Quantifying the degree of self-nestedness of trees. Application to the structural analysis of plants

Christophe Godin, Pascal Ferraro

Quantifying the degree of self-nestedness of trees. Application to the structural analysis of plants

Christophe Godin — Pascal Ferraro

N° 6800

Janvier 2009

Thème BIO

ISSN 0249-6399 ISRN INRIA/RR--6800--FR+ENG

Quantifying the degree of self-nestedness of trees.
Application to the structural analysis of plants

Christophe Godin [*] , Pascal Ferraro[†]

Thème BIO — Systèmes biologiques
Équipes-Projets VirtualPlants

Rapport de recherche n° 6800 — Janvier 2009 — 42 pages

Abstract: In this paper we are interested in the problem of approximating trees by trees with a particular self-nested structure. Self-nested trees are such that all their subtrees of a given height are isomorphic. We show that these trees present remarkable compression properties, with high compression rates. In order to measure how far a tree is from being a self-nested tree, we then study how to quantify the degree of self-nestedness of any tree. For this, we define a measure of the self-nestedness of a tree by constructing a self-nested tree that minimizes the distance of the original tree to the set of self-nested trees that embed the initial tree. We show that this measure can be computed in polynomial time and depict the corresponding algorithm. The distance to this nearest embedding self-nested tree (NEST) is then used to define compression coefficients that reflect the compressibility of a tree.

To illustrate this approach, we then apply these notions to the analysis of plant branching structures. Based on a database of simulated theoretical plants in which different levels of noise have been introduced, we evaluate the method and show that the NESTs of such branching structures restore partly or completely the original, noiseless, branching structures. The whole approach is then applied to the analysis of a real plant (a rice panicle) whose topological structure was completely measured. We show that the NEST of this plant may be interpreted in biological terms and may be used to reveal important aspects of the plant growth.

Key-words: tree reduction, self-similarity, tree compression, tree-to-tree edit distance, plant architecture, branching structures, meristem, differentiation state.

Proposed to IEEE TCBB

[*] C. Godin is with INRIA Sophia-Antipolis, Virtual Plants project-team, UMR DAP, Montpellier.

[†] P. Ferraro is with University of Bordeaux, LABRI in a CNRS leave of absence at the Pacific Institute for Math. Sciences of the University of Calgary.

Quantification du degr d'auto-embotement des arborescences. Application l'analyse structurelle des plantes.

Résumé : Dans ce rapport nous nous intéressons au problème d'approximation d'arborescences á partir d'arborescences ayant une structure emboitée particulière. Les arborescences autoemboitées sont telles que tous leurs sous-arborescences d'une hauteur donnée sont isomorphes. Nous montrons que ces arborescences présentent des propriétés de compression remarquables. Afin de mesurer de combien une arborescence s'éloigne d'une arborescence autoemboitée, nous étudions alors la quantification du degré d'autoemboitement d'une arborescence. Pour cela, nous définissons une mesure d'autoemboitement pour tout arbre en construisant l'arborescence autoemboitée qui minimise la distance d'édition entre l'arborescence originiale et l'ensemble des arborescences autoemboitées qui contiennent l'arborescence initiale. Nous montrons que cette mesure peut être calculée en un temps polynomial et présentons l'algorithme correspondant. La distance au plus petit arbre autoemboitée (NEST) est alors utilisée pour définir les coefficients de compression qui reflètent la compressibilité de l'arborescence.

Pour illustrer cette approche, nous appliquons ces notions à l'analyse de structures ramifiées. En nous appuyant sur une base de données de plantes théoriques dans laquelle différents niveaux de bruits ont été introduits, nous évaluons la méthode proposée et montrons que le NEST de telles structures ramifiées est isomorphe partiellement ou complètement à la structure ramifiée originale sans bruit. L'approche globale est ensuite appliquée à l'analyse d'une plante réelle (une panicule de riz) dont la structure topologique a été entièrement mesurée. Nous montrons que le NEST de cette plante peut être interprété en termes biologiques et peut être utilisé pour révélé les aspects importants de sa croissance.

Mots-clés : réduction d'arborescences, auto-similarité, compression, distance d'édition, architecture des plantes, systèmes ramifiés, méristème, état de differentiation.

1 Introduction

Biological motivation. Plants are branching living organisms that develop throughout their lifetimes. Organs are created by small embryogenetic regions at the tip of each axis, called apical meristems (or simply meristems). During plant ontogeny, meristems develop branching structures that show remarkable organizations, made up of many similar organs at different scales: leaves, shoots, axes and branching systems of different sizes [1, 2, 3, 4, 5]. Important progresses in the understanding of these growth processes have been made in the last decades by studying and quantifying real plants and their development under various environmental conditions. Two complementary approaches are being used. In simulation approaches, developmental models are built in order to reproduce the essence of the plant development with a few parameters in a simulation model, see [6, 7, 8] for reviews. On the other hand, in descriptive approaches, quantitative analyses of observed plant structures are tentatively used to reveal regularities or gradients hidden in the complex organization of plant structures, see [9] for a review. In the recent years, both the simulation and the descriptive approaches are being combined to obtain more accurate models of plant development and assess them quantitatively against real data, e.g. [10, 11, 12]. In the descriptive approach, a lot of techniques have been developed for analyzing distributions of events in the plant structure (e.g. [13, 14]) or sequences of events along a branch or a meristem trajectory (e.g. [15, 16]). Comparatively less attention has been paid to the development of methods for directly characterizing tree-like structures, e.g. [17, 18, 19]. However, the natural organization of plants is primarily observed at the level of branching systems which qualitatively show strong internal similarities between their own parts. In many cases, this repetition of quasi-identical structures is accompanied with the impression that the branching systems are nested one into the others. Although these phenomena have been empirically described by botanists for decades, [20, 21, 22, 23], no algorithmic approach was developed yet to address this problem of recognizing similar, possibly nested, patterns in plant structures. The aim of this paper is to develop such a computational framework and to illustrate its application to plant architecture analysis.

Characterizing the nested structure of rooted trees. Plant structures are usually represented by either ordered or unordered rooted trees [24, 25] and a number of algorithms have been developed in computer science on such trees that have connections to our problem.

A first set of approaches makes it possible to compare quantitatively the structure of two trees. They are based on the use of an *edit-distance approach*, in which a metric is defined that reflects the minimum number of elementary edit operations necessary to transform one tree into the other. These algorithms solve different tree-to-tree comparison problems, such as defining a metric between trees, finding whether a tree is included into another one [26], finding the consensus tree between two trees (*i.e.* the minimal tree that contains both) [27], or the maximum common subtree [28, 29], *etc.* Usually, to answer each question, a whole family of algorithms is developed to account for different characteristics of either the input trees (ordering of nodes, labelling) or the comparison problem (constraints on the valid edit operations, etc.). In the context of plant modeling, based on an original algorithm proposed by Zhang in 1993 [30, 31], we studied in a previous work how to use and adapt such algorithms to compare plant architectures from a structural point of view [17]. However, all these studies concentrate on the comparison between two different trees and usually pay no attention to characterizing the internal structure of a tree.

In a different spirit, the problem of studying internal repetitions of structures in a tree has been addressed by eliminating the structural redundancy appearing in trees (or in graphs). For this, similar parts in a tree are condensed, resulting in a *directed acyclic graph* (DAG). Such an approach was used in different domains. Based on a pioneering work by Akers [32], Bryant [33] introduced one of the very first uses of such DAGs to represent efficiently the graph of Boolean functions. In this application, these function graphs are actually binary, ordered DAGs (each node of such a DAG has exactly two children, with a first and a second child). An algorithm is depicted that reduces these function DAGs to canonical DAGs from which all the structural redundancy of the initial DAG has been removed. It is closely related to the algorithm described in [34] for testing whether two trees are isomorphic. DAG representations of trees are also much used in computer graphics where the process of condensing a tree into a graph is called *object instancing* [35]. This process, first used by Sutherland in 1963 [36], makes it possible to share nodes representing scene objects and thus avoids the unnecessary duplication of different instances of the same graphical object. This efficient scheme is now commonly implemented as scene graphs in computer graphics applications. It allows to manipulate efficiently very huge scenes and was notably applied to the rendering of complex fractal scenes [35, 37] and plant scenes, e.g. [38, 35, 39].

Finally, our problem is also connected to the notion of structural self-similarity in trees. While self-similarity usually refers to purely geometric properties of objects (parts of an object are *geometrically* similar to the entire object up to a scaling factor), structural self-similarity attempts to capture an equivalent idea for structures and graphs. Different approaches of structural self-similarities have been tentatively proposed. Authors defined self-similarity by analysing either global branching parameters of trees e.g. [40, 41, 42, 43] or topological structural properties of graphs [44, 7]. Interestingly, in the context of studying efficient subtree isomorphisms, Greenlaw [45] introduced the definition of *nested trees*. As such, nested trees are not strictly self-similar, but they have an internal recursive structure that makes them a closely related notion. In this paper, we call such trees *self-nested trees* to insist on their recursive structure and on their proximity to the notion of structural self-similarity. Structural self-similarity was also introduced in the context of plant modelling by Prusinkiewicz [7] based on botanical insights of Arber [21]. This definition relies on the use of L-system rules, and was shown to grasp the essence of the pattern recursion through scales included in the idea of self-similarity. Subsequently, an alternative approach to structural self-similarity in plants was proposed by Ferraro *et al.* [19], who used edit-distance metrics to find recursively similarities between the high order branches of a plant and its trunk.

Based on concepts coming from these three areas, we introduce in this paper a new algorithmic measure to quantify the degree of self-nestedness of trees. In a first step, we present the formal framework and main theoretical results. We then show how such tools can be applied to the analysis of patterns in plant structures and how they can give first insights on the more or less important self-nested nature of plants and on their development. The paper starts by studying different algorithms to reduce trees as DAGs (section II). We extend Bryant algorithm to unordered trees and show that this extension is closely connected to the definition of tree-to-tree edit-distance algorithms. Then, in section III, we introduce the notion of self-nested trees as the trees whose reduced graph is linear and study several of their properties. Using this framework, we consider the question of computing, for any given tree, a nearest embedding self-nested tree (NEST), *i.e.* a self-nested tree that minimizes the tree-to-tree edit-distance to the initial tree and that embeds it. This leads us to the main result of this paper in which we show that this question can be solved in polynomial time and give the algorithm. The distance between a tree and its NEST derives from this algorithm and defines a notion of degree of self-nestedness of a tree. In the second part of the paper, we then apply this theoretical framework to the analysis of plant self-nestedness (section IV). We illustrate the notion on different simulated theoretical plants and on a measured plant. We show that the study of similarities between all parts of a plant boils down to studying the self-nested nature of the plant structure. We define the degree of self-nestedness of any plant as a departure coefficient from pure self-nestedness. Finally, as a by-product of such an analysis, we show that the method enables us to identify putative hierarchies of meristem states that could be further exploited in combination with investigations at a biomolecular level to better understand plant development.

2 Tree reduction

2.1 Definitions and notations

In the sequel, we will use the following definitions and notations. A *multiset* is a set of typed elements such that the number of elements of each type is known. It is defined as a set of pairs $M = \{(k, n_k)\}_k$ where k varies over the element types and n_k is the number of occurrences of type k in the set. A *finite oriented graph*, or simply a *graph*, is a pair $G = (V, E)$ where V denotes a finite set of *vertices* and E denotes a finite set of ordered pairs of vertices called *edges*. Let (x, y) be an edge of E, x is called a parent of y and y is a child of x. The set of children of a node x is denoted by $child(x)$. A vertex that has no child is called a leaf. $|G|$ represents the number of vertices of G. We shall sometimes say that a vertex x is in G, meaning $x \in V$. A chain between vertex x and vertex y is a (possibly empty) sequence of vertex pairs $\{\{x_i, y_i\}\}_{i=1,M}$ such that either (x_i, y_i) or (y_i, x_i) is an edge, $\{x_i, y_i\} \cap \{x_{i+1}, y_{i+1}\} \neq \emptyset$ and x_1 or $y_1 = x$ and x_M or $y_M = y$. A path from a vertex x to a vertex y is a (possibly empty) sequence of edges $\{(x_i, x_{i+1})\}_{i=1,M-1}$ such that $x_1 = x, x_M = y$. A vertex x is called an *ancestor* of a vertex y (and y is called a *descendant* of x) if there exists a path from x to y. A *cycle* is a non-empty chain between one vertex and itself. A *directed cycle* is a non-empty path from one vertex to itself. Two vertices of a simple graph are *connected* if there exists a chain between the two vertices (a vertex is always connected to itself). A graph is connected if any pair of vertices are connected.

A *directed acyclic graph* (DAG) is a graph containing no directed cycle (but which may contain cycles). In a DAG, the ancestor relationship is a partial order relation denoted by $\preceq$, [46].

A *tree* is a connected graph containing no cycle. A *rooted tree* is a tree such that there exists a unique vertex, called the root, which has no parent vertex, and any vertex different from the root has exactly one parent vertex. In the following, a rooted tree is called simply a tree. In this paper, we consider the set of rooted unordered trees, noted $\mathcal{T}$. Unordered trees are trees for which the order among the sibling vertices of any given vertex is *not* significant. The degree deg of a tree is the maximum number of children of a vertex of T. The height $h(x)$ of a vertex x in a tree T is recursively defined as $h(x) = 0$ if x is a leaf and as $h(x) = \max_{y \in child(x)}\{h(y)\} + 1$ otherwise.

A *subtree* is a particular connected subgraph of a tree. Let x be a vertex of a tree $T = (V, E)$, $T[x]$ is a *complete subtree* if it is the maximal subtree rooted in x: $T[x] = (V[x], E[x])$, where $V[x] = \{y \in V | x \preceq y\}$ and $E[x] = \{(u, v) \in E | u \in V[x], v \in V[x]\}$. In the sequel, we will only consider complete subtrees and use the simpler term "subtree" as a shorthand notation. A *forest* is a graph whose connected components are trees. Let $x_1, x_2, \ldots, x_K$ be the children of a vertex x of a tree T, $F[x]$ denotes the forest rooted in x, *i.e.* the forest consisting of the subtrees of $T[x_k]$ respectively rooted in $x_1, x_2, \ldots, x_K$.

