N° d'ordre :

REPUBLIQUE ALGERIENNE DEMOCRATIQUE & POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR & DE LA RECHERCHE SCIENTIFIQUE



UNIVERSITE DJILLALI LIABES
FACULTE DES SCIENCES EXACTES
SIDI BEL ABBÈS

THESE DE DOCTORAT

Présentée par

HABIBECHE Salah Eddine Benamrane

Domaine : Mathématiques et Informatique
Filière : Informatique
Intitulé de la formation : Technologie de l'information

Intitulée

Optimisation des algorithmes de calcul de la distance d'édition entre les arbres ordonnés et semi-ordonnés à partir de leur compression en graphes dirigés acycliques.

Soutenue le……
Devant le jury composé de :

Président : Pr ELBERRICHI Zakaria, UDL de Sidi Bel Abbes
Examinateurs : Pr. GODIN Christophe, Directeur de recherche INRIA Lyon
Pr. SAKHRI Larbi, Univ. Oran 1
Pr. BELALEM Ghalem; Univ. Oran 1
Dr. KHOBZAOUI Abdelkader, UDL de Sidi Bel Abbes

Directeur de thèse : Dr Farah Ben-Naoum

Année universitaire : 2022/2023

To my Family

Acknowledgements

First and foremost I am extremely grateful to my supervisor, Dr Farah Ben-Naoum for her invaluable advice, continuous support, and patience during my PhD study. her immense knowledge and plentiful experience have encouraged me in all the time of my academic research. I would like to extend my sincere appreciation to the member of the defense jury for their willingness to review and evaluate my work, showcasing their genuine interest in it. I would also like to thank Christophe Godin, Research Director at Inria and Romain Azaïs Research Manager at Inria, who have always guided, advised and supported me without reservation, which has greatly contributed to the realization of my work. Furthermore, i would like to thank the various institutions that welcomed me to their premises: Djilali Liabes University, EEDIS laboratory, RDP laboratory, to all their teams, their researchers who contributed to this work.

Abstract

Trees form a large family of combinatorial objects offering a wide range of application domains. The broadening of these application domains has naturally led to the division of this family into three large classes of trees: ordered, unordered and semi-ordered trees. This diversity has also motivated the development of a large panorama of algorithms that aim at solving the different problems related to this structure. Today, and in front of an explosion of the size of the trees, the new stake is to study how to optimize these algorithms to gain orders of orders of magnitude in complexity. One way to achieve this goal is to approach the original trees by compressed structures offering better algorithmic properties, and it is in this context that our current work is situated. We consider the problem of optimising the tree edit distance calculation algorithms by compressing them into DAGs. We are then interested firestly in defining the compression of ordered and semi-ordered trees inspired by the work done on unordered trees, and secondly we are interested in the lossy compression of unordered labeled trees.

Keywords: Graph Theory, Tree,DAG ,Data compression, Plan architecture.

Field: Computer Science..

Résumé

Les arbres forment une grande famille d'objets combinatoires offrant un large éventail de domaines d'application. L'élargissement de ces domaines d'application a naturellement conduit à la division de cette famille en trois grandes classes d'arbres : les arbres ordonnés, non ordonnés et semi-ordonnés. Cette diversité a également motivé le développement d'un large panorama d'algorithmes visant à résoudre les différents problèmes liés à cette structure. Aujourd'hui, et face à une explosion de la taille des arbres, le nouvel enjeu est d'étudier comment optimiser ces algorithmes pour gagner des ordres de grandeur en complexité. Une façon d'atteindre cet objectif est d'approcher les arbres originaux par des structures compressées offrant de meilleures propriétés algorithmiques, et c'est dans ce contexte que se situe notre travail actuel. Nous considérons le problème de l'optimisation des algorithmes de calcul de la distance d'édition des arbres en les compressant en DAGs. Nous nous intéressons alors tout d'abord à la définition de la compression des arbres ordonnés et semi-ordonnés en nous inspirant des travaux réalisés sur les arbres non ordonnés, puis nous nous intéressons à la compression avec perte des arbres étiquetés non ordonnés.

Mots-clefs : Theorie des graphes, Arborescence, DAG ,Compression de données, Architecture des plantes.

Discipline : Informatique.

ملخص

تشكل الأشجار مجموعة كبيرة من العناصر الاندماجية التي تقدم نطاقًا واسعًا من مجالات التطبيق. أدى توسع مجالات التطبيق هذا بشكل طبيعي إلى تقسيم هذه العائلة إلى ثلاث فئات كبيرة من الأشجار: الأشجار المرتبة وغير المرتبة وشبه المرتبة. وقد حفز هذا التنوع أيضًا على تطوير بانوراما كبيرة من الخوارزميات التي تهدف إلى حل المشكلات المختلفة المتعلقة بهذا الهيكل. اليوم ، وأمام انفجار احجام الأشجار ، تتمثل المهمة الجديدة في دراسة كيفية تحسين هذه الخوارزميات للحصول على خوارزميات اقل تعقيدا . تتمثل إحدى طرق تحقيق هذا الهدف في تقريب الأشجار الأصلية بهياكل مضغوطة تقدم خصائص خوارزمية أفضل ، وفي هذا السياق يقع عملنا الحالي. نحن نعتبر مشكلة تحسين خوارزميات حساب مسافة التعديل بين الهياكل الشجرية عن طريق ضغطها في بيانات موجهة لا دورية. نتطرق في البداية الى تعريف ضغط الأشجار المرتبة وشبه المرتبة المستوحى من العمل المنجز على الأشجار غير المرتبة ، قبل ان نهتم بالضغط بالإضاعة للأشجار غير المرتبة ذات العقد المسماة.

الكلمات الدلالية: هندسة البيانيات، بيانات موجهة لا دورية ، هيكل الشجرة ، ضغط البيانات ، هندسة النبات.

التخصص: اعلام ألي

Table of Contents

ACKNOWLEDGEMENTS **1**

Introduction **10**

1 Definitions and notations **12**
1.1 Graph Theory Notations and Definitions . . . 13
1.2 Definitions related to trees . . . 17
1.3 Definitions related to algorithms . . . 20
1.4 State of the art . . . 21
1.4.1 Edit distance . . . 22
1.4.2 Ordered trees edit distance . . . 25
1.4.3 Unordered trees edit distance . . . 26
1.4.4 Semi-ordered trees edit distance . . . 28
1.4.5 Lossless DAG reduction of labeled trees . . . 28
1.4.6 Lossless DAG reduction of a forest . . . 29
1.5 Conclusion . . . 29

2 Lossless Compression of Ordered Tree **31**
2.1 Introduction . . . 31
2.2 Identification of redundancies . . . 31
2.3 Computing quotient graph . . . 32
2.4 Computing tree reduction . . . 33
2.5 Properties of tree reduction . . . 34
2.6 Validation . . . 35
2.7 Conclusion . . . 37

3 Lossless Compression of Semi-Ordered Trees **38**
3.1 Introduction . . . 38
3.2 Separation of children . . . 38
3.3 Signature of a vertex . . . 40
3.4 Tree reduction and reconstruction algorithms . . . 41
3.4.1 Properties of tree reduction . . . 42
3.5 Experiments . . . 43
3.5.1 Creation of the plant database: . . . 44
3.5.2 Analysis and results: . . . 46

3.6 Conclusion . . . 48

4 Lossy compression of labeled trees: cut-based approximation approach 49
4.1 Introduction . . . 49
4.2 Problem formalization . . . 49
4.3 Global clustering strategy . . . 50
4.4 Tree cut strategy . . . 51
4.4.1 Local Clustering . . . 52
4.4.2 Cut approximation . . . 52
4.4.3 Tree approximation from an optimal cut . . . 54
4.4.4 Constraints on cuts . . . 54
4.4.5 Optimal cut selection . . . 54
4.5 Application on virtual plants . . . 55
4.5.1 Plant architecture database . . . 55
4.5.2 Constraint on plant cuts . . . 55
4.5.3 Analysis and performance . . . 57
4.6 Towards an exploration of symmetries in plant architectures . . . 61
4.6.1 From cuts to antichains and Maximal symmetry antichain . . . 62
4.6.2 The graph of maximal symmetry antichains . . . 62
4.7 Conclusion . . . 63

Conclusion and perspectives 65

A Ordered Tree Compression pseudo algorithms 66

B Lossless Compression of Semi-Ordered Trees pseudo algorithms 69

C Lossy compression of labeled trees pseudo algorithms 72

D Stochastic plant models 77
D.1 Model M_1 . . . 77
D.2 Model M_2 . . . 77
D.3 Model M_3 . . . 77

