = \prod_{i=1}^{H} \#\left\{ n_{i,j} \geq 0 \,:\, \sum_{j=0}^{i-1} n_{i,j} \leq d-1 \right\}.$$

Furthermore, the set under the product sign is only the regular discrete simplex of dimension i having d 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

$$\#\left\{ n_{i,j} \geq 0 \,:\, \sum_{j=0}^{i-1} n_{i,j} \leq d-1 \right\} = \binom{d+i-1}{i},$$

which yields the expected result via a change of index.

D Proof of Proposition 5

D.1 Asymptotics for self-nested trees

Substituting the binomial coefficients by their value in the expression of $\#\mathbb{T}^{sn}_{=H,\leq d}$ stated in Proposition 4, we get

$$\#\mathbb{T}^{sn}_{=H,\leq d} = \Gamma(d)^{-H} \prod_{i=1}^{H} \frac{\Gamma(d+H-i+1)}{\Gamma(H-i+2)}$$

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

$$\begin{aligned} \log \#\mathbb{T}^{sn}_{=H,\leq d} &= -H \log \Gamma(d) + \sum_{\substack{1\leq i\leq H \\ 2\leq k\leq d}} \log(H-i+k) \\ &= -H \log \Gamma(d) + \sum_{\substack{0\leq j\leq H-1 \\ 2\leq k\leq d}} \log(j+k), \end{aligned} \tag{5}$$

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

$$-H \log \Gamma(d) \sim -Hd \log d. \tag{6}$$

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 $1 \leq k \leq d$. We have

$$\begin{aligned} \sum_{j=1}^{H}\sum_{k=1}^{d} \log(j+k) &= \sum_{j=1}^{H}\sum_{k=1}^{d} \int_1^{j+k} \frac{\mathrm{d}x}{x} \\ &= \sum_{j=1}^{H} \left[\sum_{l=0}^{d-1} (d-l) \int_{j+l}^{j+l+1} \frac{\mathrm{d}x}{x} + \int_1^{j} \frac{\mathrm{d}x}{x} \right] \\ &= \sum_{l=0}^{d-1} (d-l) \left[\sum_{j=1}^{H} \int_{j+l}^{j+l+1} \frac{\mathrm{d}x}{x} + \log j \right] \\ &= \sum_{l=0}^{d-1} (d-l) \int_{l+1}^{l+H+1} \frac{\mathrm{d}x}{x} + d \sum_{j=1}^{H} \log j \\ &= \sum_{l=0}^{d-1} (d-l) \log\left(1 + \frac{H}{l+1}\right) + d \sum_{j=1}^{H} \log j. \end{aligned} \tag{7}$$

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

$$\sum_{l=0}^{d-1} (d-l) \log\left(1 + \frac{H}{l+1}\right) + d \sum_{j=1}^{H} \log j \sim \frac{(d+H)^2}{2} \log(d+H) - \frac{H^2}{2} \log H - \frac{d^2}{2} \log d + R(d,H), \tag{8}$$

where the rest $R(d,H)$ is neglectable with respect to the other terms and to $Hd \log d$. Let us remark that the expression of the equivalent is symmetric in H and d as expected. Finally, (5), (6), (7) and (8) show the result.

D.2 Asymptotics for unordered trees

Roughly speaking, an unordered tree with maximal height H and maximal outdegree d may be obtained by adding at most d trees of height less than $H-1$ to an isolated root. More precisely, one has to choose d elements with repetitions among the set $\mathbb{T}_{\leq H-1,\leq d} \cup \{\bullet\} \cup \{\emptyset\}$ and add them to the list of children (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 d times the symbol $\emptyset$), or a tree with maximal height H. As a consequence, one has the formula,

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

which shows that

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

with $u_0(d) = 1$ and

$$u_H(d) = \binom{u_{H-1}(d) + d}{d}.$$

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

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

We have $u_1(d) = \binom{1+d}{d} = d + 1$ and

$$\begin{aligned} u_2(d) &= \binom{u_1(d) + d}{d} \\ &= \binom{2d+1}{d} \\ &\geq \left(\frac{2d+1}{d}\right)^d \\ &= \left(2 + \frac{1}{d}\right)^d. \end{aligned}$$

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

$$u_H(d) \geq \frac{\left(2 + \frac{1}{d}\right)^{d^{H-1}}}{d^{\frac{d^{H-1}-1}{d-1} - 1}},$$

we have

$$\begin{aligned} u_{H+1}(d) &= \binom{u_H(d) + d}{d} \\ &\geq \left(\frac{u_H(d)}{d} + 1\right)^d \\ &\geq \left(\frac{u_H(d)}{d}\right)^d \\ &\geq \left(\frac{\left(2 + \frac{1}{d}\right)^{d^{H-1}}}{d^{\frac{d^{H-1}-1}{d-1}}}\right)^d \end{aligned}$$

by the induction hypothesis. Using

$$d\left(\frac{d^{H-1} - 1}{d-1}\right) = \frac{d^H - 1}{d-1} - 1,$$

we obtain

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

Moreover,

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

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

$$u_H(d) \leq 3^{d^{H-1}}\,\mathrm{e}^{\frac{d^H-1}{d-1}-1},$$

we obtain

$$\begin{aligned} u_{H+1}(d) &= \binom{u_H(d)+d}{d} \\ &\leq \left(\left(\frac{u_H(d)}{d}+1\right)\mathrm{e}\right)^d \\ &\leq \left(\left(\frac{3^{d^{H-1}}\,\mathrm{e}^{\frac{d^H-1}{d-1}-1}}{d}+1\right)\mathrm{e}\right)^d \end{aligned}$$

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 $1.693\ldots$ (obtained by numerical methods), we obtain

$$\begin{aligned} u_{H+1}(d) &\leq \left(3^{d^{H-1}}\,\mathrm{e}^{\frac{d^H-1}{d-1}-1}\,\mathrm{e}\right)^d \\ &= 3^{d^H}\,\mathrm{e}^{\frac{d^{H+1}-1}{d-1}-1}. \end{aligned}$$

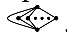
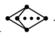
This shows the expected result.

E Proof of Proposition 7

Computing all the multiplicities $\mu((h_1,i))$ can be done in one traversal of all the edges of D via the recursive formula (3). By definition of $\widehat{D}$, for each couple (h_1,h_2), computing $\widehat{N}(h_1,h_2)$ requires to traverse all the edges $(h_1,i) \to (h_2,j)$ with no overlap. Finally, all the edges of D have been traversed once, which states the complexity.

F Proof of Proposition 8

We begin this proof 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 n intermediate vertices (that is to say n different subtrees of height 1) labeled from I_1 to I_n, $n \leq d$. Of course, $n=1$ 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 $1 \leq k \leq n$.

We shall investigate the worst case for a given value of n. 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 d is even and n divides d. The explicit solution of the problem is thus $p_k = \frac{d}{n}$, $l_k = \frac{kd}{n}$, $x = \frac{d}{2}$ and $A = \emptyset$. The remarkable fact is that the corresponding cost is given by

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

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

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 n different patterns $I_1, \ldots, I_n$ appearing $p_1, \ldots, p_n$ times under the root. The cost of editing operations (adding or deleting leaves) at distance h to the root is in the worst case $p_k d^{h-1}$. As a consequence, at least for d large enough, height$(I_k) = H - 1$ and the only difference with the other patterns is on the fringe: all the vertices of I_k have d 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

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

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