Definition 1 (tree isomorphism) *Let us consider two rooted trees, $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$. A bijection ϕ from V_1 to V_2 is a tree isomorphism if for each $(x, y) \in E_1$, $(\phi(x), \phi(y)) \in E_2$.*

If there exists an isomorphism between two structures T_1 and T_2, the two structures are identical up to a relabeling of their components. In this case, we write $T_1 \equiv T_2$, and say that T_1 is *isomorphic to* T_2. By extension, two subforests $F_1[x]$ and $F_2[y]$ of T_1 and T_2 are isomorphic if there exists a bijection ψ from the children x_i of x to the children of y such that $T_1[x_i] \equiv T_2[\psi(x_i)]$.

Let us now consider the equivalence relation defined by the tree isomorphisms on the set of (complete) sub-trees of a tree $T = (V, E)$. We say that vertices x and y in V are equivalent if $T[x] \equiv T[y]$, and we note by extension $x \equiv y$. For each $x \in V$, let $c(x)$ denote the equivalence class of x. Throughout the paper, we consider the following partial order relations between two trees T_1 and T_2:

- $T_1 \subseteq T_2$ if T_1 *is a subtree (not necessarily complete) of* T_2 (*i.e.* T_2 can be obtained from T_1 by adding vertices only).
- $T_1 \sqsubseteq T_2$ if T_1 *is isomorphic to a (complete) subtree of* T_2. In a tree T, if $z \preceq x$, then $T[x] \sqsubseteq T[z]$ (Fig. 1.a).

Let us consider the quotient graph $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$ obtained from T using the above equivalence relation on T vertices. $V_\mathcal{Q}$ is the set of equivalence classes I on V. $E_\mathcal{Q}$ is a set of pairs of equivalence classes such that $(I, J) \in E_\mathcal{Q}$ if and only if $\exists (x, y) \in E$, $c(x) = I$ and $c(y) = J$. Note that in this case, $x \preceq y$ and $T[y] \sqsubset T[x]$.

Proposition 1 *Let T be a finite tree, then $\mathcal{Q}(T)$ is a DAG.*

Proof: Assume that there exists an oriented cycle $\{(I_1, I_2), ..., (I_{n-1}, I_n)\}$ in $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$, with $I_n = I_1$. Then for $k \in \{1, .., n-1\}$, $(I_k, I_{k+1}) \in E_\mathcal{Q}$, implies that there exists x_k and x_{k+1} such that $c(x_k) = I_k, c(x_{k+1}) = I_{k+1}$ and $T[x_{k+1}] \sqsubset T[x_k]$,where the inclusion between both trees is strict. This means that $T[x_n] \sqsubset ... \sqsubset T[x_k] \sqsubset ... \sqsubset T[x_1]$, where all inclusions are strict. However,

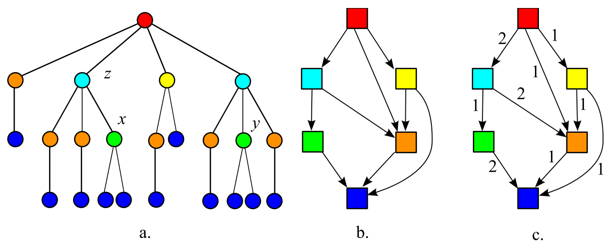

Figure 1: a. A rooted tree T. Isomorphic nodes are colored identically: $T[x] \equiv T[y]$ and $z \preceq x$ then $T[y] \sqsubseteq T[z]$. b. Quotient graph $\mathcal{Q}(T)$ associated with T: vertices of this graph are equivalence classes colored according to the color of the class of isomorphic vertices they represent in T. c. Reduction graph of T corresponding to $\mathcal{Q}(T)$ whose edges are labeled with the signature distribution function n.

since $I_n = I_1$, $x_n \equiv x_1$ and then $T[x_n] \equiv T[x_1]$, which is a contradiction with the preceding series of strict nested inclusions. ■

$\mathcal{Q}(T)$ has a single source (resp. sink) vertex. The source vertex represents the class of the entire tree while the sink vertex represents the class of all leaves. Each path $(x_1, \ldots, x_k)$ in T corresponds to a path $(c(x_1), \ldots, c(x_k))$ with identical length in $\mathcal{Q}(T)$ (Fig. 1). $\mathcal{Q}(T)$ obviously condenses the structural information contained in the tree T. However, $\mathcal{Q}(T)$ does not necessarily contain the same information than T. We thus consider now how the definition of $\mathcal{Q}(T)$ can be augmented so that the resulting condensed representation can be used to reconstruct the original tree.

For this, we shall associate integers with the edges of $\mathcal{Q}(T)$. Let us consider a vertex x of T and denote $n(x, J)$ the number of children of x that have class J:

$$n(x, J) = |\,\{z \in T | z \in \mathit{child}(x) \text{ and } c(z) = J\}\,|$$

To characterize each vertex x in the tree, we can count for each class J the number of children of x that have class J. This makes it possible to associate with each vertex x a *signature* defined as a multiset $\sigma(x)$.

Definition 2 (Signature of a vertex) *Let x be a vertex of T. We associate with x the multiset $\sigma(x)$ defined as:*

$$\sigma(x) = \{(J, n(x, J)),\ J \in \mathcal{Q}(T)\}$$

For unordered trees, it is natural to define the signature as a multiset. However, for different types of trees, this definition should be modified to adapt our approach. For example for an ordered tree, the signature of a vertex would naturally be defined by the ordered list of the classes of its children.

Signatures can be used to characterize recursively vertices having identical equivalence classes, based on the signatures of their children.

Proposition 2 $\forall x, y \in T$, $T[x] \equiv T[y] \Leftrightarrow \sigma(x) = \sigma(y)$.

Proof: If part. Let us consider two vertices x and y in a tree T, such that $c(x) = c(y)$. This means that $T[x]$ and $T[y]$ are isomorphic. Then the forest $F[x]$ and $F[y]$ are isomorphic as well and $\sigma(x) = \sigma(y)$.

Only if part. Let us consider two vertices x and y in a tree T, such that $\sigma(x) = \sigma(y)$. This means that the sets $child(x)$ and $child(y)$ have the same number of vertices and that the forest rooted respectively in the vertices of $child(x)$ and $child(y)$ are isomorphic. Therefore, the trees $T[x]$ and $T[y]$ are isomorphic as well. ■

Since the function σ is constant over a class I, we shall define by extension $\sigma(I)$ as $\sigma(x)$ for any x in class I.

Corollary 1 *Let x_1 and x_2 be two vertices with identical class I (*i.e. *$c(x_1) = c(x_2) = I$). Then for any class J, $n(x_1, J) = n(x_2, J)$.*

The quantity $n(x, J)$ is constant for any x in I, and is thus denoted by $n(I, J)$. This function, defined on the edges of $\mathcal{Q}(T)$, is called the *signature distribution* of T.

Definition 3 (Reduction of a tree) *Let T be a tree and $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$ be its quotient graph. The reduction $\mathcal{R}(T)$ of T is a graph $(V_\mathcal{Q}, E^+_\mathcal{Q})$, where $E^+_\mathcal{Q}$ is the multiset $\{((I, J), n(I, J))\}_{(I,J)\in E_\mathcal{Q}}$, n being the signature distribution of T.*

$\mathcal{R}(T)$ is thus the DAG $\mathcal{Q}(T)$ augmented with labels on its edges corresponding to $n(I, J)$. Intuitively, the reduction $\mathcal{R}(T)$ represents the tree T where all the structural redundancy of the tree has been removed. Fig. 1.b depicts the quotient DAG $\mathcal{Q}(T)$ of the tree of Fig. 1.a, while Fig. 1.c depicts its reduction $\mathcal{R}(T)$. In the sequel, we shall manipulate DAGs augmented with integer labels but call them DAGs for sake of simplicity. We shall denote $\mathcal{D}$ the set of such DAGs.

2.2 Computing tree reduction

The problem of constructing a compression of a tree has been raised in the early 1970's. For ordered trees, previous algorithms have been proposed to allow the reduction of a tree with complexities ranging in $O(n^2)$ to $O(n)$ (see [47] for a review).

We present hereafter two different algorithms to compute the reduction in the case of *unordered tree.* They are derived from completely different approaches but lead to the same time complexity (optimization procedures are not considered here).

2.2.1 Signature-based algorithm

Proposition 3 *The reduction $\mathcal{R}(T)$ of an unordered tree T can be computed in time $O(|T|^2 \deg(T) \log \deg(T))$.*

Proof: Let T be a tree. The proof consists of defining an algorithm that enables us to build $\mathcal{R}(T)$ from T. The algorithm proceeds from the leaves up to the root of T. We thus index the vertices $\{x_k\}_{k=1,|T|}$ of T such that the indexes are increasing from the bottom to the top of the tree, and are consistent with the ancestor relationship in T, *i.e.* $x_{k_1} \preceq x_{k_2} \Rightarrow k_1 \geqslant k_2$ (a post-fix numbering of the vertices for instance verifies this condition). Let us proceed by induction on k.

Initialization: let $R_0 = (V_0, E_0)$ represent the initial state of the reduction graph of T, with $V_0 = \varnothing$, $E_0 = \varnothing$. We also set up a list of signatures L_0, initially empty.

By induction, let us assume that $R_k = (V_k, E_k)$, which represents the state of the reduction graph at step $k \in [0, |T| - 1]$, was computed at step k, together with the associated list of signatures L_k. V_k is the set of classes and E_k is a multiset of edges between classes (already identified from the observation of the k first vertices in the list $\{x_i\}_{i=1,|T|}$).

Let us now consider vertex x_{k+1}. Two cases must be considered.

- If x_{k+1} is a leaf of T, then $I = c(x_{k+1})$ is the signature of a leaf. If it does not already exist in the signature list L_k (x_{k+1} is the first leaf encountered by the algorithm), then both classes and signatures must be updated : V_{k+1}, *i.e.* $V_{k+1} = V_k \cup I$, and $L_{k+1} = L_k \cup I$. Otherwise nothing is done. The time complexity of this operation is constant.

- If x_{k+1} has children $x_{k_1}, \ldots, x_{k_M}$ in the tree T, then, by induction, we know that the class of every child x_{k_j} has already been determined at a previous step since $k_j < k + 1, \forall j \in [1, M]$. $\sigma(x_{k+1})$ is thus well defined and can be compared to the signatures already stored at previous steps. Since the number of children of x_{k+1} is $\deg(T)$ in the worst case, the time complexity of each comparison is $O(\deg(T) \log \deg(T))$. Since there are at most k signatures already stored in L_k, $O(k)$ comparisons have to be performed. If the signature does not exist in L_k, both the set of classes and the signature list must be updated: $L_{k+1} = L_k \cup \sigma(x_{k+1})$, $V_{k+1} = V_k \cup c(x_{k+1})$ and $E_{k+1} = E_k \cup \{(c(x_{k+1}), c(x_{k_j}))\}$ for $j \in [1, M]\}$ (note that the number $n(c(x_{k+1}), c(x_{k_j}))$ is automatically updated by the update of the multiset E_{k+1}). Otherwise nothing is done.

  Since this operation is repeated for each vertex, the overall time complexity is $O(\sum_{k=1}^{|T|} k \deg(T) \log \deg(T)) = O(|T|^2 \deg(T) \log \deg(T))$.

■

The corresponding algorithm for tree reduction is depicted in Appendix II. By using a more efficient data structure (for instance a self-balancing binary search tree, [48]) to store the signature, the overall time complexity of this algorithm would fall to $O(|T| \deg(T) \log \deg(T))$.

2.2.2 Edit-distance-based algorithms

The above algorithm proceeds by detecting isomorphic subtrees in a bottom-up manner. It relies at each stage on the detection of exact isomorphisms between subtrees. Interestingly, a different approach, based on tree edit-distance, can be used to carry out similar detections with additional advantages. Various algorithms make it possible to compute approximate (ordered, unordered) tree isomorphisms and edit distance based on all-against-all subtree comparison [31, 49, 50]. The complexity of these algorithms is kept polynomial by the use of bottom-up recursion and dynamic programming. A null distance between two trees (or subtrees) denotes the existence of an isomorphism between the two structures. In our context, the key idea is to use these algorithms to compute the distance from a tree T to itself. Obviously the resulting distance is null, but as a by-product, all the distances between any two subtrees of T are computed recursively in close to quadratic time. Since a null distance between two subtrees denotes isomorphic structures, these algorithms can be exploited to build the reduction graph of T. As a counterpart of their slightly higher complexity ($O(|T|^2 \deg(T) \log \deg(T)$) for Zhang's algorithm for instance), they open the way to the extension of the reduction techniques presented in this paper to the detection of non-perfect isomorphisms and approximate tree reduction.

2.3 Properties of tree reductions

Remarkably, a tree and its reduction are actually equivalent:

Proposition 4 *A tree T can be exactly reconstructed from its reduction $\mathcal{R}(T)$.*

Proof: Omitted. A description of the algorithm computing a tree from a DAG is presented in Appendix III. ∎

Let A be a DAG, the tree obtained from A is denoted by $\mathcal{T}(A)$ (*i.e.* $\mathcal{R}(\mathcal{T}(A)) = A$), and the number of vertices of $\mathcal{T}(A)$ is denoted by $n(A) = |\mathcal{T}(A)|$.