Bibliography 78

Illustrations

1.1 The seven bridges problem of Konigsberg with a diagram of the bridges taken from [12](a) and a modeling of the problem in the form of a graph(b). 13
1.2 A graph $G = (V, E)$ where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$. v_1 and v_2 are the endpoints of edge e_1. v_2, v_3 and v_4 form the neighborhood of v_5. v_6 is an isolated vertex. e_7 is a self-loop 14
1.3 A digraph $G = (V, E)$. $deg^-(v_2) = 1$ and $deg^+(v_2) = 2$. $deg(v_2) = 3$. $|G| = 5$, $deg(G) = 3$ 15
1.4 The two graphs are isomorphic if we only consider the structures but they are not if we consider the labels. 17
1.5 The graph on the left is a tree; the other two graphs are not. 18
1.6 a tree T (a) and its subtree $T[b]$rooted at vertex b (b), $depth(d) = 2, height(d) = 1\, root(T) = a, height(T) = 3, leaves(T) = \{f, g, h, i, j\}$, $child(d) = \{f, g\}$, $parent(d) = b$, $Ancestors(d) = \{b, a\}$, $Descendants(b) = \{d, f, g, e, h\}$, d and e are sibling. 19
1.7 (a) T_1 tree. (b) T_2 tree. if we view the trees as unordered trees T_1 and T_2 are isomorphic. However, in the case of ordered trees, they are not isomorphic. 20
1.8 Compression and reconstruction. 22
1.9 Edit operations on trees 23
1.10 Transforming (a) into (c) via edit operations. (a) A tree. (b) The tree after inserting the node labeled k. (c) The tree after deleting the node labeled d and relabeling c to l 23
1.11 The mapping corresponding to the edit script in Fig 1.10 24
1.12 Invalid mapping in case of constrained edit distance. 27
1.13 At left an unordered labeled tree T, in right its reduction $\mathcal{R}(T)$. Roots of isomorphic subtrees in T and the corresponding equivalence classes nodes in $\mathcal{R}(T)$ are colored identically. 29
1.14 A step by step illustration of the construction of the reduction of a forest (a) from the trees of this forest (b) and from their DAG reductions. 30

2.1 An ordered tree. $\sigma(f) = \sigma(g) = \sigma(h) = \sigma(i) = \sigma(j) = \{\theta\}$ simplified 0, $\sigma(d) = \sigma(c) = \{0, 0\}$ simplified 1, $\sigma(e) = \{0\}$ simplified 2, $\sigma(b) = \{1, 2\}$ simplified 3, $\sigma(a) = \{3, 1\}$ simplified 4. 32

2.2 (a) An ordered tree where vertices having the same signatures are colored by the same colors. (b) the resulting quotient graph with the class colors shown in the same color as their nodes. 33
2.3 (a),(b) Fig.2.2, (c) the quotient dag with augmented edges. 34
2.4 Evaluation of vertices and edges compression factors according to the size of the trees. 36
2.5 Average time compression. 37
2.6 Comparison of compression rates between ordered and unordered trees. 37

3.1 a) a semi-ordered tree in which equivalent (unordered) sibling are colored identically. (b) the ordered partition $\Gamma(A)$ induced by its total semi-order relations $\sqsubseteq_A$. 39
3.2 The quotient graph corresponding to tree T of Fig.3.1 41
3.3 The reduction corresponding to tree T of Fig.3.1 42
3.4 (a) Visualization of plant architecture using a schematic plant representation. (b)Visualization using the equivalent unordered tree T. . . 44
3.5 (a) Visualization of plant architecture using a schematic plant representation. (b)Visualization using the equivalent unordered tree T. . . 45
3.6 a) Differentiation graph used for the definition of a theoretical plant b) the resulting apical axes where each component has the same color of the differentiation graph state in which this component has been produced. (c)The whole resulted branching structure 45
3.7 The differentiation graphs of models M_1, M_2 46
3.8 Theoretical plants: the F_1 family (first row) and the F_2 family (second row). Both of T_1 and T_2 present repetitions of the lateral branches hence a high degree of compressibility. However, for the others trees the random removal in lateral structures introduces variation that is expected to reduce their degree of compressibility. 47
3.9 Average vertices and edges compression factors of tree families generated from the models M_1, M_2 according to the probability of lateral axis growing 47

4.1 An unordered tree T (left). A global clustering of T subtrees generating 5 clusters (right) where we use the same color for roots of subtrees belonging to the same cluster. 51
4.2 A tree in which we consider that subtrees (A, D) belong to the same cluster while (B, C) belong to an other one. The approximation of C by B does not remove the occurrence of C in the approximated tree since the latter is a subtree of B, it will also create a new class of subtrees and as a result the DAG after approximation is greater than the initial DAG. Similarly, if we approximate B by C this does not guarantee that the result on A is always similar to D. 51
4.3 (a) A horizontal cut crosses the tree T and separate it into: (b) a rooted tree $\mathcal{T}$ at $root(T)$ with non-terminal leaves (anchors) colored in black and (c) a disjoint forest κ. 52

4.4 (a) Local clustering of the forest κ of Fig.4.3.c, roots of clusters representatives are colored in red. (b) Approximation of subtrees belonging to the same cluster by the cluster representative in the initial tree of Fig.4.3.a. 53
4.5 (a) An unordered tree T (left), its DAG reduction (right), (b) An approximation T^* of T (left), the DAG reduction of T^* and the lossy compression of T (right). 53
4.6 Theoretical plants generate with models M_1, M_2 and M_3 in that order. 56
4.7 An artificial plant with a compression rate of 0.1% (left), and its approximation with a compression rate of 94.2% (right). 56
4.8 In red the elementary part of the plant to be preserved during the approximation and in green the dynamic part on which the cuts are generated. 57
4.9 From left to right a plant generated with the M_1 model followed by the elementary parts corresponding to a threshold of 0.2, 0.3, and 0.4 on λ_e. 57
4.10 The lines corresponding to experiments with thresholds of 0.2, 0.3 and 0.4 on λ_e, in each line from left to right the initial tree followed by the approximations corresponding to values of τ_{min} among [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], the absence of the plant for a specific value of τ_{min} means that no approximation with a compression rate greater than or equal to τ_{min} was found. 58
4.11 On the left the initial plant followed by: on the first line the result of the first cut with a threshold of 0.2 on λ_e then that of the second after threshold relaxation to 0.3, on the second line the result of the first cut with a threshold of 0.2 then that of the second after threshold relaxation to 0.4. 59
4.12 The loss of information due to the approximation against τ_{min}. 60
4.13 Evolution of the gain in the edit distance computation time (on the y-axis) against the minimum compression rate (on the x-axis) 61
4.14 Evolution of the error rate (on the y-axis) against the minimum compression rate (on the x-axis) 62
4.15 A plant architecture where we show the subtrees of an antichain colored with the same color 63
4.16 A plant architecture with 6 maximal symmetry antichains (identically colored subparts of the plant) which nesting links are given by the graph on the right. 64

Tables

2.1 Evaluation of vertices and edges compression ratios according to the size of the trees. 36
2.2 Average time compression. 36

3.1 Average tree statistics of tree families generated from the models M_1, M_2. Representation of trees sizes variances and their compression ratio according to the probability of lateral axis growing 48

Introduction

Interest in graph theory and its applications due to the usefulness of graphs as models for computation and optimization grows rapidly, largely and in several areas such as telecommunication networks, large-scale integrated circuit design, social networks and biological networks and other different fields. Therefore, this discipline has become central in engineering and computer science curriculum and several works have been carried out on different families of graphs.

The family of acyclic directed graphs, also called trees, quickly found its place and was used in various applications in biology, genealogical studies, computer vision and pattern recognition, programming compilation and natural language process.

An important tree-related problem is comparison, which concerns the definition and calculation of similarities between the architectures and information of two trees. The comparison of trees can be a problem in itself but can also be part of another problem like indexing, pattern recognition, symmetry detection...The edit distance addresses the problem of comparing trees by calculating the minimal cost of transforming an initial tree to a target tree with basic edit operations (add, delete or relabel nodes).

The diversity of the application fields but also the very large tree sizes in some areas make the optimisation of edit distance calculation interesting. Work is needed for the optimisation not only on the algorithms but also on the data. Tree compression can be defined as the representation of trees in a compact form, its principle is to identify similar parts of the tree and to represent them only once. The calculation of the edit distance will be done afterwards on the compact forms instead of on the trees.

We consider in this manuscript the problem of optimising algorithms for computing the edit distance between trees by compressing them into acyclic directed graphs. Initial work aimed at lossless compression of unordered trees into DAGs [16]. This thesis first focused on a generalization of the proposed methods for ordered and semi-ordered trees and then on a lossy compression of unordered trees but also to the consideration of additional information attached to the nodes.