Definition 4 (DAG partial order relations) *Let A and B be two DAGs, the partial order relations $\subseteq$ and $\sqsubseteq$ between trees induces respective partial order relations $\subseteq$ and $\sqsubseteq$ between DAGs:*

$$A \subseteq B \Leftrightarrow \mathcal{T}(A) \subseteq \mathcal{T}(B)$$

$$A \sqsubseteq B \Leftrightarrow \mathcal{T}(A) \sqsubseteq \mathcal{T}(B)$$

Let us consider two DAGs A and B, reductions of respectively two trees $\mathcal{T}(A)$ and $\mathcal{T}(B)$, then the distance $D(A,B)$ will represent the edit distance (introduced in the previous sub-section) between $\mathcal{T}(A)$ and $\mathcal{T}(B)$. This distance between DAGs has the following properties:

Proposition 5 *Let A, B and C be three DAGs:*

$$\begin{array}{rcl} A \subseteq B & \Leftrightarrow & D(A,B) = n(B) - n(A) \\ A \subseteq B \subseteq C & \Leftrightarrow & D(A,C) = D(A,B) + D(B,C) \end{array}$$

Proof: Let A and B be two DAGs such that $A \subseteq B$. Since $\mathcal{T}(A)$ is a subtree of $\mathcal{T}(B)$, any vertex of $\mathcal{T}(A)$ can be mapped onto a vertex of $\mathcal{T}(B)$ (there is only insertions). The number of vertices that are not mapped defines the distance between $\mathcal{T}(A)$ and $\mathcal{T}(B)$:

$$\begin{array}{rcl} D(A,B) & = & D(\mathcal{T}(A), \mathcal{T}(B)) \\ & = & |\mathcal{T}(B)| - |\mathcal{T}(A)| = n(B) - n(A) \end{array}$$

Let A, B and C be three DAGs verifying $A \subseteq B \subseteq C$:

$$\begin{array}{l} \left\{ \begin{array}{l} D(A,B) = n(B) - n(A) \\ D(B,C) = n(C) - n(B) \end{array} \right. \\ \Leftrightarrow D(A,B) + D(B,C) = n(C) - n(A) \\ \quad = D(A,C) \end{array}$$

∎

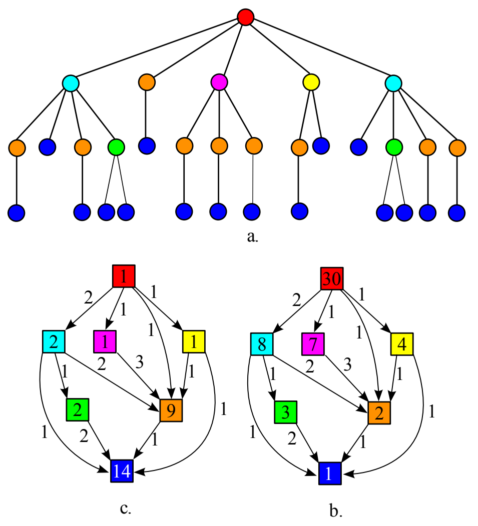

Figure 2: a. A tree T whose vertices are colored according to their signature. b. Corresponding reduction graph showing the value with $n(I)$ on each vertex. c. Reduction graph where vertices are valuated with $m(I)$. Both quantities are computed recursively.

Different quantities that correspond to different characteristics of the tree T can be directly computed on $\mathcal{R}(T)$. For any I in $\mathcal{R}(T)$, there exists a vertex x in T such that $I = c(x)$. We define $n(I) = |T[x]|$ the size of the tree rooted in x and $m(I)$=$|\{x \in T | c(x) = I\}|$ the number of trees in T isomorphic to the tree rooted in x.

Proposition 6 *Size $n(I)$ of a given subtree. For any I in $\mathcal{R}(T)$,*

$$n(I) = 1 + \sum_{J \in child(I)} n(I,J)n(J)$$

Note that for a leaf of T, child$(I) = \emptyset$, and then $n(I) = 1$.

Proposition 7 *Number $m(I)$of trees in T isomorphic to a given subtree of T. For any I in $\mathcal{R}(T)$,*
if parent$(I) = \emptyset$,

$$m(I) = 1$$

if parent$(I) \neq \emptyset$,

$$m(I) = \sum_{K \in parent(I)} n(K,I)m(K)$$

Fig. 2 illustrates the computation of $n(I)$ and $m(I)$. $n(I)$ is computed bottom-up and $m(I)$ is computed top-down.

Since a tree reduction corresponds to a compacted version of the original tree, we are interested in quantifying corresponding reduction factors.

Definition 5 (Compression factors) *Let $n_v(I)$ and $n_e(I)$ be respectively the number of vertices (resp. of edges) of the sub-DAG of $\mathcal{R}(T)$ rooted in I. The vertex compression factor is defined by*

$$\rho_v(I) = 1 - \frac{n_v(I)}{n(I)} \tag{1}$$

Similarly, the edge compression factor is defined by:

$$\rho_e(I) = 1 - \frac{n_e(I)}{n(I) - 1} \tag{2}$$

Definition 6 *Let G be a DAG. We define $h(G)$ as the maximum length of a path in G*

Proposition 8 *Let T be a tree and $\mathcal{R}(T)$ its reduction. Then, $h(T) = h(\mathcal{R}(T))$*

Proof: Every path in $\mathcal{R}(T)$ corresponds to a path in T and reciprocally. Therefore, the path with maximum length in T corresponds to a path with maximum length in $\mathcal{R}(T)$ ■

3 Self-nested trees

We are now going to consider particular trees, whose reductions show remarkable compression properties.

3.1 Definition

Let us first give a recursive definition of tree self-nestedness.

Definition 7 *[Self-Nested tree] A tree T rooted in r is self-nested either*

- *if it is a single leaf*
- *or if all the trees of $F[r]$ are self-nested and one of them contains the others as subtrees.*

Based on this definition, self-nested trees can be characterized as follows:

Proposition 9 *Let T be a tree. The following propositions are equivalent:*

- *T is a self-nested tree*
- *all the subtrees of T with identical height are isomorphic:*

$$\forall x, y \in T, h(x) = h(y) \Leftrightarrow T[x] \equiv T[y]$$

- *any two subtrees of T are either isomorphic or included one into another (one is a subtree of the other):*

$$\forall x, y \in T, T[x] \sqsubseteq T[y] \;\; or \;\; T[y] \sqsubseteq T[x]$$

Proof: The proof which presents no particular difficulty is omitted. ■

In the sequel, we shall denote $\mathcal{S}$ the set of self-nested trees and $\mathcal{S}^+(T)$ the set of all self-nested trees that contain T:

$$\mathcal{S}^+(T) = \{S \in \mathcal{S} | T \subseteq S\}$$

Now, let us consider the reduction of self-nested trees and their properties.

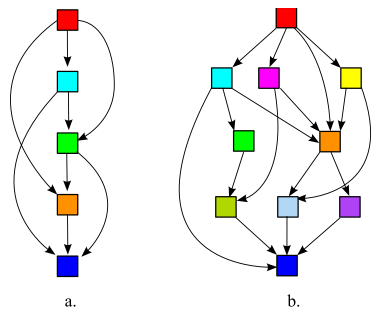

Figure 3: a. a linear DAG. b. a non-linear DAG

3.2 Reduction of self-nested trees

Definition 8 *[Linear DAG] A linear DAG is a DAG containing at least one path that goes through all its vertices.*

The difference between a linear and a non-linear DAGs is illustrated in Fig. 3. We shall denote $\mathcal{L}$ the set of all linear DAGs and $\mathcal{L}^+(T)$ the set of linear DAGs that contain $\mathcal{R}(T)$, T being a tree.

Linear DAGs are tightly connected with self-nested trees. This is expressed by the following property:

Proposition 10 *A tree T is self-nested if and only if its reduction $\mathcal{R}(T)$ is a linear DAG.*

Proof: This proposition is an immediate consequence of proposition 9. In particular, note that a vertex in a linear DAG is determined without ambiguity by its height. ■

Fig. 4a. , b. and c. show different configurations of self-nested trees and their respective reductions. In Fig. 4d. a non self-nested tree and its reduction, a non-linear DAG, are illustrated.

From proposition 8 we know that a tree T of height H has a reduction containing at least $h(T) = H$ vertices ($h(\mathcal{R}(T)) = H$). Since the reductions of self-nested trees of height H are linear graphs (prop. 10) of height H (prop. 8), these DAGs have exactly H vertices. Self-nested trees thus achieve maximal compression rates with respect to other trees with the same height H. To take into account this remark, we can modify the definition of the compression factor, $\rho_v(T)$, so that self-nested trees have 100% compression rates by definition. We therefore need to correct the definition of eq. 1 as :

$$\rho'_v(T) = 1 - \frac{n_v(\mathcal{R}(T)) - h(T)}{n(T)}. \qquad (3)$$

3.3 Determination of the nearest self-nested tree of a tree

The previous compression factor gives us a first account of the compressibility of a tree. However, this quantity does not reflect the exact compression of information as it only takes into account the number of vertices of the DAGs. Similarly, a coefficient based on edges would be also incomplete and combinations of both would lack a theoretical justification. In this section we are

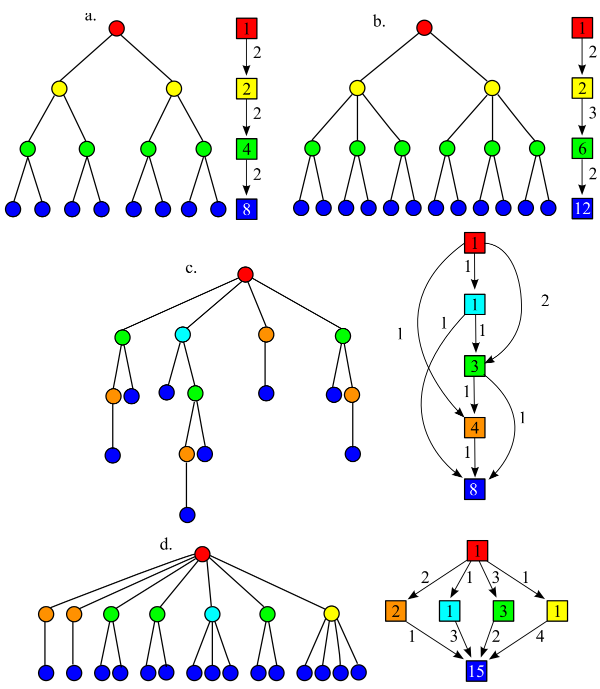

Figure 4: Examples of self-nested trees and their reduction graphs

interested in the problem of defining a measure reflecting the compressibility of a tree on a theoretically sound basis.

For any tree T, the number of vertices of its reduction is bounded by its height $h(T)$. For self-nested trees, this number is exactly $h(T)$, *i.e.* $\mathcal{R}(T)$ has exactly $h(T)$ vertices. Therefore, self-nested trees achieve a maximal vertex compression rate. We can therefore intuitively think of the compressibility of any tree T a measure of the distance at which the tree T is from perfect self-nestedness. This would define a degree of self-nestedness for T. Let us put this idea in formal terms.

Let us consider a distance D defined on $\mathcal{T}$ (the set of all trees). Let us call $\mathrm{NST}(T)$ the set of self-nested trees with minimal distance to T:

$$\mathrm{NST}(T) = \underset{S\in\mathcal{S}}{\mathrm{argmin}} D(T,S) \tag{4}$$

In general for topological distances, there exists more than one self-nested tree S^* with minimal distance $D(T,S^*)$. This quantity characterizes how distant the tree T is from the nearest self-nested tree and therefore is a good candidate to quantify the degree of self-nestedness of T.

In equation 4, D can be any distance between two trees. A particular choice of D corresponds to different definitions of the NST of a tree. In this paper, we are interested in edit-distances corresponding to mappings that preserve certain structural properties between the compared trees, e.g. Zhang's distance. The definition of the NST can also be modified by solving equation 4 over self-nested sets with different characteristics. For example, the nearest self-nested tree of a given tree T can be looked for in the set of the self-nested trees that contain T, or in the set of self-nested-trees that are contained in T.

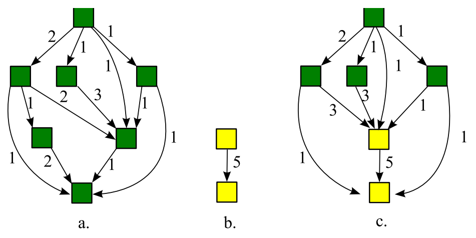

Figure 5: a. a DAG G b. a linear DAG L such that $h(L) \leq h(G)$. c. DAG $G\langle L\rangle$ corresponding to the DAG G partially linearized by L.

In our context, due to biological motivations detailed in section 4, we are interested in carrying out the optimization process over the set denoted $\mathcal{S}^+(T)$ of self-nested trees that contain T, *i.e.* that can be obtained from T by inserting nodes only:

$$\mathrm{NEST}(T) = \operatorname*{argmin}_{S \in \mathcal{S}^+(T)} D(T, S) \tag{5}$$

In the sequel, we are going to show that a solution of this optimization problem can be found in polynomial time for any given tree T. We chose to develop and illustrate the reasoning on DAGs, although a reasoning on the dual space of trees would have also been possible. In particular, proposition 5 can be expressed in terms of DAG optimization: if $G = \mathcal{T}$, let denote

$$A^*(T) = \operatorname*{argmin}_{A \in \mathcal{L}^+(T)} D(G, A)$$

then

$$\mathrm{NEST}(T) = \mathcal{T}(A^*)$$

Let us consider a tree T and its reduction $G = \mathcal{R}(T)$. Let us also consider a linear DAG L such that $h(L) \leq h(G)$.

Definition 9 *[DAG partially linearized by a linear DAG] We denote by $G\langle L\rangle$ the DAG obtained by modifying G with L as follows:*

- *remove all the vertices of height $k = 1..h(L)$ in G*
- *for each pending edge, which used to connect a vertex v of height $K > h(L)$ to a removed vertex of height k in G, connect v to the unique vertex of L of height k by a new edge with the same edge label.*
- *in the obtained DAG, if several edges appear between two vertices, replace them by one edge whose label is the sum of the edge labels.*

The figure 5 illustrates the partial linearization $G\langle L\rangle$ of a DAG G by a linear DAG L.

Let K be an integer. We denote $\mathcal{S}_K^+(G)$ the set of partially linearized DAGs by linear DAGs of height K, $K \leq h(G)$:

$$\mathcal{S}_K^+(G) = \{G\langle L\rangle \,|L \in \mathcal{L}, h(L) = K, \text{and such that:} G \subseteq G\langle L\rangle\}$$

All the DAGs of this set contain G. We consider the subset of $\mathcal{S}_K^+(G)$ of DAGs with the smallest number of vertices and let Z_K be an element of this subset:

$$Z_K \in \operatorname*{argmin}_{H \in \mathcal{S}_K^+(G)} \{n(H)\}$$

Note that Z_K also minimizes the distance $D(G,H) = n(H) - n(G)$ for $H \in \mathcal{S}_K^+(G)$. In particular, $Z_{h(G)}$ is a linear DAG that minimizes this distance and therefore corresponds to a solution of equation 5. To compute the NEST of G (*i.e.* equivalently the NEST of T), we are thus going to show that $Z_{h(G)}$ can be computed in polynomial time. The following proposition shows that this computation can be carried out recursively.

Proposition 11 *For any* $K = 1..h(G)$,

$$\begin{aligned} D(G, Z_K) &= \min_{H \in \mathcal{S}_K^+(G)} D(G,H) \\ &= D(G, Z_{K-1}) + \min_{H \in \mathcal{S}_K^+(Z_{K-1})} D(Z_{K-1}, H) \end{aligned}$$

Proof: A detailed proof of this proposition is given in Annex 1. ■

The computation of $D(G, Z_K)$ makes use of the dynamic programming principle frequently used in discrete optimization problems, *e.g.* [51]. The optimal solution at stage K, *i.e.* the best linearized graph at height K, is a function of the optimal solution at stage $K-1$ and of a local optimization to pass from the optimal solution at stage $K-1$ to the optimal solution at stage K (right-hand member of the recursive equation). Interestingly, this local optimization problem (*i.e.* $\min_{H \in \mathcal{S}_K^+(Z_{K-1})} D(Z_{K-1}, H)$) is expressed in a way similar to the global optimization problem (*i.e.* $\min_{H \in \mathcal{S}_K^+(G)} D(G,H)$), in which the DAG G has been substituted by the linear DAG Z_{K-1}.

From a DAG perspective, this local optimization problem comes down to finding the smallest DAG that embeds all the subDAGs of Z_{K-1} rooted respectively in the vertices w_i at height K of the DAG Z_{K-1}. This problem is equivalent in the dual tree space to finding the smallest common super-tree of all the trees corresponding to the subDAGs rooted in w_i. This remark can be exploited to solve the local optimization problem.

Proposition 12 *The local optimization problem* $\min_{H \in \mathcal{S}_K^+(Z_{K-1})} D(Z_{K-1}, H)$ *can be solved in time* $O(\deg(Z_{K-1}))$

Proof: A detailed proof of this proposition is given in Annex 1. ■

Based on propositions 11 and 12, we can finally derive our main result:

Theorem 1 *The NEST optimization problem (equation 5) can be solved in time* $O(h(G)^2 \times deg(G))$

Proof: The proof of theorem 1 is based on a constructive approach of the solution. We describe here the corresponding *NEST* algorithm.

Let us denote G the reduction of a tree T. The NEST algorithm consists of computing recursively a sequence of DAGs $Z_K \in \mathcal{S}_K^+(G)$, for $K = 1..h(G)$,

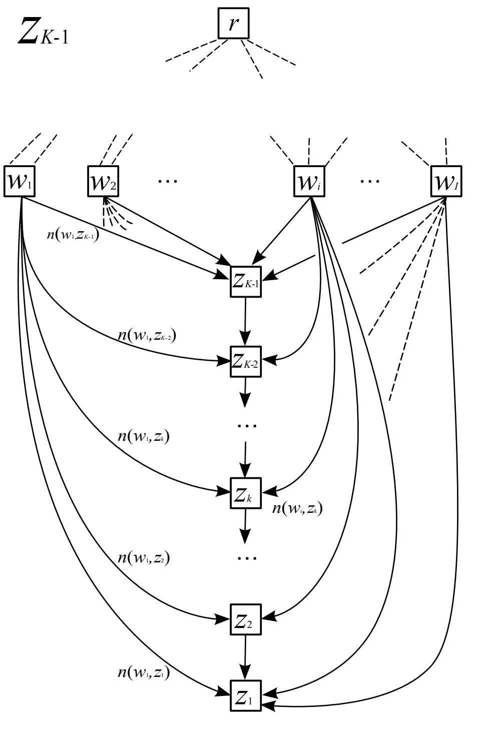

Figure 6: DAG Z_{K-1}. The minimization $D(Z_{K-1}, H)$ for $H \in \mathcal{S}_K^+(Z_{K-1})$ consists of merging all the vertices at level K in a minimal way to obtain a new DAG in $S_K^+(G)$, *i.e.* linearized up to height K

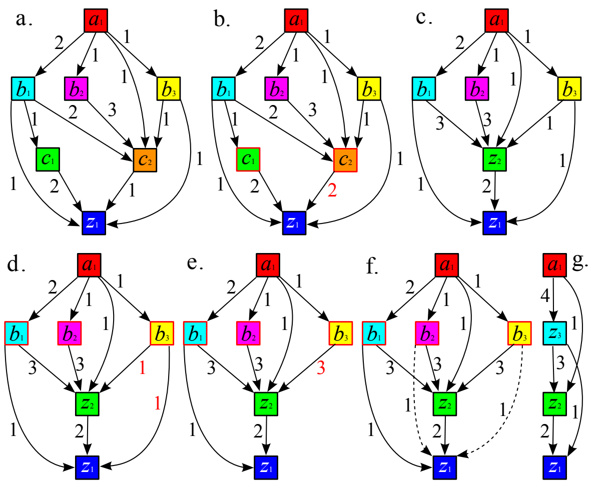

Figure 7: The seven steps of the algorithm needed to find the nearest linear DAG of the DAG introduced in Fig. 2

starting from $Z_0 = G$. Each Z_K has a linear part of height K and the final DAG $Z_{h(G)}$ is a linear DAG. The NEST of G is recursively computed from the leaves to the root of G. At a given step of the computation, we suppose that all the nodes z_i $(1 \leq i \leq K-1)$ and the edge weights $n(w_l, z_i)$ have been determined (Fig. 6). We then consider the calculation of z_K and the edges between z_K and all the nodes z_i. This local optimization problem is solved in $O(\deg(Z_{K-1}) \subset O(\deg(G))$ (prop. 12). However, in order to fully determine Z_K, for any $i \in \{1..K-1\}$, $n(z_K, z_i)$ must be updated according to the local optimization problem solution. These values are thus computed from z_{K-1} to z_1, for each w_l, the weight between w_l and any z_i is updated as follows:

- if the sum of weights between w_l and any z_i is smaller than the maximum value, then all the weights between w_l and z_i for $i < K-1$ become equal to 0 and $n(w_l, z_{K-1})$ is set to this maximum value,
- otherwise, the $n(w_l, z_i)$ are set to the current $n(w_l, z_{K-1})$.

In the worst case, the update is repeated for each vertex at height K, *i.e.* in time $O(\deg(Z_{K-1}) \times K) \subset \deg(G) \times K)$. Finally, the nodes w_l are removed and replaced by a node z_K such that $n(z_K, z_i)$ is equal to the maximum value of $n(w_l, z_i)$. The recursive computation is repeated for each K in $1..h(G)$ which leads to the overall complexity: $O(\sum_{K=1}^{|h(G)|} K \deg(G)) = O(h(G)^2 \times \deg(G))$. ■

To illustrate the NEST algorithm, Fig. 7 presents the different steps that occur during its application to a particular a DAG:

- Let us consider (Fig. 7a.) the DAG introduced in Fig. 2. The bottom vertex (of height 0) is called z_1, which initializes the recursion.
- At a first stage, we consider all the vertices at height 1, in this case c_1 and c_2 7b.). The value $n(c_i, z_1)$ of edges pointing from the vertices c_i to the extremity z_1 of the DAG are compared and the maximum of these values is chosen (2 in this example).

- The edge values $n(c_i, z_1)$ are then updated to this maximum. The resulting DAG contains two equivalent vertices that can be collapsed in a new node z_2, Fig. 7c.
- So far, the original DAG has been transformed into a partially linear DAG at height 1 and 2 that contains the original DAG itself and such that a minimum number of vertices has been added to the corresponding tree. We then iterate these two previous steps at height 2.
- The values $n(b_i, z_2)$ of edges from vertex b_i (Fig. 7d.) to the vertex z_2 are compared to identify the maximum (3 in this example). All edges (b_i, z_2) must be homogenized using this maximum value. Only $n(b_3, z_2)$ which is equal to 1 must be updated. To add a minimum number of vertices to the underlying tree, we first use the complete tree(s) available from edge (b_3,z_1) to participate in the increase of $n(b_3, z_2)$. Only $n(b_3, z_1) = 1$ tree is available. The vertices of this tree are removed from the underlying tree, and used to create a new tree on edge b_3, z_2 leading us to $n(b_3, z_2) = 2$. $n(b_3, z_1)$ is now zero (the edge between b_3 and z_1 can be removed. Now, additional vertices can be added to augment the number of trees on edge b_3, z_2 by one unit, leading to $n(b_3, z_2) = 3$,(Fig. 7e.).
- The edge homogenization procedure for vertices at height 2 is then repeated for each vertex z_i from z_2 to z_1, actually at this stage only for z_1. The edge values $n(b_i, z_1)$ are compared to find the maximum value (1 in this example). Values of $n(b_i, z_1)$ are augmented to reach this maximum (dotted edges in Fig. 7f.).
- Finally, the DAG obtained contains three equivalent vertices at height 2. These vertices can be collapsed into a new vertex z_3 leading us to a linear DAG, *i.e.* the Nearest Embedding Linear DAG (Fig. 7g.).

Let T^* be an element of $NEST(T)$. We define the degree of self-nestedness of T, $\delta_{NEST}(T)$, as the percentage of vertices occupied by T in T^*. Since $T \sqsubset T^*$, $\delta_{NEST}(T)$ can be defined by :

$$\delta_{NEST}(T) = 1 - \frac{D(T, T^*)}{|T^*|}. \tag{6}$$

This measure is independent of the particular element T^* chosen in $NEST(T)$. Indeed, for any tree T^* in $NEST(T)$, $T \sqsubseteq T^*$ and then $D(T, T^*) = |T^*| - |T|$. Since by definition $D(T, T^*)$ is constant over this set, two different trees in $NEST(T)$ must have the same number of vertices, and therefore $\delta_{NEST}(T)$ is independent of the choice of T^* in $NEST(T)$.

4 Application to the structural analysis of plants

Internal similarities and nested structures in plants have for long been studied by botanists to understand the general organization of plants and the dynamics of their development [20, 52, 1, 2, 21, 53]. Different techniques were used until now ranging from qualitative analysis, with the help of descriptions based on botanical drawing or pictures, to varying levels of quantitative analysis. However, first techniques for the systematic analysis of the internal similarities of plant branching systems have been proposed only recently [19, 18]. Such techniques are important for two major reasons:

- First, they may be used to reveal structures of plants deeply hidden in a complex branching system. If all the redundancy that is expressed in the branching structure of a plant is removed, we would be likely to characterize *the deep structure* of the plant. Being more simple, this structure could be easier to interpret, to characterize or to compare with the corresponding structure of other plants.

- Second, the fact that plants are made up of the repetition of many similar components at different scales provides macroscopic presumptions for the existence of similarities in processes that drive meristem activity at microscopic scales. Thus characterizing the internal similarities of plants is expected to give important clues to understand meristem growth.

From a biological perspective, since the pioneering work of Goethe on plant metamorphosis [54, 55], repetitions and gradients in plants are supposed to express the idea that the meristems of a plant undergo series of differentiation stages throughout their lives and that these differentiation routes derive from a unique common differentiation process. In the last decades, this hypothesis was applied to the study of various plant architectures by several teams of botanists who developed and confirmed its unifying ability, e.g. [1, 56, 52, 57, 23, 53]. Recently, a new important stage was reached in the support of this hypothesis by Prusinkiewicz's and Coen's groups [58] who developed a first plausible physiological model of inflorescence meristem differentiation (measured as vegetativeness) based on molecular genetic studies. The model was shown to explain various types of inflorescence architectures found in nature and their association with particular climate and life history during evolution.

In this section, we show that the notion of self-nestedness can be used as a new tool to investigate quantitatively, and from a macroscopic perspective, the differentiation stages followed by meristems in a plant. In the spirit of the previous works on plant architecture analysis, we assume that the organization of macroscopic structures in plants reflects (at least partially) processes at a more microscopic scale characterizing the states of meristems during ontogeny. As a first approximation of this connection, we shall rely on the following simplifying and explicit assumption:

Hypothesis 1 (Continuous developmental ability) *If two branching structures in a plant are similar, they were produced by meristems with similar differentiation states.*

In other terms, if we consider the function that associates each meristem with the branching structure it produces, this hypothesis states that this function, expressing the developmental ability of meristems, should be continuous. In the sequel, we shall show that it is possible to use this idealized - but useful - hypothesis to organize the multitude of meristem states by classes of equivalence with respect to the similarity of what they produce.

The hypothesis of continuous developmental ability implicitly requires that metrics are defined on both the branching system space and the meristem state space. To compare branching structures we use a metric based on edit distances. In the paper we use the metric based on Zhang's edit-distance algorithm between unordered trees [31]. However, other metrics could be used, taking into account other types of mapping constraints between the tree structures. Similarly, we also consider only topological distances based on the use of binary local distances between plant components. The component shape for instance is not taken into account.

To study the self-nestedness of plants, we shall use the paradigm of meristem differentiation as follows. We assume that each plant meristem is potentially able to produce a maximal self-nested structure, depending on its differentiation stage. However, the actual development conditions of plant meristems (light environment, water or nutrient stress, pest diseases, accidents, etc.) modify this optimal production by altering the ability of certain meristems to develop. In this context, an actual plant can be considered as an altered version of a self-nested plant, where some components are missing. We will therefore naturally quantify the degree of self-nestedness of a plant T by computing the distance between T and the smallest self-nested plant that contains it, *i.e.* in $\mathcal{S}^+(T)$.

We first apply the method to a set of simulated theoretical plants to assess its performances on plants with controlled architectures. In particular, based on the hypothesis of continuous developmental ability, we test the ability of the method to retrieve the theoretical meristem states that were used to generate each plant. The method is then applied to the architecture analysis of a real plant. It enables us to compute a compressed version of the plant, its NEST, and to derive hypothetical meristem states for all its internal branching systems, under the hypothesis of continuous developmental ability.

Analysis of theoretical plants

Creation of the plant database

A database of plants corresponding to 3 contrasted types of architecture was created. Four models, denoted by M_0, M_1, M_2 and M_3, were designed to create plants of this database with a procedure similar to that described in [19] and that is briefly recalled hereafter.

In each model we assumed that the apical meristem of the main axis progresses through a sequence of morphological differentiation states, from germination to the flowering state. The set of states is ordered and an apical meristem in state s produces a branch segment whose characteristics (geometry, color) are defined by s and can only pass in a state greater than or equal to s. The state of the apical meristem thus gradually increases from the initial state s_0 until the apex reaches the final state and becomes a terminal organ (a flower). At each step, in addition to apical productions, the apical meristem of any axis can produce lateral meristems. When a lateral meristem is created, its initial state must be equal to or greater than that of the apical meristem that created it according to the model specification. A branch refers to a maximal sequence of segments produced by a given meristem in a plant. See [9] for complementary details.