Plants are branching living organisms that develop throughout their lifetimes. The architectures of plants are either described by ordered or unordered rooted trees([14] [30]). In order to show the usefulness of the proposals we also show in this manuscript the application of our proposal for the compression of semi-ordered trees on the compression of plant architectures, as well as the application of the lossy compression of unordered labeled trees on plant architecture.

The implementation of all proposed algorithms used the models developed in the TreeX library [4] and today our algorithms are also part of this library. TreeX is an open source Python package distributed under the LGPL License. TreeX manipulates rooted trees ordered or not, labeled or not, offers several features (generation of trees, edit operations...) and implements several algorithms(DAG compression[16], edit distance calculation between trees [3], Self-nested approximations of trees [16]...).

The thesis is structured in four chapters:

In Chapter 1, we present in the first part notions of graph theory. As well as the different data structures that we will manipulate in the following, notably graphs and trees and their properties, and we end this part with definitions relating to the complexity of algorithms. In the second part of this chapter, we review the work done in the context of our problem and evaluate the results obtained.

Chapter 2 describes a lossless compression algorithm for a particular family of trees namely ordered trees. We give the algorithm of reduction of an ordered tree as well as that of its reconstruction from its compression and we present measures for the evaluation of this compression.

We generalize the method presented for ordered trees compression as well as other work carried out in the context of compression of unordered trees to present a lossless method of compression of semi-ordered trees in Chapter 3. We present some properties of this compression and we end this chapter with an experiment on theoretical plants.

In chapter 4, we consider another type of compression: lossy compression. We first show the limits of lossless tree compression and present the principle of lossy compression which consists in grouping elements that are not totally similar. We then describe the problems encountered with this grouping and how they were remedied in order to build our lossy compression model of unordered labeled trees. Finally, we describe the techniques for evaluating the method and present some applications to plant architecture.

Chapter 1

Definitions and notations

Configurations of nodes and connections occur in several diverse domains. They can be used to represent physical networks, such as roadways, electrical circuits, or organic molecules. but also the different interactions in the ecosystem, sociological relationships, databases, or even the flow of control in a computer program.

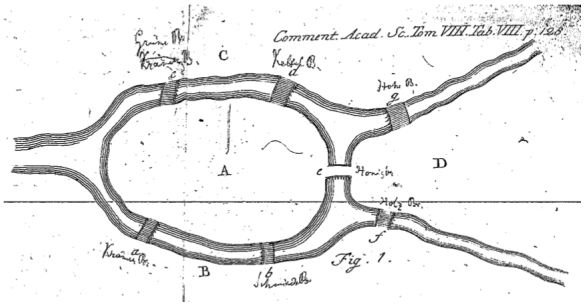
Formally, such configurations are modeled by combinatorial structures called graphs, consisting of two sets called vertices and edges and an incidence relation between them. The vertices and edges may have additional attributes, such as color or weight, or anything else useful to a particular model.

Although the first mention of a graph was not until 1878, The origins of graph theory can be traced back to 1735 when Leonhard Euler (1707–83) presented his solution of the Konigsberg bridges problem. The problem is to determine if it is possible to take a walk crossing all the seven bridges of Konigsberg starting from any place in the city without going through the same one twice and to return to their starting point. Modeling of the problem in graph consists in representing the parts of the city by nodes and the bridges by edges. Fig. 1.1, partly extracted from the original article, illustrates the layout of the bridges in the city and a modeling of the problem in the form of a graph.

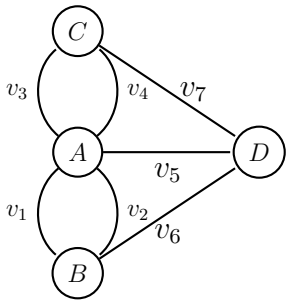
The term graph was introduced in 1878 by James Joseph Sylvester, an English mathematician in an article entitled "Chemistry and Algebra"[34]. Since graph theory has evolved, it now constitutes a discipline in its own right. It is a widely used modeling tool. For example, in biology, metabolic networks are modeled by graphs.

The diversity of the fields of application and the nature of the problems have given rise to different types of graphs. the family of acyclic graphs also called trees occur in several applications and several problems related to this structure (Comparison, inclusion, compression, detection of symmetry ...) have attracted scientific community.

In the first part of this chapter we give a brief introduction to certain basic concepts and results in graph theory, trees and algorithms. For easy reference we also provide a list of commonly used symbols for graph parameters and concepts. In the second part we survey the various works carried out in this context and which form the foundations of our contributions.



(a)



(b)

Figure 1.1: The seven bridges problem of Konigsberg with a diagram of the bridges taken from [12](a) and a modeling of the problem in the form of a graph(b).

1.1 Graph Theory Notations and Definitions

In this section, we introduce definitions related to graph theory. The following definitions are partly taken from [18] and [17].

Definition 1.1.1 (Graph) *A graph $G = (V, E)$ is mathematical structure consisting of two finite sets V and E. where V is a nonempty finite set whose elements are called vertices (or nodes) and E is a set of unordered pairs of elements of V called edges. Each edge has a set of one or two vertices associated to it, which are called its endpoints. An edge is said to join its endpoints.*

Terminologie

- *If vertex v is an **endpoint** of edge e, then v is said to be incident on e, and e is incident on v.*
- *A vertex u is adjacent to vertex v if they are joined by an edge.*
- *Two adjacent vertices may be called neighbors.*
- *The (open) **neighbordhood** of a vertex v is the set of all the neighbors of v. If we add v to this set it becomes the closed neighbordhood of v.*
- *an **isolated vertex** denotes a node with an empty neighborhood*
- ***Adjacent edges** are two edges that have an endpoint in common.*
- *A **proper edge** is an edge that joins two distinct vertices.*
- *A **multi-edge** is a collection of two or more edges having identical endpoints.*
- *A **simple adjacency** between vertices occurs when there is exactly one edge between them.*
- *The **edge-multiplicity** between a pair of vertices u and v is the number of edges between them.*

- *A **self-loop** is an edge that joins a single endpoint to itself.*
- *A **simple graph** is a graph that has no self-loops or multi-edges.*
- *A **trivial graph** is a graph consisting of one vertex and no edges.*
- *A **null graph** is a graph whose vertex- and edge-sets are empty.*
- ***General graph** is a term used to refer to graphs that allows self-loops and multi-edges.*
- *A **loopless graph** is a graph that has no self-loops. (It might have multi-edges.) Sometimes a loopless graph is referred to as a **multigraph**.*
- *The **dipole** D_n is a loopless graph with two vertices and n edges joining them.*
- *The **bouquet** B_n is a graph with one vertex and n self-loops.*

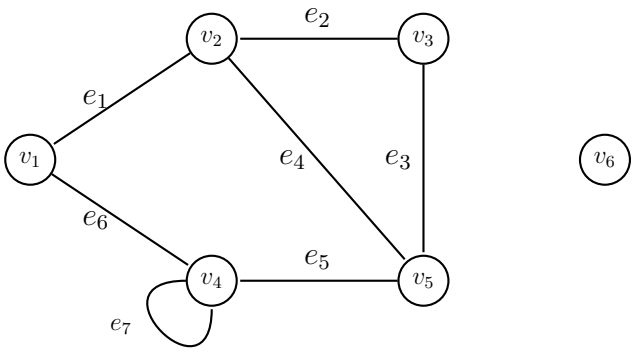


Figure 1.2: A graph $G = (V, E)$ where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$. v_1 and v_2 are the endpoints of edge e_1. v_2, v_3 and v_4 form the neighborhood of v_5. v_6 is an isolated vertex. e_7 is a self-loop

Definition 1.1.2 (A directed edge) *or (arc) is an edge e, one of whose endpoints is designated as the tail, and whose other endpoint is designated as the head. They are denoted head(e) and tail(e), respectively. A directed edge is said to be directed from its tail and directed to its head.*

Definition 1.1.3 (A digraph) *(or directed graph) is a graph each of whose edges is directed.*

Definition 1.1.4 (A mixed graph) *or partially directed graph) is a graph that has both undirected and directed edges. In a mixed graph, using the unmodified term edge avoids specifying whether the edge is directed or undirected.*

Definition 1.1.5 (The indegree and the outdegree of vertex) *The indegree of a vertex v in a digraph noted $deg^-(v)$ is the number of arcs directed to v; the outdegree of vertex v noted $deg^+(v)$is the number of arcs directed from v. Each self-loop at v counts one toward the indegree of v and one toward the outdegree.*

Definition 1.1.6 (The degree of vertex) *The degree of a vertex v in a digraph denoted $deg^(v)$ is the number of proper edges incident on v plus twice the number of self-loops that is $deg^(v) = deg^-(v) + deg^+(v)$*

Definition 1.1.7 (The degree of graph) *The degree of a graph G denoted by $deg^(G)$ is the maximum of its vertices' degrees.*