We visualize the above process using *differentiation graphs* [19] that show the set of states and two types of possible transitions between them (apical and lateral). Colored *circles* represent differentiation states. *Solid arrows* represent

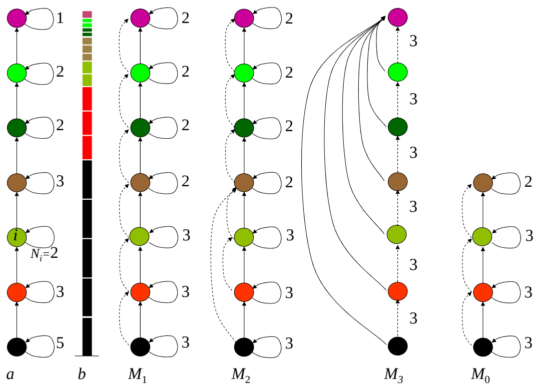

Figure 8: Differentiation graphs used for the definition of theoretical plants. a) The differentiation graph of a non-branching plant structure. b) The resulting axis structure, where component colors correspond to the differentiation graph states in which these components were created. The numbers attached to each loop indicate the number of steps a meristem stays in the corresponding state. Right: The differentiation graphs of models M_1, M_2, M_3 and M_0. Solid arrows correspond to possible transitions of the apical meristem states. Dashed arrows correspond to possible transitions from the apical meristem state to the axillary meristem states.

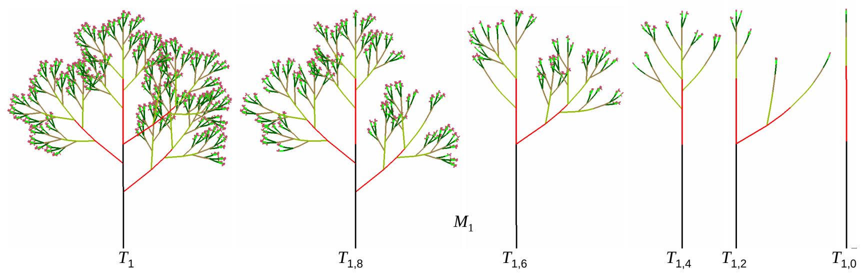

Figure 9: Deterministic model T_1 and 5 of its random versions from the plant samples $\mathcal{T}_{1,8}$, $\mathcal{T}_{1,6}$, $\mathcal{T}_{1,4}$, $\mathcal{T}_{1,2}$, $\mathcal{T}_{1,0}$

possible state changes of the apical meristem during the *apical growth* of an axis. The meristem stays in the same state for the number of steps indicated by the label associated with a loop, then progresses to the next state. For example, the differentiation graph of Fig. 8.a corresponds to the axis shown in Fig. 8.b. At each step, apical meristems may produce lateral meristems as indicated by the *dashed arrows*. The state transitions represented by these arrows relate the state s of the apical meristem with the state s' of the lateral meristem. Differences in these transitions are the key feature distinguishing the three types of models M_1, M_2 and M_3, discussed next.

The differentiation graph of each plant model M_1, M_2, M_3 has seven states, with 1 denoting the initial state and 7 denoting the terminal (flowering) state.

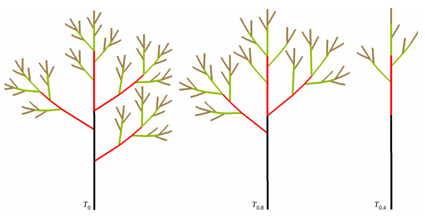

Figure 10: The $\mathcal{T}_0$ family: (a) template plant T_0 and random trees from (b) $\mathcal{T}_{0,8}$ and (c) $\mathcal{T}_{0,4}$.

The model M_0 is similar to M_1 with a smaller number of differentiation states. The differentiation graphs of the deterministic models M_1, M_2 and M_3 are shown in Fig. 8. In model M_1, the lateral meristems that are generated by an apical meristem in state s have state $s+1$. The state of the apical meristem remains unchanged for the given number of steps, then advances by 1 (except for the final state). Model M_2 differs from M_1 in that some lateral meristems produced by the apical meristem in state s may assume state s' greater than $s+1$. For example, the apical meristem in state 1 produces a lateral meristems directly in state 3. In model M_3, a meristem in state s produces 3 lateral meristems in states $s' = s+1$, but there is no gradual progression of states along either the main or the lateral axes. Instead, at each step, apical meristems directly differentiate into flowers.

For each model M_i, we generated a template plant T_i in a deterministic way and a set of 10 other derived plant samples, labeled $\mathcal{T}_{i,0}, \mathcal{T}_{i,1}, \dots, \mathcal{T}_{i,9}$, by randomizing the functioning of the lateral meristems. With probability p, a lateral meristem was allowed to develop into a branch; otherwise, the branch was aborted. This probability p is indicated in the plant sample name: $\mathcal{T}_{i,0}$ for $p = 0.0$, $\mathcal{T}_{i,1}$ for $p = 0.1$, and so on. Each sample $\mathcal{T}_{i,j}$ contains $K = 10$ individuals generated from M_i with constant branching probability $p = j/10$. Fig. 9 shows the template plant M_1 and 5 randomized trees obtained using different branching probabilities. The whole set of plants generated from a model M_i defines the $\mathcal{T}_i$ family. Fig. 10 and 11 respectively show similarly trees from the $\mathcal{T}_0$ and the $\mathcal{T}_2$, $\mathcal{T}_3$ families.

According to this design, the template plants T_1 and T_2 have a well defined hierarchy of branches with a marked trunk (they would correspond to 'monopodial' plants). Their lateral branches repeat parts of the main stem structure; thus, structures T_1 and T_2 are self-nested in the sense of definition 7. T_3 illustrates a different type of self-nested plant where the trunk itself is not repeated while the branching sequences are (this would correspond to 'sympodial' plants). The random removal of branches in these structures introduces variations that are expected to reduce their degree of self-nestedness.

The plants were generated using the L-system-based modeling program `cpfg` [59], incorporated into the plant modeling software `L-studio/VLab` [6, 60]. Structure generation begins with a single shoot apical meristem (emerging from the seed) and proceeds in a sequence of simulated developmental steps. At every step, the meristem adds a growth unit to the plant axis and changes its state according to its differentiation graph. Each growth unit supports a lateral meristem, which

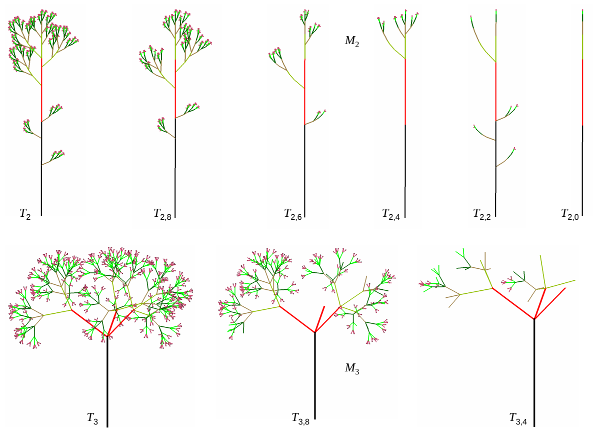

Figure 11: Top row: template plant T_2 and trees in $\mathcal{T}_{2,i}$ for $i = 8, 6, 4, 2, 0$. Bottom row: T_3 and 2 of its random versions from $\mathcal{T}_{3,8}$, $\mathcal{T}_{3,4}$.

Table 1: Average tree statistics of tree families $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ and size of the tree reductions. MaxOrder is the average maximum branch order in a tree and Branch Nb is the average total number of branches in a tree.

Proba.	Tree Size				MaxOrder				Branch Nb				$\|\mathcal{R}(T)\|$			
	$\mathcal{T}_0$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_0$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_0$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_0$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$
0.0	11	17	17	1	1	1	1	1	0	0	0	0	11	17	17	1
0.1	18	34	31	2	2.2	2.5	2.5	1.4	1.5	1.2	1.8	0.9	12.3	22.5	21.2	1.4
0.2	29	71	50	4	2.9	4.3	3.5	2.1	2	2.8	3.1	1.5	16.8	34.4	36	2.1
0.3	31	108	102	4	3.3	5	5	1.9	3.1	3.6	4.1	1.5	15.9	41.4	38.7	1.9
0.4	45	221	119	25	3.7	6.2	5.1	3.9	3.3	5.4	5.5	1.8	19.8	65.5	40	5.2
0.5	69	522	224	48	3.9	6.7	5.3	4.6	5	8.7	7.3	2.1	23.4	104.4	54.1	8.1
0.6	83	974	330	95	4	7	5.8	5.4	5.2	10.1	8.8	2.4	22.4	143.4	61	11.5
0.7	102	1336	582	176	4	7	5.9	5.8	6.6	10.9	10.7	2.4	24.1	149.6	77.8	15.1
0.8	141	2318	813	376	4	7	6	5.8	7.5	11.8	12.8	2.4	25.9	173.5	79.2	19.1
0.9	165	3787	1176	651	4	7	6	7	7.8	13.6	14.2	3	22.8	170.9	73.8	20.5
1.0	191	5183	1538	1093	4	7	6	7	9	15	15	3	11	17	17	8

may give rise to a new lateral axis. This process repeats for higher-order axes, resulting in the formation of a branching structure (see [9] for further details).

Reduction of theoretical trees

We first computed the tree reduction $\mathcal{R}(T)$ of each tree T in the database, Table 1. Fig. 12 depicts three DAGs corresponding to these tree reductions for trees of the $\mathcal{T}_0$ family. Tree reductions are obviously linear DAGs for all the deterministic trees T_i, $i = 0, 1, 2, 3$ and their absolute compression factors, ρ_v as defined by eq. 1, are very close to 1 since the number of vertices of $\mathcal{R}(T)$ is equal to $h(T)$ which is much smaller than $|T|$ (Fig. 13). Then, as the branching probability decreases, the vertex compression rate decreases as well, expressing a loss of compressibility of the trees with decreasing values of p. Interestingly,

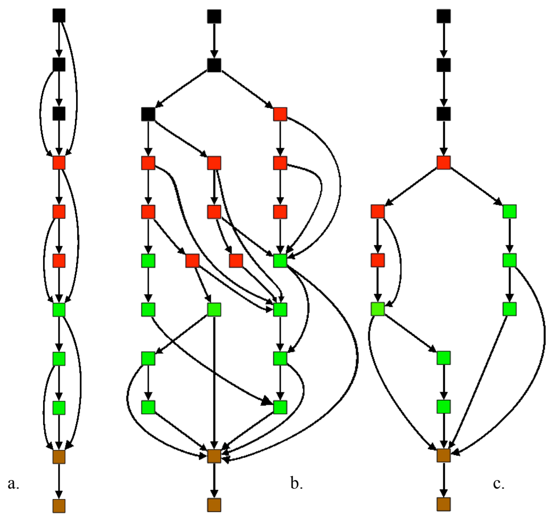

Figure 12: DAGs corresponding to the reductions of tree individuals in the families a. $\mathcal{T}_0$, b. $\mathcal{T}_{0,4}$ and c. $\mathcal{T}_{0,8}$.

this decrease is not linear and we can observe for example that a plant in $\mathcal{T}_1$ or $\mathcal{T}_2$ families with 40% of aborted branches ($p = 0.6$) can still be compressed by more than 75%. From eq. 1, we can see that the loss of compressibility is caused by the combined effect of both a degraded self-nestedness of the random trees ($|\mathcal{R}(T)|$ increases) and the decreasing size $|T|$ of the tree. In the extreme case where $p = 0$, $\mathcal{T}_i$ contains a single tree $T_{i,0}$, and $\mathcal{R}(T_{i,0})$ is isomorphic to $T_{i,0}$, and the compression factor is thus $\rho_v(T_{i,0}) = 0$. The effect of the tree size can be suppressed by using the alternative relative definition of the compression factor in eq. 3 (see relative coefficients in Fig. 13). For all tree families except M_3, we can see that for high probabilities both the absolute and relative compression factors are similar and decrease roughly linearly. Then, the relative compression factor gets significantly different from the absolute compression factor around $p = 0.4$, where the effect of the size of the tree starts to appear. For probabilities lower than $p = 0.3$ the relative compression factor increases again due to the fact that the plants sizes get closer and closer to $h(T)$. When $p = 0$, $|\mathcal{R}(T)| = h(T)$ and $\rho'_v(T_{i,0}) = 1$, showing that the compression of a tree with this height could not be better.

NEST of theoretical trees

For each tree T in the database, an element T^* of $NEST(T)$ was computed using the NEST algorithm. For template plants T_i, T^* is T_i itself and their reduction DAG, $\mathcal{R}(T_i)$, is linear. In this case, the structure of $\mathcal{R}(T_i)$ reflects exactly the structure of the original differentiation graph M_i that was used to generate the corresponding theoretical template trees. The loops on vertices of M_i appear in an expanded way in the computed DAG, where the corresponding vertices have been repeated in the graph as specified by the loop label in M_i, Fig. 14. Note however that no difference is made between the edge types in

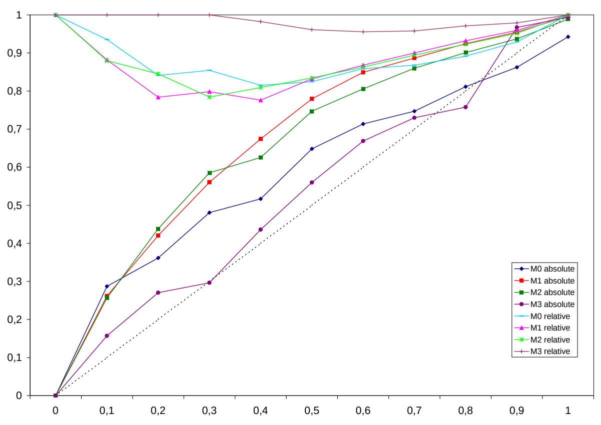

Figure 13: Average vertex absolute and relative compression factors ($\rho_v(T)$ and $\rho'_v(T)$) of the trees in the $\mathcal{T}_i$ families, $i = 0, 1, 2, 3$ (standard deviations are not shown here). Curves passing through the point (0,0) correspond to the absolute coefficients.