Definition 1.1.8 (Graph size) *Let $G = (V, E)$ be a graph, note $|V|$ the number of elements in the set V. The size of the G graph noted $|G|$ is equal to $|V|$, so the number of vertices in the G graph.*

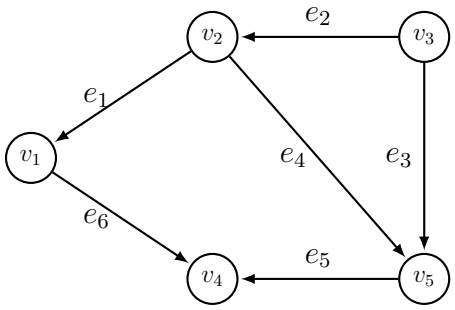


Figure 1.3: A digraph $G = (V, E)$. $deg^-(v_2) = 1$ and $deg^+(v_2) = 2$. $deg(v_2) = 3$. $|G| = 5$, $deg(G) = 3$

Definition 1.1.9 (Subgraph) *A subgraph of a graph $G(V_G, E_G)$ is a graph $H(V_H, E_H)$ such that $V_H \subset V_G$ and $E_H \subset E_G$.*

Definition 1.1.10 (Walk, Trail, Path)

- *A walk in a graph G is an alternating sequence of vertices and edges. $W = v_O, e_o, v_1, e_1, ..., e_n, v_n$*
- *A trail in a graph is a walk such that no edge occurs more than once.*
- *A path in a graph is a trail such that no internal vertex is repeated.*

Terminologie

- *The length of a walk is the number of edges (counting repetitions).*
- *An eulerian trail in a graph G is a walk that contains each edge of G exactly once.*

- *A cycle is a closed path of length at least 1.*
- *An acyclic graph is a graph that does not contain a cycle*
- *A graph is connected if between every pair of vertices there is a walk.*
- *A walk is closed if the initial vertex is also the final vertex; otherwise, it is open.*

Definition 1.1.11 (Connectivity and Distance)

- *A graph is* ***connected*** *if between every pair of vertices there is a walk.*
- *digraph is strongly connected if from each vertex to each other vertex there is a directed walk (otherwise, the graph is said weakly connected).*
- *The distance between two vertices in a graph is the length of the shortest walk between them.*
- *The directed distance from a vertex u to a vertex v in a digraph is the length of the shortest directed walk from u to v.*

Definition 1.1.12 (Directed acyclic graph) *A directed graph G is directed acyclic denoted DAG if for each pair of vertices there (u,v) if is directed path from vertex u to vertex v then there is no path from vertex v to u . In other words there are no cycles in G.*

Definition 1.1.13 (Attributes) *To allow the graph to contain more information beyond edges and nodes and allow them to serve as mathematical models for a wide variety of applications. it is possible to assign labels to the nodes or(and) edges of the graph*

- *A vertex attribute is a function $\mathcal{L}_V$ from the V to a finite set of attribute L_V .*

$$\mathcal{L}_V : V \to L_V$$
$$v \mapsto \mathcal{L}_V(v)$$

- *A edge attribute is a function $\mathcal{L}_E$ from the E to a finite set of attribute L_E*

$$\mathcal{L}_E : E \to L_E$$
$$e \mapsto \mathcal{L}_E(e)$$

.

We denote a labeled graph by $G = (V, E, L_V, L_E)$ where V is the set of vertices, E the set of edges, L_V the set of labels assigned to vertices and L_E the set of labels assigned to edges.

Definition 1.1.14 (Isomorphism) *An isomorphism between two graphs* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *is a vertex bijection* ϕ *from* V_1 *to* V_2 *such that for each* $(x, y) \in E_1$ $,(\phi(x), \phi(y)) \in E_2$.

$$\phi : V_1 \to V_2 / \forall (x, y) \in E_1 \ ,(\phi(x), \phi(y)) \in E_2$$

If there exists an isomorphism between two graphs G_1 *and* G_2*, the two structures are identical then we write* $G_1 \equiv G_2$

Definition 1.1.15 (Isomorphism of a labeled graph) *An isomorphism between two labeled graphs* $G(V_G, E_G, L_{VG}, L_{EG})$ *and* $H(V_H, E_H, L_{VH}, L_{EH})$ *is is both a bijection* ϕ *from* V_G *to* V_H *extended from* L_{VG} *to* L_{VH} *and from* L_{EG} *to* L_{EH} *means that for each* $x \in V_G$ $,\mathcal{L}_{V_G}(x) = \mathcal{L}_{V_H}(\phi(x))$ *and*

$$(x, y) \in E_G = \begin{cases} (\phi(x), \phi(y)) \in E_H \\ \mathcal{L}_{E_G}((x, y)) = \mathcal{L}_{E_H}((\phi(x), \phi(y))) \end{cases}$$

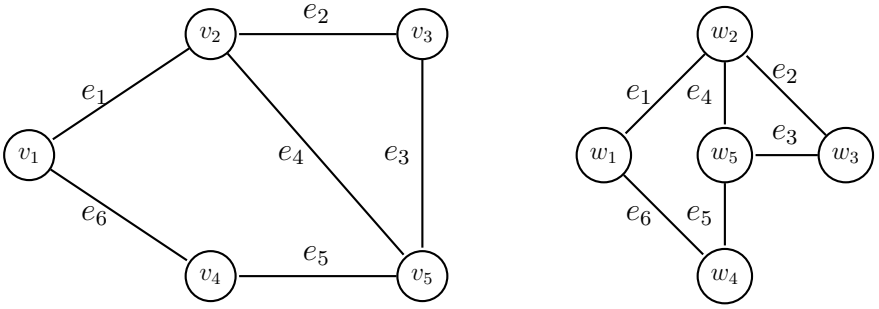


Figure 1.4: The two graphs are isomorphic if we only consider the structures but they are not if we consider the labels.

1.2 Definitions related to trees

The family of particular graphs without cycles called trees has always attracted the attention of the scientific community. The English mathematician Arthur Cayley and the Hungarian mathematician George Pólya presented in the early 1900s applications of this structure in chemistry. Nowadays, Trees are among the most important nonlinear structure arising in computer science . They occur in several diverse areas such as genealogical studies, computer vision and pattern recognition [33], programming compilation and natural language processing [8].

Definition 1.2.1 (Tree) *A tree is a connected directed acyclic graph[17].*

Definition 1.2.2 (Forest) *A forest is a (not necessarily connected) acyclic graph. A forest can be seen as a collection of trees[17].*

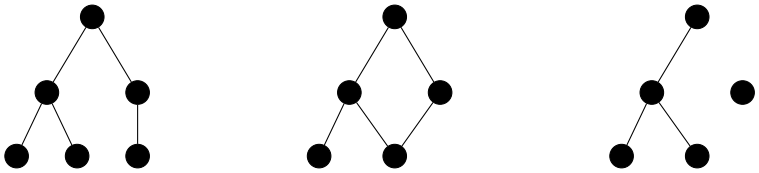


Figure 1.5: The graph on the left is a tree; the other two graphs are not.

Definition 1.2.3 (Root) *A root of a tree $T(V, E)$ denoted $root(T)$ is a node $v \in V$ with $deg^-(v) = 0$*

Definition 1.2.4 (Internal node) *Let v be a node of a tree, if $deg^+(v) > 0$ then v is an internal node.*

Definition 1.2.5 (Leaf) *Let v be a node of a tree T, if $deg^+(v) = 0$ then v is a leaf. The set of leaves of T is denoted by $leaves(T)$.*

Definition 1.2.6 (Parent) *Let $e = uv$ an edge of a tree, then u is called parent of v denoted $parent(v)$.*

Definition 1.2.7 (child) *Let v be a node of a tree $T = (V, E)$, the nodes $u_i \in V$ such that $parent(u_i) = v$ forms the set of children of v denoted $child(v)$.*

Definition 1.2.8 (sibling) *Two nodes of a tree are sibling if they have the same parent node.*

Definition 1.2.9 (Ancestor, descendant) *A vertex u is called the ancestor of another vertex v if a sequence of vertices $(w_1, w_2, ...w_n)$ exists such that $w_1 = u, w_n = v$ and for each consecutive pair of vertices (w_i, w_{i+1}) in the sequence $w_i = parent(w_{i+1})$. respectively v is said to be descended from u and u, v are said to be in relation ancestor-descendant. we denote by $Ancestors(v)$ the set of ancestors of node v and by $Descendant(v)$ the set of its descendants.*

Definition 1.2.10 (Subtree) *Let T be a tree, a subtree is a connected subgraph of T. The subtree denoted by $T[v]$ with $v \in V$ is the maximal subtree rooted at v containing v and the set of its descendants.*

Definition 1.2.11 (Subforest) *let T be a tree, a sub-forest is a subgraph of T denoted $F[v] = (V', E')$ with $v \in V$ such that V' is the set of nodes of $T[v] \backslash \{v\}$ and E' the set of arcs of $T[v]$ having no incoming or outgoing extremity v. otherwise, the forest under node v is the set of trees rooted at the child of v:*

$$F[v] = \{T[v_i], v_i \in child(v)\}$$

Definition 1.2.12 (Depth) *The depth of a node v in the tree T is the length of the path from $root(T)$ to v.*

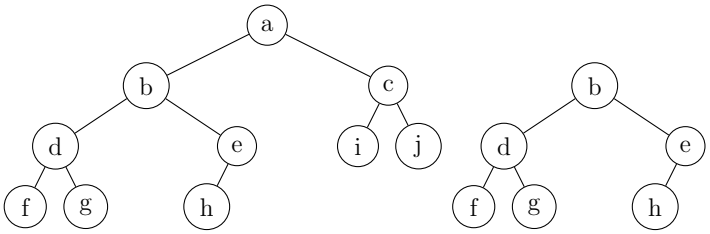


Figure 1.6: a tree T (a) and its subtree $T[b]$rooted at vertex b (b), $depth(d) = 2, height(d) = 1$ $root(T) = a$, $height(T) = 3$, $leaves(T) = \{f, g, h, i, j\}$, $child(d) = \{f, g\}$, $parent(d) = b$, $Ancestors(d) = \{b, a\}$, $Descendants(b) = \{d, f, g, e, h\}$, d and e are sibling.

Definition 1.2.13 (Height) *Let v be a vertex of a tree $T = (V, E)$, the height of the vertex v is defined as the distance between v and the furthest leaf of $T[v]$*

We extend the definitions of the depth, the height and the degree of vertices to Trees such that $depth(T)$,$height(T)$ and $deg(T)$ denotes the maximum depth, height and degree of its vertices.

The diversity of the fields of application of the trees and the nature of the trees already studied have led to a division of the trees in several ways and on several criteria; we are interested in the families of trees resulting from the division according to the order between sibling.

Definition 1.2.14 (Binary relation) *A binary relation $*$ from a set E to a set F is defined by a subset H of the cartesian product $E \times F$ .A collection of pairs (x, y) whose first component is in E and the second in F. For any pair $(x, y) \in H$, x is said to be in relation to y and we denote by $x * y$.*

Definition 1.2.15 (Order relation) *In mathematics, an order relation $\leq$ on a set E is a binary relation that allows its elements to be comparable. Formally an order relation is :*

- $\forall x \in E, x \leq x$ *(reflexive)*
- $\forall x, y \in E^2, (x \leq y \text{ and } y \leq x) \Rightarrow x = y$ *(antisymmetric)*
- $\forall x, y, z \in E^3, (x \leq y \text{ and } y \leq z) \Rightarrow x \leq z$ *(transitive)*

A set endowed with an order relation is an ordered set.
the order is total and the set is totally ordered if in addition:

$$\forall x, y \in E^2 \quad x \leq y \text{ or } y \leq x.$$

Definition 1.2.16 (Total semi-order relation) *A binary relation $\sqsubseteq$ is of total semi-order if it is reflexive, transitive and total. A total semi-order relation $\sqsubseteq$ on a set E implies:*

$$\forall x, y \in E^2 : x \sqsubseteq y \Rightarrow \begin{cases} x = y \\ x \leq y \quad \text{otherwise.} \end{cases}$$

Definition 1.2.17 (Unordered Tree) *An unordered tree is a tree in which no significant order exists between siblings [40].*

Definition 1.2.18 (Ordered Tree) *An ordered tree is a pair $(T, S_{\leq})$ where T is a rooted tree and $S_{\leq}$=$\{\leq_x, x \in T\}$ is a set of total order relations such that for any vertex x of T, $\leq_x$ is a total order relation on child(x) [29].*

Definition 1.2.19 (Semi-ordered tree) *A semi-ordered tree is a pair $(T, S_{\sqsubseteq})$ where T is a tree and $S_{\sqsubseteq} = \{\sqsubseteq_x, x \in T\}$ is a set of total semi-order relations such that for any vertex x of T, $\sqsubseteq_x$ is a total semi-order relation on child(x) [26].*

Remark *An isomorphism on ordered or semi-ordered trees must respect the meaningful order relation on the siblings.*

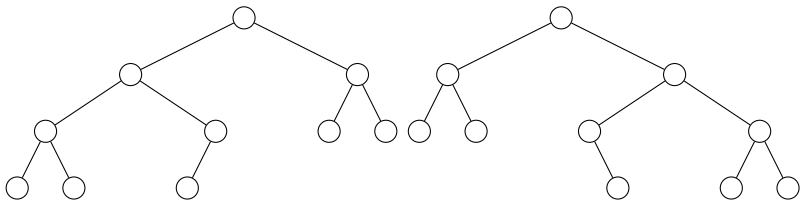


Figure 1.7: (a) T_1 tree. (b) T_2 tree. if we view the trees as unordered trees T_1 and T_2 are isomorphic. However, in the case of ordered trees, they are not isomorphic.

1.3 Definitions related to algorithms

The term algorithm appeared around the 9th century to designate a series of operations aimed at solving a problem. The evaluation of the efficiency of an algorithm is measured with two measures: time complexity and space complexity.

Definition 1.3.1 (The space complexity) *The space complexity of an algorithm is a function defined as follows:*

$$\begin{aligned} f : \mathbb{N} &\to \mathbb{N} \\ n &\mapsto f(n) \end{aligned}$$

where $f(n)$ is the amount of memory space required to complete its execution and n the input size.

Definition 1.3.2 (The time complexity) *The space complexity of an algorithm is a function defined as follows:*

$$\begin{aligned} f : \mathbb{N} &\to \mathbb{N} \\ n &\mapsto f(n) \end{aligned}$$

where $f(n)$ the amount of time taken by an algorithm to run and n the input size.

In this thesis we will be mainly concerned with time complexities of algorithms but also complexity at worst, i.e. maximum value of complexity.

Definition 1.3.3 (big-Oh *O*) *A function $f(n)$ is said to be $O(g(n))$ if there exist some constant c and some nonnegative integer n_0 such that:*

$$\forall n \geq n_0, \exists\ c > 0 : f(n) \leq c \times g(n)$$

Definition 1.3.4 (Polynomial problem (algorithm)) *The complexity of an algorithm is said to be polynomial if it is $O(n^k)$, for some integer k; we also say that the algorithm itself is polynomial.*

Definition 1.3.5 (NP problem (algorithm)) *The class NP is the class of problems which can be verified in polynomial time (verifies if a candidate solution is a solution to a given instance of the problem).*

1.4 State of the art

The comparison of the trees is an essential process which returns in several application (programming compilation and natural language processing[8], vision, pattern recognition, molecular biology[32], waveform correlation[7] automatic error recovery and correction for programming languages [35]). Several methods exist to compare two tree structures, we are interested in a particular distance called the edition distance for other tree comparison method and the problems related to it the reader can then refer to [5]. Initially defined for sequences [24] before it was generalized for tree structures. The edit distance is defined as the cost of transforming an initial tree to a target tree using simple primitive operations defined by [35] (changing the label of a node, deleting or adding a node) Since then, for some applications, new operations have been added [22][1][21].

Several versions of this distance have been developed to respond to different problems and according to the tree family used for each problem. The complexities are highly variable, ranging from polynomial for the case of ordered trees [35], [42], [23], [10], [9] to MAX-SNP-hard for the cases of semi-ordered and unordered trees [43],[40] ,[26].In this section we survey different versions of edit distance and achieved results.

Our problem is also related to tree compression. Indeed, the wide variety of algorithms resolving problems on tree remains less effective in front of the size of data which does not stop increasing. This requires not only effective processing, but also effective representation which takes into consideration the properties of this data structure. Tree structures are characterized by exact or approximate internal redundancy at different scales. A way to address the data storage problem is to take advantage of these redundancies by representing them only once to build a reduced form that keeps the major original tree properties and achieves a better algorithmic conditions.

In the literature, data compression is an operation which consists in passing from an initial representation of an object to a more compact representation, that is to say

requiring less storage space. A compression algorithm takes as input a data X and generates a representation X_c which requires fewer bits, and there is a reconstruction algorithm which generates from the compressed representation X_c a reconstruction Y (Fig 1.8). Based on the reconstruction requirements, data compression techniques can be divided into two classes: lossless compression, in which X is identical to Y, and lossy compression which allows X to be different from Y [31]. In this thesis, we are concerned with both lossy and lossless compression. Several works were carried out in this context, in particular the unordered trees lossless compression, we also present in this section achieved results.