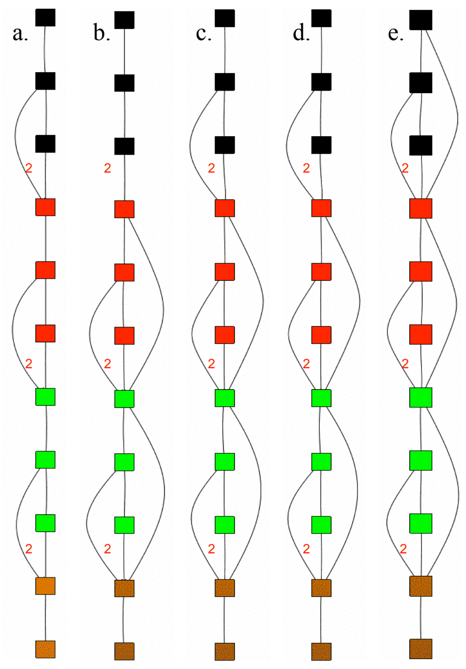

Figure 14: $\mathcal{R}(NEST(T))$ corresponding to a. both T_0 and particular instances of b. $T_{0,8}$, c. $T_{0,6}$, d. $T_{0,4}$ and e. $T_{0,2}$.

the reconstructed $\mathcal{R}(T_i)$ as we did not consider different types of edges in our approach.

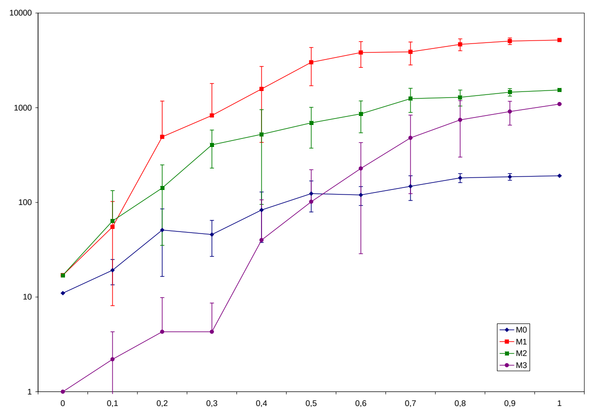

Figure 15: Average number of vertices in $NEST(T)$ for trees of the different families as a function of the branching probability.

For randomized plants, the structure of the NESTs could not be analyzed directly from the reduction DAGs that were too complex. Instead, we investigated different global aspects of these NESTs. First, their average size was estimated over each plant sample $\mathcal{T}_{i,j}$. When the branching probability decreases, the size of T^* tends to decrease in each tree family, Fig. 15. The difference between the number of vertices in T^* and T is reflected by the degree of self-nestedness, Fig. 16, which can be interpreted as the percentage of nodes of T that cover the NEST. This percentage is minimum for intermediate probabilities, showing that the degree of self-nestedness is lower for structures that are moderately perturbed compared with the template plant. If the perturbation is too strong, then the decrease of the plant size counter-balances the perturbation and the degree of self-nestedness gets higher again. For all the plants in the database, we observe that a degree of self-nestedness greater than 0.5 ($NEST(T)$ contains no more than 50% extra vertices than T) is achieved by trees of families $\mathcal{T}_{i,j}$ with either $j < 1$ or $j > 8$, for any $i = 0, .., 3$. This shows that the degree of self-nestedness is particularly sensitive to the perturbation intensity (here represented by p) of the template plants.

Then, to quantify how much the NEST structure of a tree is resistant to noise, we computed the number of times the NEST of a randomized plant in $\mathcal{T}_i$ was isomorphic to its original template plant T_i, Fig. 17.a. Surprisingly, the correct original template tree (without noise) was identified as the NEST of a randomized tree T for many perturbed trees. This suggests that the NEST structure is rather robust to the introduction of 'noise'. For plants of $\mathcal{T}_{0,8}$ and $\mathcal{T}_{1,8}$ for example, the NEST corresponded to the template plant for more than 60% of the trees, thus expressing a high degree of redundancy in these plants. However, when the 'noise' increases (*i.e.* p decreases), less and less trees in the database have a NEST that corresponds to their original template tree. At a branching probability of 0.5, only a few plants from families $\mathcal{T}_0$ and $\mathcal{T}_1$ have a NEST corresponding to the original template tree, Fig. 17. As the probability of branching does not directly account for the amount of vertices removed in the trees, we also computed the cumulated percentage of trees whose NEST

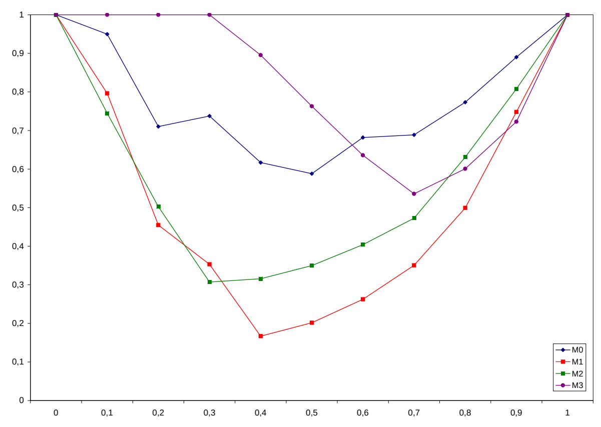

Figure 16: Average degree of self-nestedness depending on the branching probability.

preserves the template tree as a function of noise (defined as the ratio between a tree size and its template tree size) (Fig. 17.b).

Analysis of a real plant

Rice panicle

The structure of a rice panicle (*Oryza sativa* (rice) cv 'Nippon Bare') was entirely described including vegetative and floral parts [61] (Fig. 18.a). A panicle usually has complex lateral structures that are interpreted as systems reiterated from the main stem and slightly 'reduced', Fig. 18.b. The structure V_1 of the considered individual was made of a main axis bearing a main inflorescence (panicle, P_1) and four lateral reiterated systems (called tillers, $V_i,\ i = 2, .., 5$), each composed of a vegetative part (in green) and inflorescences (in red, $P_j,\ j = 2, .., 8$).

Analysis of the panicle self-nestedness

We first computed the reduction tree $\mathcal{R}(T_{rice})$ (Fig. 19). This DAG, from which the original tree can be reconstructed, is not linear and shows a number of different meristem differentiation sequences.

In Table 2, a set of global statistics of the different tillers and inflorescences is depicted with their computed self-nested properties. We first observe that the vertex compression factors $\rho'(T)$ of the structures are all above 90 percent, suggesting that such a real plant contains a high level of structural redundancy. For inflorescences $P_j,\ j = 1, .., 8$, the degree of self-nestedness is high (with a mean value of 0.91 and an average size of 97 vertices per tree), meaning that on average less than 10 percent of vertices need be added to each inflorescence to obtain a perfect self-nested tree. For bigger vegetative structures (V_1, V_2, V_3), the self-nestedness dramatically drops, reaching only 15 percent for the entire panicle (V_1).

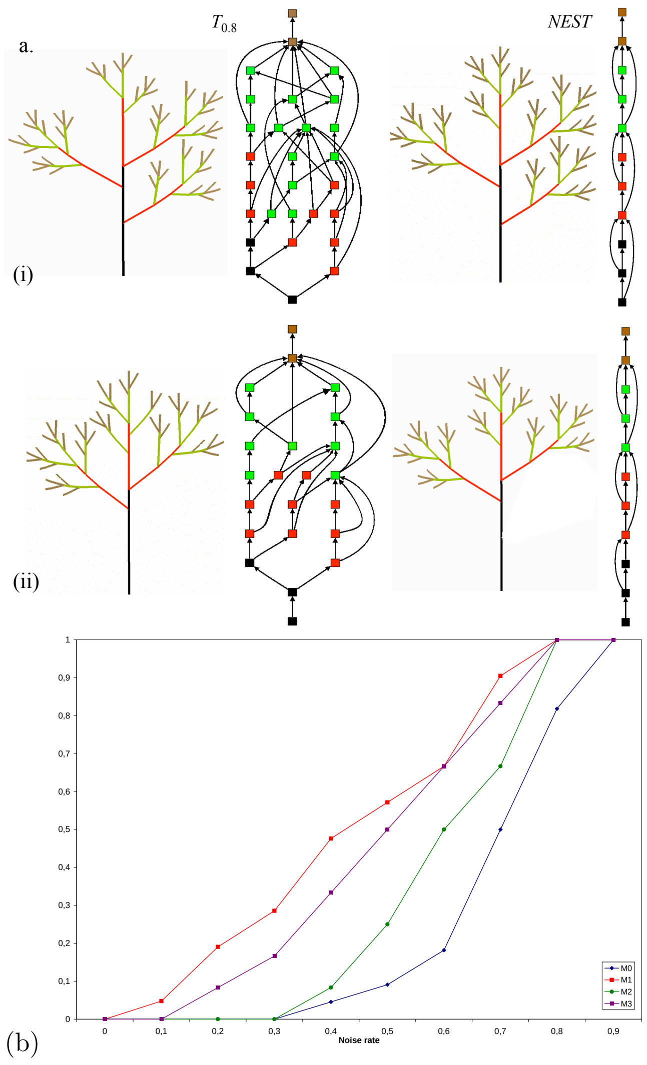

Figure 17: a. Two random versions of $T_{0,8}$ such that (i) the NEST is isomorphic to the template or (ii) not. b. Cumulated percentage of plants from the different families having a NEST isomorphic to their template plant as a function of noise.

Retrieving meristem differentiation sequences

We then computed a linear DAG T^*_{rice} in $NEST(T_{rice})$ using the NEST algorithm (Fig. 20.a). Based on the hypothesis that similar structures were produced by meristems in similar differentiation states (hypothesis of continuous developmental ability), the DAG sequence can be interpreted as the meristem differentiation sequence that best explains the original plant structure. The states of this sequence can be projected onto the original topological structure using the mapping resulting from the building of T^* from T. To interpret visually this mapping, a small number of contiguous, 4- or 5-vertices long, zones have been defined by a set of arbitrary colors (Fig. 20.a). The states of a

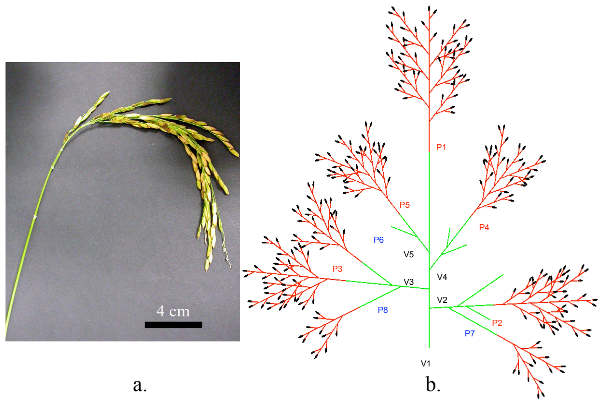

Figure 18: a) photo of a rice panicle (courtesy of Y. Caraglio). b) corresponding topological structure (reconstructed with the AMAPmod/VPlants open software [62]).

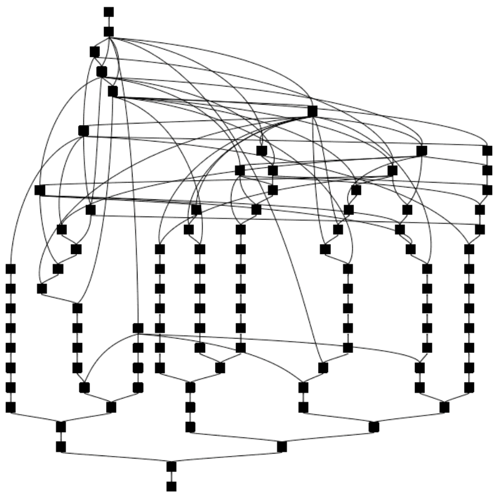

Figure 19: Reduction of the tree representing the topological structure of the rice panicle.

given differentiation zone have the same color. The color mapping on Fig. 20.b therefore characterizes parts of the panicle that were elaborated by meristems in similar differentiation states and provides a biological interpretation of the entire structure in terms of meristem differentiation.

Table 2: Rice panicle statistics.

Struct.	Size	MaxOrder	BranchNb	$\|\mathcal{R}(T)\|$	$\rho_v(T)$	$\rho'_v(T)$	$\|NEST(T)\|$	δ_{NEST}
P_1	168	3	13	19	0.89	0.96	192	0.88
P_2	128	3	13	16	0.88	0.97	164	0.78
P_3	126	3	11	14	0.88	0.98	132	0.95
P_4	95	3	10	13	0.87	0.98	107	0.89
P_5	111	3	9	14	0.87	0.97	117	0.95
P_6	69	3	7	12	0.82	0.98	75	0.92
P_7	33	3	5	9	0.78	1.00	36	1.00
P_8	42	3	6	9	0.72	0.99	42	0.92
V_1	843	5	13	106	0.87	0.90	5314	0.15
V_2	192	4	13	35	0.82	0.91	360	0.53
V_3	246	4	11	40	0.84	0.92	661	0.37
V_4	107	3	10	21	0.80	0.96	121	0.88
V_5	119	3	9	20	0.83	0.97	127	0.93

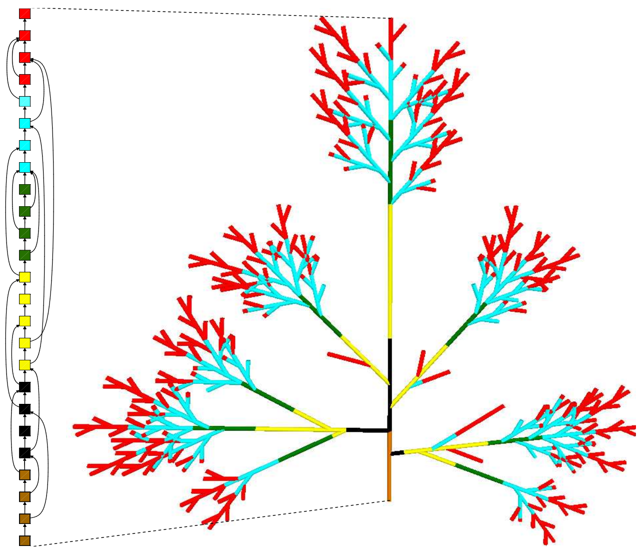

Figure 20: a. An element T^* of $NEST(T)$ b. Backward projection of the differentiation states inferred from T^* onto the initial tree structure of the panicle (see text for color interpretation).