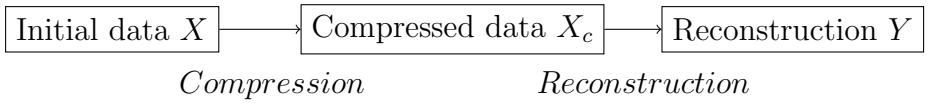


Figure 1.8: Compression and reconstruction.

1.4.1 *Edit distance*

Several attempts have been proposed to solve the treeproblem compression problem. We also present in this section the different works presented in this context and which form the basis of our contribution.

Definition 1.4.1 (Distance) *A distance d on a set E is a function $d : E \times E \to \mathbb{R}_+$ such that $\forall x, y \in E^2$:*

- $d(x, y) \geq 0$ *(not negative)*
- $d(x, y) = 0 \Leftrightarrow x = y$ *(separation)*
- $d(x, y) = d(y, x)$ *(symmetric)*
- $d(x, y) \geq d(x, z) + d(z, y)$ *(triangle inequality)*

Remark *x is all the closer to y as the distance $d(x, y)$ between x and y is small.*

Definition 1.4.2 (Edit Operations) *To transform a tree T_1 to T_2 we consider the basic edit operations defined by Tai[35] as follows:*

- ***relabel:*** *Change the label of a node v in T_1.*
- ***delete:*** *Delete a node v in T_1 means making $child(v)$ become the children of $u = parent(v)$.*
- ***insert:*** *Insert a node v as a child of u in T_1 is making v the parent of a consecutive subsequence of the children of u.*

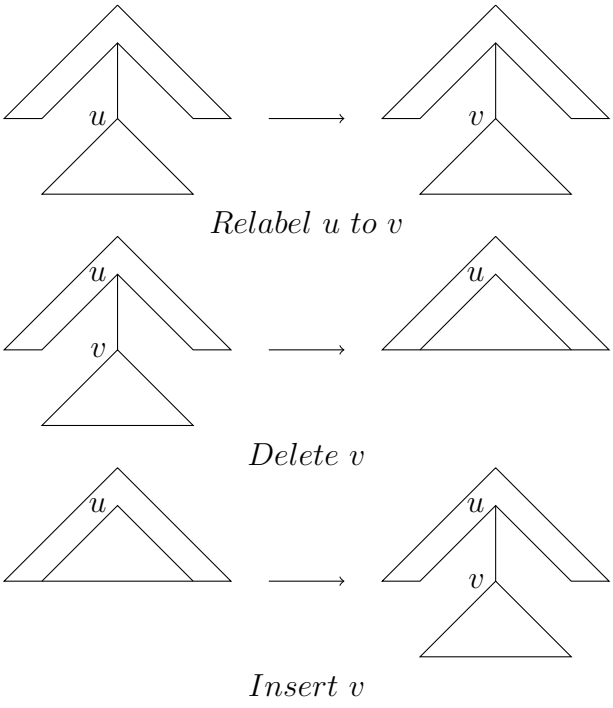


Figure 1.9: Edit operations on trees

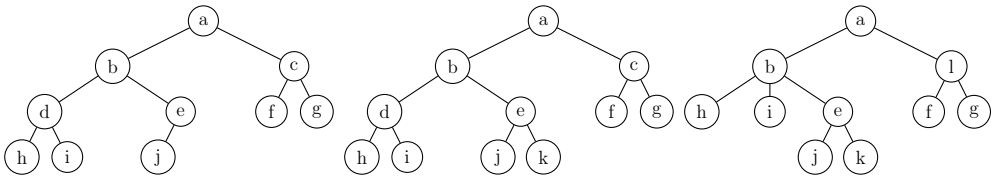


Figure 1.10: Transforming (a) into (c) via edit operations. (a) A tree. (b) The tree after inserting the node labeled k. (c) The tree after deleting the node labeled d and relabeling c to l .

Definition 1.4.3 (Cost of an edit operation) *Let $T_1 = (V_1, E_1, L_{V_1})$ and $T_2 = (V_2, E_2, L_{V_2})$ be labeled trees (we only consider the labels on the vertices), we also consider the set $\sum = \{(L_{V_1} \cup L_{V_2}) \cup \lambda\}$ the finite set of labels of T_1 and T_2 to which we add the white symbol denoted λ.*

We define a cost function γ such as:

$$\gamma : (\sum \times \sum) \setminus (\lambda, \lambda) \to \mathbb{R}$$
$$(a, b) \mapsto \gamma(a, b)$$

A function that assigns a cost to each pair of labels. we assume that γ is a distance Def. 1.4.1

For simplicity we keep the notation of Tai [35] each edit operation is represented by $(l_1 \to l_2)$, where $(l_1, l_2) \in (\sum \times \sum) \setminus (\lambda, \lambda)$. The operation is a relabeling if $l_1 \neq \lambda$ and $l_1 \neq \lambda$, a deletion if $l_2 = \lambda$, and an insertion if $l_1 = \lambda$. We extend the notation such that $(u \to v)$ for nodes v and w denotes $(\mathcal{L}_V(v) \to \mathcal{L}_V(w))$.

Then we define the cost of an edit operation by setting $\gamma(u \to v) = \gamma(\mathcal{L}_V(u), \mathcal{L}_V(v))$

The cost of a sequence $S = s_1, ..., s_k$ *of operations is given by* $\gamma(S) = \sum_{i=1}^{k\gamma(s_i)}$. *The edit distance,* $D(T_1, T_2)$, *between* T_1 *and* T_2 *is formally defined as:*

$D(T_1, T_2) = min\{\gamma(S)|S$ *is a sequence of operations transforming* T_1 *into* $T_2\}$.

Definition 1.4.4 (Edit distance mapping)

Let $T_1 = (V_1, E_1, L_1)$ *and* $T_2 = (V_2, E_2, L_2)$ *be two labeled trees. A* mapping *M from* T_1 *to* T_2 *is a graphical description of how a sequence of edit operations transforms* T_1 *into* T_2, *ignoring the order in which the edit operations are applied.*

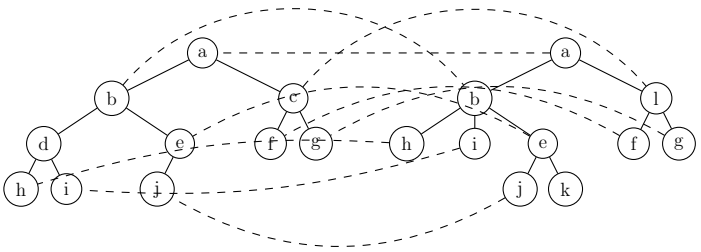
Formally, a mapping *M from* T_1 *to* T_2 *is any set of ordered pairs in* V_1 *and* V_2, *namely* $M \subset (V_1 \times V_2)$. *Each pair* $(v_1, v_2) \in M$ *is a relabeling operation of* v_1 *to* v_2. *Each vertex v in I (resp J) the set of vertices of* V_1 *(resp* V_2*) which do not appear in M gives rise to a delete (resp insert) operation during the transformation.*

Let $\mathcal{M}$ *be the set of all* T_1 *to* T_2 *mappings. A* mapping $M \in \mathcal{M}$ *is* valid *if it satisfies the following properties* $\forall(x_1, y_1), (x_2, y_2) \in M$*:*

1. $x_1 = x_2 \Leftrightarrow y_1 = y_2$
2. *Preserve the ancestor order:* x_1 *is an ancestor of* x_2 *iff* y_1 *is an ancestor of* y_2.
3. *Preserves the predefined order between siblings if it is an ordered or semi-ordered tree (Def. 1.2.18 Def.1.2.19)*

According to the definition of the costs of the edit operation we define the cost of a mapping as:

$$\gamma(M) = \sum_{(x,y)\in M} \gamma(x \to y) + \sum_{x \in I} \gamma(x \to \lambda) + \sum_{y \in J} \gamma(\lambda \to y)$$



(a) (b) (c)

Figure 1.11: The mapping corresponding to the edit script in Fig 1.10

Definition 1.4.5 (Edit distance) *With the mapping definition we can formally define the edit distance between two trees* T_1 *and* T_2 *as follows:*

$$\mathcal{D}(T_1, T_2) = min\{\gamma(M)|M \text{ is a valid mapping from } T_1 \text{ to } T_2\}$$

1.4.2 *Ordered trees edit distance*

The ordered edit distance problem firstly introduced by Tai[35] as a generalisation of the string edit distance. Tai presented a solution of order of

$$O(|T_1||T_2|depth(T_1)^2 depth(T_2)^2)$$

further improved by Zhang and Sacha[42] in the limite of

$$O(|T_1||T_2|min(|leaves(T_1)|, depth(T_1))min(|leaves(T_2)|, depth(T_2)))$$

optimized again by Klein[23] to achieve better worst-case time complexity bound of

$$O(min(|T_1|^2|T_2|log|T_2|, |T_2|^2|T_1|log|T_1|).$$

We describe in the following briefly these algorithms.

The tree-to-tree correction problem [35] Tai present a generalization of the well-known string edit distance problem [38] using a dynamic programing. In adapting the string algorithm to trees, Tai used a preorder traversal in which the first node visited is the root, then all subtrees are traversed in preorder from left to right. The root of the tree remains unchanged during editing. During the traversal, each node is numbered, so that node 1 is always the root of the tree and node i is the node that was visited after node $i-1$. Tai then uses mapping (Def 1.4.4) to catalog which operations are used to transform T to T'. As in the string to string algorithm, a distance matrix M_D is used and for each location in the matrix $M_D[i,j]$ the cost of deleting $T[i]$, cost of inserting $T'[j]$and the cost of mapping T to T' are assessed and the minimum selected. Solving the problem ultimately comes down to solving a constant number of smaller sized subproblems. Each new subproblem can be computed in constant time. Hence, the time complexity is bounded by the number of subproblems.

A similar edit distance was given by Tanaka in [36] with a weak adaptation in the definition of the mapping gives result in $O(|T_1|\times|T_2|\times max\{|leaves(T_1)|, |leaves(T_2)|\})$ in time and $O(|T_1| \times |T_2|)$ in space. An other algorithm using a post-order traversal of the tree and having a better time complexity, and much better space complexity, is given in [42]. The next paragraph discusses that algorithm.

Simple fast algorithms for the editing distance between trees and related problems The principle idea is to improve the previous algorithm by not carrying the calculation on all the nodes of the tree, but only on certain strategic nodes.

Zhang and Shasha used post-order traversal to label each node, in this traversal method the root node is visited last, hence the name. First, we traverse the left subtree, then the right subtree and finally the root node.

Zhang and Shasha Define the keyroots of a rooted ordered tree T as follows:

$$\text{keyroot(T)}=\{\ \text{root(T)}\bigsqcup\text{v}\in\text{V(T)}—\text{v has a left sibling}\}$$

The keyroots of a tree allows the algorithm to separate the concept of tree distance and forest distance. if a node is not keyroot its forest distance to all other nodes is equal to its tree distance otherwise we use the following lemma to calculate it:

Lemma 1.4.1 *Let $F - v$ denote the forest obtained by deleting v from the forest F. Furthermore, let $F - T[v]$ denote the forest obtained by deleting v and all descendants of v. The following lemma provides a way to compute edit distances D for the general case of forests.*

Let F_1 and F_2 be ordered forests and γ be a metric cost function defined on labels. Let v and w be the rightmost (if any) roots of the trees in F_1 and F_2 respectively. We have:

$D(\theta, \theta) = 0$
$D(F_1, \theta) = D(F_1 - v, \theta) + \gamma(v \to \lambda)$
$D(\theta, F_2) = D(\theta, F_2 - w) + \gamma(\lambda \to w)$

$$D(F_1, F_2) = min \begin{cases} D(F_1 - v, F_2) + \gamma(v \to \lambda) \\ D(F_1, F_2 - w) + \gamma(\lambda \to w) \\ D(F_1[v], F_2[w] + D(F_1 - T_1[v], F_2 - T_2[w]) - \gamma(v \to w) \end{cases}$$

Klein's algorithm[23] A key concept in the algorithm is the decomposition of a rooted tree T into disjoint paths called heavy paths.

Each node of T is considered as either heavy or light as follows. The root is light. For each internal node the child having the largest subtree (the leftmost in case of ties) is heavy. The remaining children are light. An edge is heavy iff it leads to a heavy node otherwise it is considered as light edge. The light depth of a node v, $ldepth(v)$, is the number of light edges on the path from v to the root.

By removing the light edges T is partitioned into heavy paths. Authors used this heavy path decomposition to to guide the choice between two recursive solutions while applying Lemma 1.4.1.

For ordered trees T_1 and T_2 the edit distance problem can be solved in time and space $O(|T_1|^2|T_2|log|T_2|)$.

Klein also showed that his algorithm can be extended within the same time and space bounds to the unrooted ordered edit distance problem between T_1 and T_2, defined as the minimum edit distance between T_1 and T_2 over all possible roots of T_1 and T_2.

1.4.3 *Unordered trees edit distance*

The unordered edit distance problem has been shown to be MAX SNP-hard [41]. Problems of the Max-SNP-hard complexity class are at least as hard as the set of all Max-SNP problems. Algorithms known to solve problems of the Max-SNP-hard complexity class have exponential time complexity in the size of the input and do not have an approximation that can be computed in polynomial time. An algorithm was proposed by Zhang et al.[43] to solve general editing between two unordered trees, the time complexity is $O(|T_1||T_2| + k!2^k(k^3 + deg(T_2)^2)|T_2|)$. with $k = |leaves(T_1)|$.

However for some applications, it is possible to take into account additional constraints for the mapping in order to reduce the complexity of the problem so that it is no longer Max-SNP-hard.

Constrained edit distance In [40][39] Zhang introduced the constrained edit distance, denoted by D_c a distance under restricted mapping. The idea of a restricted mapping between two trees T_1 and T_2, is that two disjoint subtrees from T_1 tree must be mapped to two disjoint subtrees in T_2. a formal definition of contrained mapping is given in Def. 1.4.6.

Definition 1.4.6 (Constrained edit distance mapping) *A triple (M, T_1, T_2) is a constrained edit distance mapping from T_1 to T_2, where M is a set of ordered pairs of vertices satisfying:*

1. *M is an edit distance mapping.*
2. *For any triple (u, u'), (v, v'), and (w, w') in M let $lca(u, v)$ and $lca(u', v')$ be the least common ancestor of u, v and u', v' (resp), then $lca(u, v)$ is a proper ancestor of w iff $lca(u', v')$ is a proper ancestor of w'.*

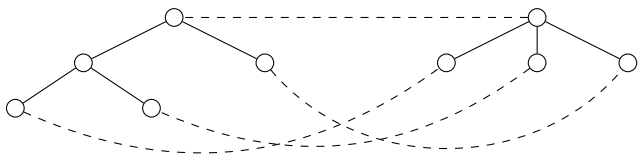


Figure 1.12: Invalid mapping in case of constrained edit distance.

Zhang's algorithm [40] proposes a solution to the constrained edit distance problem with a complexity of $O(|T_1||T_2|(Deg(T_1) + Deg(T_2))log((Deg(T_1) + Deg(T_2)))$, the equations are given in Def.1.4.7 and Def.1.4.8. Note that the problem reduces to a matching problem which is solved as a min cost flow problem by the algorithm of Edmons and Karp [11] then improved by Tarjan [37].

Definition 1.4.7 (Constrained edit distance between sub-forests) *let T_1 and T_2 be two unordered trees, $F_1[u]$ and $F_2[v]$ two forests induced by u and v, the constrained edit distance D_c between the two sub-forests is defined as follows:*

$$D_c(F_1[u], F_2[v]) = min \begin{cases} min\{\gamma(M_c)\} \\ D_c(\theta, F_2[v]) + min_{w \in child(v)}\{D_c(F_1[u], F_2[w]) - D_c(\theta, F_2[w])\} \\ D_c(F_1[u], \theta) + min_{w \in child(u)}\{D_c(F_1[w], F_2[v]) - D_c(F_1[w], \theta)\} \end{cases}$$

Definition 1.4.8 (Constrained edit distance between subtrees) *let T_1 and T_2 be two unordered trees rooted in u and v, the constrained edit distance D_c between the two subtrees is defined as follows:*

$$D_c(T1[u], T_2[v]) = min \begin{cases} D_c(F_1[u], F_2[v]) + \gamma(u \to v) \\ D_c(\theta, T_2[v]) + min_{w \in child(v)}\{D_c(T_1[u], T_2[w]) - D_c(\theta, T_2[w])\} \\ D_c(T_1[u], \theta) + min_{w \in child(u)}\{D_c(T_1[w], T_2[v]) - D_c(T_1[w], \theta)\} \end{cases}$$

1.4.4 *Semi-ordered trees edit distance*

The case of semi-ordered trees was treated by Ouangraoua and Ferraro [26]. The general problem remains Max-SNP-hard but they proposed a solution to constrained editing in which distinct subtrees must be mapped to distinct subtrees but semi-order must also be preserved. The proposed method is in a time of $O(|T_1||T_2|(deg(T_1) + deg(T_2))log_2(deg(T_1) + deg(T_2)))$.

1.4.5 *Lossless DAG reduction of labeled trees*

In [16] authors presented 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 of $O(T^2 deg(T) log(deg(T)))$. Both approaches start by sorting the subtrees of the input tree into equivalence class each by its principle:

- The first is based on signatures calculated for each node recursively from bottom to top, the signature of a node v describes the signatures of its children. Two nodes u and v having the same signature means that the two subtrees $T[u]$ and $T[v]$ are equivalent and belong to the same equivalence class.