5 Conclusion

In this paper, we introduced the notion of tree self-nestedness to assess the amount of structural redundancy embedded in a tree. Self-nested trees are such that all their subtrees of a given height are isomorphic. We derived this notion from the possibility to compress unordered trees without loss of information as more compact DAGs and showed that self-nested trees are those trees that can be compressed as linear DAGs. Two algorithms were presented to achieve this compression scheme and were shown to have identical time complexities (both algorithms pseudo-codes are detailed in the supplementary material). We then defined the degree of self-nestedness of a tree as one minus the normalized distance of the tree to its nearest embedding self-nested tree (NEST). We showed that this quantity can be computed in polynomial time and described an algorithm to compute the NEST of any tree (whose pseudo-code is described in the supplementary material).

We then illustrated these notions on the structural analysis of plant architectures. The approach was first assessed on artificial plants, for which the amount of self-nestedness was controlled by gradually introducing noise in perfect self-nested trees. For all trees, high relative compression rates were achieved by the DAG reduction, ranging from 75% to 100%. We showed that even for highly perturbed self-nested trees (*i.e.* trees with up to 50% of removed vertices compared to their template self-nested tree), the template self-nested tree could be recovered by the NEST algorithm in 10% to 50% of the cases, depending on the tree type. The degree of self-nestedness of the perturbed trees was then assessed on the tree database and was shown to reach a minimum value for intermediate perturbations. We then applied our approach to the analysis of a real plant architecture (a rice panicle). Inflorescences showed highly self-nested structures while the global plant did not. Based on the hypothesis of continuous developmental ability, we then showed how the sequence of meristem differentiation states could be derived from the computation of the NEST. This opens up the perspective to use such an analysis on various plant species as a guiding principle to further investigate the notion of meristem differentiation at a biomolecular, genetic and architectural levels, in the spirit of the pioneering work described in [58].

Acknowledgements

The authors would like to thank P. Prusinkiewicz for many and fruitful discussions about self-similarity in plants and for making the L-Studio software available to them for simulating the theoretical plants. They also thank Y. Caraglio for making available the rice data to them. Graph representations of the plant DAGs were displayed with the Tulip open-software. This work has been partially sponsored by the French ACI Arborescences, ANR NatSim and ANR Brasero (ANR-06-BLAN-0045) projects.

References

[1] F. Hallé, R. A. A. Oldeman, and P. B. Tomlinson, *Tropical trees and forests. An architectural analysis.* New-York: Springer-Verlag, 1978.

[2] J. L. Harper, B. R. Rosen, and J. White, *The growth and form of modular organisms.* London, UK: The Royal Society, 1986.

[3] D. Barthélémy, C. Edelin, and F. Halle, "Architectural concepts for tropical trees," in *Symposium on Tropical Forests* (L. B. Holm-Nielsen, I. C. Nielsen, and E. Balslev, eds.), Tropical forests: Botanical dynamics, speciation and diversity, (Aarhus, Danemark), pp. 89–100, Academic Press, London, 1989.

[4] D. Barthélémy, “Levels of organization and repetition phenomena in seed plants,” *Acta Biotheoretica*, vol. 39, pp. 309–323, 1991.

[5] P. Room, L. Maillette, and J. Hanan, “Module and metamer dynamics and virtual plants,” *Advances in Ecological Research*, vol. 25, pp. 105–157, 1994.

[6] P. Prusinkiewicz, “Modeling of spatial structure and development of plants: a review,” *Scientia Horticulturae*, vol. 74, pp. 113–149, 1998.

[7] P. Prusinkiewicz, “Selfsimilarity in plants : integrating mathematical and biological perspectives.,” in *Thinking in Patterns: Fractals and Related Phenomena in Nature* (M. M. Novak, ed.), pp. 103–118, Singapore: World Scientific, 2004.

[8] O. Deussen and B. Lintermann, *Digital Design of Nature*. Springer-Verlag, 2005.

[9] C. Godin, E. Costes, and H. Sinoquet, “Plant architecture modelling - virtual plants and complex systems,” in *Plant Architecture and its Manipulation* (C. Turnbull, ed.), vol. 17 of *Annual plant reviews*, pp. 238–287, Blackwell, 2005.

[10] M. Renton, Y. Guédon, C. Godin, and E. Costes, “Similarities and gradients in growth unit branching patterns during ontogeny in ’fuji’ apple trees: a stochastic approach.,” *J. Exp. Bot.*, vol. 57, no. 12, pp. 3131–3143, 2006.

[11] L. Mündermann, Y. Erasmus, B. Lane, E. Coen, and P. Prusinkiewicz, “Quantitative modeling of arabidopsis development.,” *Plant Physiol*, vol. 139, pp. 960–968, Oct 2005.

[12] Y. Guo, Y. Ma, Z. Zhan, B. Li, M. Dingkuhn, D. Luquet, and P. D. Reffye, “Parameter optimization and field validation of the functional-structural model greenlab for maize.,” *Ann Bot (Lond)*, vol. 97, pp. 217–230, Feb 2006.

[13] C. Godin, Y. Guédon, E. Costes, and Y. Caraglio, “Measuring and analyzing plants with the AMAPmod software,” in *Plants to ecosystems - Advances in Computational Life Sciences, 2nd International Symposium on Computer Challenges in Life Science* (M. Michalewicz, ed.), pp. 53–84, Melbourne, Australia: CSIRO Australia, 1997.

[14] H. Sinoquet and P. Rivet, “Measurement and visualisation of the architecture of an adult tree based on a three-dimensional digitising device.,” *Trees-Structure and Function*, vol. 11, pp. 265–270, 1997.

[15] Y. Guédon, D. Barthélémy, Y. Caraglio, and E. Costes, “Pattern analysis in branching and axillary flowering sequences.,” *J Theor Biol*, vol. 212, pp. 481–520, Oct 2001.

[16] Y. Guédon, P. Heuret, and E. Costes, “Comparison methods for branching and axillary flowering sequences.,” *J Theor Biol*, vol. 225, pp. 301–325, Dec 2003.

[17] P. Ferraro and C. Godin, "A distance measure between plant architectures," *Annals of Forest Science*, vol. 57, no. 5/6, pp. 445–461, 2000.

[18] J.-B. Durand, Y. Guédon, Y. Caraglio, and E. Costes, "Analysis of the plant architecture via tree-structured statistical models: the hidden markov tree models.," *New Phytol*, vol. 166, pp. 813–825, Jun 2005.

[19] P. Ferraro, C. Godin, and P. Prusinkiewicz, "Toward a quantification of self-similarity in plants," *Fractals*, vol. 13, no. 2, pp. 91–109, 2005.

[20] W. Troll, "Die wuchsform der baeume und straeucher," in *Vergleichende Morphologie der hoeheren Pflanzen* (W. Troll, ed.), vol. 1, pp. 636–643, Berlin: Borntraeger, 1937.

[21] A. Arber, *Natural philosophy of plant form.* University Press, Cambridge, 1950.

[22] D. Frijters and A. Lindenmayer, "Developmental descriptions of branching patterns with paracladial relationships," in *Formal Languages, Automata and Development* (G. Rozenberg and A. Lindenmayer, eds.), (Noordwijkerhout, The Netherlands), pp. 57–73, North-Holland Publishing Company, 1976.

[23] D. Barthélémy, Y. Caraglio, and E. Costes, "Architecture, gradients morphogénétiques et âge physiologique chez les végétaux," in *Modélisation et Simulation de l'Architecture des Végétaux* (J. Bouchon, P. d. Reffye, and D. Barthélémy, eds.), Science Update, pp. 89–136, Paris, France: INRA Editions, 1997.

[24] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants.* New York: Springer Verlag, 1990.

[25] C. Godin and Y. Caraglio, "A multiscale model of plant topological structures," *Journal of theoretical biology*, vol. 191, pp. 1–46, 1998.

[26] P. Kilpelainen and H. Mannila, "The tree inclusion problem," in *Proc. Internat. Joint Conf. on the Theory and Practice of Software*, vol. 1, pp. 202–214, 1991.

[27] L. Wang and D. Gusfield, "Improved approximation algorithms for tree alignment," *Journal of Algorithms*, vol. 25, no. 2, pp. 225–273, 1997.

[28] J. T. Wang, K. Zhang, and C. Y. Chang, "Identifying approximately common substructures in trees based on a restricted edit distance," *Information Sciences*, vol. 126, pp. 367–386, 1999.

[29] A. Ouangraoua, P. Ferraro, S. Dulucq, and L. Tichit, "Local similarity between quotiented ordered trees," *Journal of Discrete Algorithms*, vol. 5, no. 1, pp. 23–35, 2007.

[30] K. Zhang, "A new editing based distance between unordered labeled trees," in *Combinatorial Pattern Matching, 4th Annual Symposium CPM 93*, (Padova (IT)), 1993.

[31] K. Zhang, “A constrained edit distance between unordered labeled trees,” *Algorithmica*, vol. 15, no. 3, pp. 205–222, 1996.

[32] S. Akers, “Binary decision diagrams,” *IEEE Transactions on computers*, vol. C-27, no. 6, pp. 509–516, 1978.

[33] R. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE transactions on computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[34] A. Aho, I. Hopcroft, and J. Ullman, *The design and analysis of computer algorithms.* Reading, MA: Addison-Wesley, 1974.

[35] J. Hart and T. DeFanti, “Efficient anti-aliased rendering of 3d linear fractals,” in *Computer Graphics* (T. W. Sederberg, ed.), vol. 25, pp. 91–100, 1991.

[36] I. Sutherland, “Sketchpad - a man-machine graphical communication system,” in *Proceedings of the Spring Joint Computer Conference*, 1963.

[37] J. Hart, “The object instancing paradigm for linear fractal modeling,” in *Graphics Interface*, pp. 224–231, 1992.

[38] T. Kay and J. Kajiya, “Ray tracing complex scenes,” in *Proceedings of SIGGRAPH’86*, vol. 20, (Dallas), pp. 269–278, August 1986.

[39] C. Soler, F. X. Sillion, F. Blaise, and P. Reffye, “An efficient instantiation algorithm for simulating radiant energy transfer in plant models,” *ACM Transactions on Graphics*, vol. 22, no. 2, pp. 204–233, 2003.

[40] E. Tokunaga, “Consideration on the composition of drainage networks and their evolution,” *Geogr. Rep. Tokyo Metro. Univ.*, vol. 13, pp. 1–27, 1978.

[41] X. Viennot, G. Eyrolles, N. Janey, and D. Arqués, “Combinatorial analysis of ramified patterns and computer imagery of trees,” in *SIGGRAPH ’89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 31–40, ACM, 1989.

[42] S. Peckam, “New results for self-similar trees with applications to river networks,” *Water Resource Res.*, vol. 31, pp. 1023–1029, 1995.

[43] M. Zamir, “On fractal properties of arterial trees,” *Journal of Theoretical Biology*, vol. 1997, no. 4, pp. 517–526, 2002.

[44] B. Kron, “Growth of self-similar graphs,” *Journal of Graph Theory*, vol. 45, no. 3, pp. 224–239, 2004.

[45] R. Greenlaw, “Subtree isomorphism is in DLOG for nested trees,” *International Journal of Foundations of Computer Science*, vol. 7, no. 2, pp. 161–168, 1996.

[46] F. Preparata and R. Yeh, *Introduction to discrete structures for computer science and engineering.* Reading Menlo Park London: Addison-Wesley, 1973.

[47] P. Downey, R. Sethi, and R. Tarjan, “Variations on the common subexpression problem,” *Journal of the ACM Transactions on Graphics*, vol. 27, no. 4, pp. 758–771, 1980.

[48] D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1997.

[49] G. Valiente, “An efficient bottom-up distance between trees,” in *Proceedings.Eighth International Symposium on String Processing and Information Retrieval. SPIRE 2001.*, 2001.

[50] R. Y. Pinter, O. Rokhlenkoa, D. Tsurb, , and M. Ziv-Ukelson, “Approximate labelled subtree homeomorphism,” *Journal of Discrete Algorithms*, vol. 6, pp. 480–496, september 2008.

[51] D. P. Bertsekas, *Dynamic programming : deterministic and stochastic models*. Upper Saddle River, NJ, USA: Prentice-Hall, 1987.

[52] R. Nozeran, “Integration of organismal development,” in *Positional controls in plant development* (P. Barlow and D. Carr, eds.), pp. 375–401, 1984.

[53] D. Barthélémy and Y. Caraglio, “Plant architecture: a dynamic, multilevel and comprehensive approach to plant form, structure and ontogeny.,” *Ann Bot (Lond)*, vol. 99, pp. 375–407, Mar 2007.

[54] J. W. v. Goethe, *Versuch, die Metamorphose der Pflanzen zu erklaeren*. Gotha: Carl Wilhelm Ettinger, 1790.

[55] J. W. von Goethe, *Goethe’s Botanical Writings*. Ox Bow Press, Oct. 1989.

[56] L. E. Gatsuk, O. V. Smirnova, L. I. Vorontzova, L. B. Zaugolnova, and L. A. Zhukova., “Age states of plants of various growth forms : a review,” *J. Ecol.*, vol. 68, pp. 675–696, 1980.

[57] L. Maillette, “The value of meristem states, as estimated by a discrete-time markov chain,” *OIKOS*, vol. 59, pp. 235–240, 1990.

[58] P. Prusinkiewicz, Y. Erasmus, B. Lane, L. D. Harder, and E. Coen, “Evolution and development of inflorescence architectures.,” *Science*, vol. 316, pp. 1452–1456, Jun 2007.

[59] R. Mech, M. James, M. Hammel, J. Hanan, and P. Prusinkiewicz, *CPFG v4.0 user manual*. The University of Calgary, 2005.

[60] P. Prusinkiewicz, “Art and science for life: Designing and growing virtual plants with L-systems.,” *Acta Horticulturae*, vol. 630, pp. 15–28, 2004.

[61] C. Paul-Victor, “Diversité morphologique et architecturale de quatre variétés de riz (*Oryza sativa* et *Oryza glaberrina*),” Master’s thesis, University of Jussieu, Paris 6, 2004.

[62] C. Godin, E. Costes, and H. Sinoquet, “A method for describing plant architecture which integrates topology and geometry,” *Annals of Botany*, vol. 84, pp. 343–357, 1999.

[63] A. Gupta and N. Nishimura, "Finding largest subtrees and smallest supertrees," *Algorithmica*, vol. 21, no. 2, pp. 183–210, 1998.

[64] R. E. Burkard, "Selected topics on assignment problems," *Discrete Applied Mathematics*, vol. 123, pp. 257–302, 2002.

A Proof of theorem 1

A.1 Proof of proposition 11

Lemma 1 *There exists a growing sequence* $\{Z_K\}_{K\in\{1\dots h(G)\}}$ *solution of* $\underset{H\in\mathcal{S}_K^+(G)}{\text{argmin}}\,\{n(H)\}$, i.e. $\forall K$ *in* $\{2\dots h(G)\}$,

$$Z_{K-1} \subseteq Z_K$$

Proof: By definition for any $K < h(G)$ and for any $Z_K \in \mathcal{S}_K^+(G)$ there exists $L_K \in \mathcal{L}$ such that $h(L_K) = K$ such that $Z_K = G\langle L_K\rangle$. Consequently:

$$\forall v \in G,\ h(v) \leq K \Leftrightarrow G(v) \subseteq L_K$$

This means that $\mathcal{T}(L_K)$ is a super-tree of the subtrees of height K in $\mathcal{T}(G)$.

Furthermore, $D(Z_K, G) = n(Z_K) - n(G) = \min\{D(H,G), H \in \mathcal{S}_K^+(G)\}$. $n(Z_K)$ and $n(G)$ represent respectively the number of nodes of $\mathcal{T}(Z_K)$ and $\mathcal{T}(G)$. Actually the only differences between the trees $\mathcal{T}(Z_K)$ and $\mathcal{T}(G)$ are the subtrees of height K in $\mathcal{T}(G)$ that have been replace by $\mathcal{T}(L_K)$. In other terms, $\mathcal{T}(L_K)$ is a **smallest** super-tree of the the subtrees of height K in $\mathcal{T}(G)$.

However, a smallest super-tree $\mathcal{T}(L_K)$ of a set of trees can be recursively computed from the smallest super-tree $\mathcal{T}(L_{K-1})$ of the sub-trees of height $K-1$ ([63] lemma 5.1) and is such that :

$$\begin{aligned} & \mathcal{T}(L_{K-1}) \subseteq \mathcal{T}(L_K) \\ \Leftrightarrow\ & L_{K-1} \subseteq L_K \\ \Leftrightarrow\ & G\langle L_{K-1}\rangle \subseteq G\langle L_K\rangle \\ \Leftrightarrow\ & Z_{K-1} \subseteq Z_K \end{aligned}$$

■

Lemma 2

$$\mathcal{S}_K^+(G) = \mathcal{S}_K^+(Z_{K-1})$$

Proof: According to lemma 1

$$\begin{aligned} \forall H \in \mathcal{S}_K^+(G) \qquad & \exists Z_{K-1}, Z_K:\ G \subseteq Z_{K-1} \subseteq Z_K \subseteq H \\ \Rightarrow\ & H \in \mathcal{S}_K^+(Z_{K-1}) \end{aligned}$$

Reciprocally, for any H in $\mathcal{S}_K^+(Z_{K-1})$ $\exists L \in \mathcal{L}$; $h(L) = K$ and $Z_{K-1}\langle L\rangle = G\langle L_{K-1}\rangle\langle L\rangle \subseteq H$ then $G \subseteq H$, and then $H \in \mathcal{S}_K^+(G)$. ■

Lemma 3 *Let G be a DAG, let K be an integer and let $Z_K \in \operatorname{argmin}\{n(H);\ H \in \mathcal{S}_K^+(G)\}$, then $\exists Z_{K-1} \in \operatorname{argmin}\{n(H);\ H \in \mathcal{S}_{K-1}^+(G)\}$ such that :*

$$D(G, Z_K) = D(G, Z_{K-1}) + D(Z_{K-1}, Z_K)$$

Proof: Following lemma 1, $\exists Z_{K-1} \subseteq Z_K$. Then (prop. 5):

$$D(G, Z_K) = D(G, Z_{K-1}) + D(Z_{K-1}, Z_K)$$

■

Then, according to lemma 3:

$$\begin{aligned} & min_{H \in \mathcal{S}_K^+(G)}\{D(G,H)\} \\ & = D(G, Z_{K-1}) + D(Z_{K-1}, Z_K)\} \\ & = D(G, Z_{K-1}) + min_{H \in \mathcal{S}_K^+(G)}\{D(Z_{K-1}, H)\} \\ & = D(G, Z_{K-1}) + min_{H \in \mathcal{S}_K^+(Z_{K-1})}\{D(Z_{K-1}, H)\} \end{aligned}$$

which completes the proof of proposition 11.

A.2 Proof of proposition 12

In the following, we assume that for any $h \in \{0 \dots K-1\}$, Z_h has been recursively computed fro G. Let us consider the computation of Z_K. Z_{K-1} can be represented as in Fig. 6. In Z_{K-1}, the nodes $(z_h)_{h \in 1..K-1}$ represent the nodes in the *linear part* of Z_{K-1} and the nodes $w_1, w_2, \dots, w_I$ represent the nodes of Z_{K-1} such that $h(w_i) = K$ for any $i \in \{1..I\}$. In the following, the subtrees $\mathcal{T}(Z_{K-1}[w_i])$ of $\mathcal{T}(Z_{K-1})$ that are defined by a node w_i of the DAG Z_{K-1} will be simply denoted by $\mathcal{T}(w_i)$

As proposed above, Z_K is the smallest self-nested DAG in $\mathcal{S}_K^+(Z_{K-1})$. This means that any subtrees of $\mathcal{T}(Z_{K-1})$ of height K (basically the trees $(\mathcal{T}(w_i))_{i \leq I}$) must be included in a subtree $\mathcal{T}(z_k)$ of $\mathcal{T}(Z_K)$. In other terms, $\mathcal{T}(Z_K(z_k))$ is the smallest super tree of the sequence of trees $(\mathcal{T}(w_i)_{i \leq I}$.

Gupta and Nishimura proposed in [63] a method that computes the smallest super-tree of two trees in $O(n^{2.5} \log n)$. However, with the configuration of our problem, the super-tree can be computed in linear time. We present hereafter the proof for two trees. The result can be generalized to the sequence $(\mathcal{T}(w_i))_{i \leq I}$.

Let us consider two trees $T_i = \mathcal{T}(w_i)$ and $T_j = \mathcal{T}(w_j)$ of this sequence. Determining the smallest super-tree of T_i and T_j involves solving a bipartite matching problem [63]. In the bipartite graph $G(X,Y)$ (fig. 22), X represents all the subtrees (rooted in a child of the root of T_i) of T_i and Y represents all the subtrees of T_j. Remark that X and Y are ordered according to the size of the subtrees. This order also corresponds to the nestedness of the trees. The cost c_{kl} of matching a tree $\mathcal{T}(z_k)$ from X to a tree $\mathcal{T}(z_l)$ is obviously the absolute difference of number of nodes between $\mathcal{T}(z_k)$ and $\mathcal{T}(z_l)$ (*i.e.* $|n(z_k) - n(z_l)|$). In general, X and Y do not have necessarily the same size (we suppose $|X| \geq |Y|$). To capture this possibility, it is usual to add *empty trees* to the smallest set Y. The matching cost to these *empty trees* to a tree $\mathcal{T}(z_k)$ is then the size of the tree itself. Finally, the bipartite matching problem is equivalent to find a permutation π such that $\sum_{0 \leq i < |X|} c_{i,\pi(i)}$ is minimized. It can be shown [64]

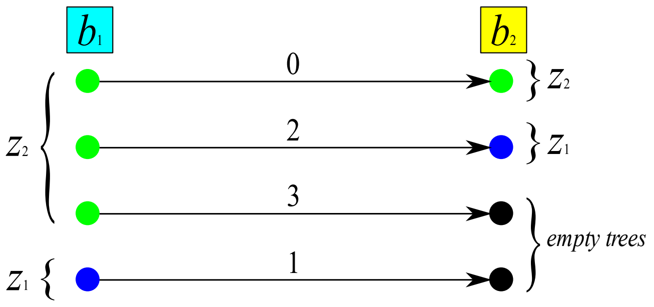

Figure 21: Bipartite Matching problem associated with Fig. 7c.

that the identical permutation is an optimal solution if the cost matrix fullfills the *weak Monge property*:

$$\begin{aligned} &for\ 1 \le i < k \le |X|,\ and\ 1 \le i < l \le |X| : \\ &c_{ii} + c_{kl} \le c_{il} + c_{ki}, \end{aligned}$$

It can be easily shown that since X and Y are ordered according to the size of the trees, the matrix cost of our problem has the *weak Monge property* (using the triangular inequality of the absolute value). Subtrees of T_i must then be assign to the subtrees of T_j according to their sizes. Finally, from the initial DAG (Fig. 6, we get the matching shown in Fig. 22.

In order to illustrate the above proof, Fig. 21 shows the bipartite matching problem when computing the node z_3 in the DAG of Fig. 7c. The cost of this matching problem is then the sum of the edge cost (6 in this case) and represent the number of nodes that should be added to $\mathcal{T}(b_3)$ in order to obtained the smallest super-tree of $\mathcal{T}(b_3)$ and $\mathcal{T}(b_1)$. The cost of the matching problem is computed in $O(max(\deg(T_i), \deg(T_j))) \subset O(\deg(Z_{K-1}))$. The matching problem can be solved simultaneously for all the subtrees of $\mathcal{Z}_{K-1}$ in the same time complexity.

B Tree Reduction Algorithm

```
input : tree = <V,E>
output : dag = <node_list,edge_list>
function treeReduction(tree):
 edge_list = []
 node_list = []
 signature_list = []
 signature = 0
 for i in 1..|V|:
  n = V[i]         # ith node of Tree
  signatures_of_children = []
  for child in childList(n) :
   signatures_of_children
      += [child.signature]
   if signatures_of_children is
```

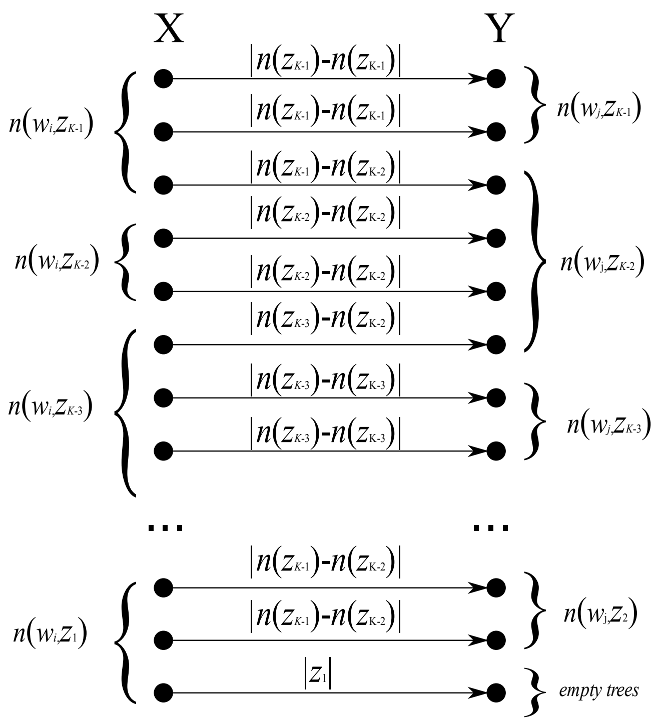

Figure 22: Bipartite Matching Problem in the general case

```
              in signature_list :
   signature += 1
   n_i = new Node
   n_i.signature = signature
   node_list += [n_i]
   i = 0
   for k in signatures_of_children:
    e = new Edge
    e.begin = node_list[signature]
    e.end = node_list[k]
    edge_list += [e]
    signature_list
        += [signatures_of_children]
   n.signature = signature
 return <node_list,edge_list>
```

C Tree Reconstruction Algorithm

```
input :  dag = <V,E>
output : tree = <node_list,edge_list>
function treeReconstruction(tree):
 edge_list = []
 node_list = []
 signature = nbNodes(dag)
 signature = 0
 for i in 1..|V|: # V[1] is the root of dag
  n_i = V[i]
  for j in 1..nbEdges(n_i):
   # nbEdges is the number of edges
   # connected to n_i
   e_j = E[n_i,j]
   # e is the jth edge connected to n_i
   for k in 1..label(e_j):
    tree_node = new Node
    tree_node.signature = e.end.signature
    tree_edge = new Edge(tree_node,n_i)
    node_list += [tree_node]
    edge_list += [tree_edge]
 return <node_list,edge_list>
```

D NEST Algorithm

```
input : tree = <V,E>
output: dag = <node_list,edge_list>
function NEST(tree):
 ZK = TREE_REDUCTION(tree)
 for K in 1 ... height(ZK):
  for l in K-1 ... 1:
   nl = node of ZK at height l
   max value = the maximum signature
               between nl and any
               node of ZK at height K
   for zk in Node(ZK):
    if height(zk)==K:
     # signature between nl and
     # zk is change in max_value
     signature_list
         = list of signature between
           zk and ni fo any ni in ZK
           at height less than K
     signature_sum = sum(signature_list)
     r = max_value - signature_list[K-1]
     if signature_sum > max_value :
      i = K-1
      while r != 0:
       signature_list[i] = 0
       i += 1
       r -= min(r,signature_list[i])
      signature_list[K-1] = max_value
     else :
      signature_list[K-1] = max_value
      for i in K-2 ... 0:
       signature_list[i]=0
   create a node N in ZK at height K
   such that the signature between N
   and any node ni in ZK at height
   than k is the maximum signature
   between a zk at height K and ni
   remove any node zk of ZK at height
   K except N (the ancestors of zk
   are connected with N)
 return ZK
```

Contents

1 Introduction **3**

2 Tree reduction **5**
- 2.1 Definitions and notations 5
- 2.2 Computing tree reduction 8
  - 2.2.1 Signature-based algorithm 8
  - 2.2.2 Edit-distance-based algorithms 9
- 2.3 Properties of tree reductions 10

3 Self-nested trees **12**
- 3.1 Definition 12
- 3.2 Reduction of self-nested trees 13
- 3.3 Determination of the nearest self-nested tree of a tree 13

4 Application to the structural analysis of plants **19**

5 Conclusion **31**

A Proof of theorem 1 **37**
- A.1 Proof of proposition 11 37
- A.2 Proof of proposition 12 38

B Tree Reduction Algorithm **39**

C Tree Reconstruction Algorithm **40**

D NEST Algorithm **41**

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399