- For the second, the key idea is to start from a distance calculation between the input tree T and 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. A null distance between two subtrees $T[u]$ and $T[v]$ denotes isomorphic structures $T[u] \equiv T[v]$.

Consider now a labeled tree $T = (V, E, L)$, two vertices v_i and v_j are equivalent means that $T[v_i] \equiv T[v_j]$ and we note by extension $v_i \equiv v_j$. For each vertex v_i from V we denote by $[v_i]$ the equivalence class of v_i. Let the graph $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$ obtained from T using the above equivalence relation on V where $V_\mathcal{Q}$ is the set of equivalence classes and $E_\mathcal{Q}$ is a set of pairs of equivalence classes such that $(X, Y) \in E_\mathcal{Q}$ if and only if $(x, y) \in E$ and $[x] = X$ and $[y] = Y$. The graph Q(T) is a DAG ([16], Proposition 1) that is a connected directed graph containing no directed cycles.

Let $(X, Y) \in E_\mathcal{Q}$ and $\mathcal{L}_D(X, Y)$ the number of children of an instance of the class X from the class Y. The quotient graph $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$ augmented with labels $\{\mathcal{L}_D(X, Y), (X, Y) \in E_\mathcal{Q}\}$ defines a reduction $\mathcal{R}(T)$ of the tree T.

Fig 1.13.a shows an example of labeled unordered tree reduction. Note that $\mathcal{R}(T)$ is a lossless compression meaning that the exact reconstruction of T from $\mathcal{R}(T)$ is always possible.

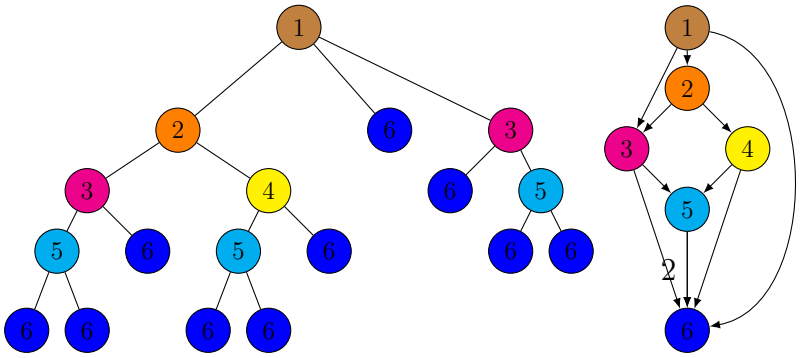


Figure 1.13: At left an unordered labeled tree T, in right its reduction $\mathcal{R}(T)$. Roots of isomorphic subtrees in T and the corresponding equivalence classes nodes in $\mathcal{R}(T)$ are colored identically.

1.4.6 *Lossless DAG reduction of a forest*

In ([2, Algorithm 1]) authors presented the lossless dag reduction of forest, an approach that exploit the previous idea. Let T_F be the super-tree obtained from a forest $F = (T_1, T_2, ..., T_n)$ by placing each T_i as a subtree of an artificial root s then the DAG reduction of the forest F is $\mathcal{R}(F) = \mathcal{R}(T_F)$. Fig.1.14 shows an example of a DAG reduction $\mathcal{R}(F)$ of unordered trees forest F, and that is also possible to compute $\mathcal{R}(F)$ from the forest of DAGs $F_D = (D_1, D_2, .., D_n)$ with $D_i = \mathcal{R}(T_i)$ for $T_i \in F$.

1.5 Conclusion

In this chapter, we have presented the data structures mainly concerned by this manuscript, namely tree structures and graphs. We have also reviewed the work carried out in the context of our problem and which form the foundations of our contributions. The importance of the tree structure is due to the fact that it is used in several different fields to represent nonlinear data, this variety has led to the development of a large panorama of algorithms to solve different problems. the optimization of data processing does not just come down to the optimization of algorithms but also the search for a better representation of these data. We are presenting in the next chapter our first contribution in this context namely the compression of ordered trees.

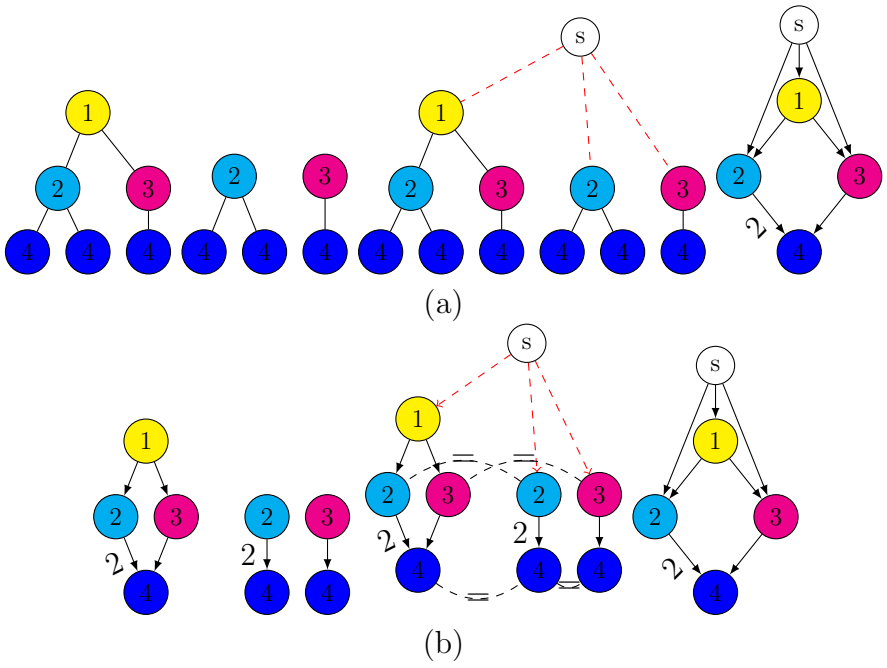


Figure 1.14: A step by step illustration of the construction of the reduction of a forest (a) from the trees of this forest (b) and from their DAG reductions.

Chapter 2

Lossless Compression of Ordered Tree

2.1 Introduction

Ordered trees are a family of trees characterized by a significant order between the siblings. We present in this chapter our first contribution: a lossless compression of ordered trees in DAG. The proposed solution is a generalization of Godin's work presented in [16] for ordered trees. We also present the tree reconstruction algorithm from the compression Dag. This work was presented at the International Conference on Artificial Intelligence and Information Technology [19].

Lossless tree compression can be split into different tasks. First, it is necessary to identify the redundancies that can be dealt with. Once this step is over, we can build a quotient graph where we represent the initial tree without its redundancies and finally an annotation of this graph which allows us to build the lossless compression of this tree. We detail each of these tasks in the following. The method is then evaluated to determine its effectiveness. Note that only the tree structure is considered in this solution. In what follows the trees are considered as unlabeled trees and the order of representation of the siblings in the figures is considered as their order.

2.2 Identification of redundancies

The key idea of compression is to group similar parts of the tree in order to represent them only once. we adopt the signature approach presented by Godin and al. in [16] to determine if two subtrees are equivalent, and for this we define the signature of a node in an ordered tree.

Definition 2.2.1 (Signature of a vertex) *The signature of a vertex in an ordered tree is defined recursively as an ordered set of his children's signatures for the root and internal vertices and* $\{\theta\}$ *for a leaf.*

for simplicity we denote each unique ordered set of signatures by a unique integer and the signature of a node becomes an ordered set of integers such that each integer denotes a signature of a child.

We use $\sigma(x)$ to denote the signature of the vertex x. Note that this definition only considers unlabeled trees and changes in the context of labeled trees. Fig.2.1 shows an example of an ordered tree with its set of vertices signatures.

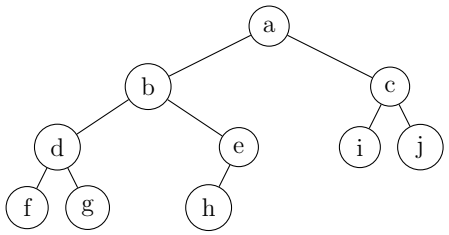


Figure 2.1: An ordered tree. $\sigma(f) = \sigma(g) = \sigma(h) = \sigma(i) = \sigma(j) = \{\theta\}$ simplified 0, $\sigma(d) = \sigma(c) = \{0, 0\}$ simplified 1, $\sigma(e) = \{0\}$ simplified 2, $\sigma(b) = \{1, 2\}$ simplified 3, $\sigma(a) = \{3, 1\}$ simplified 4.

Signatures recursively characterize vertices which have identical structure, based on the signatures of their children. Proposition.2.2.1 gives the fundamental characteristic of tree isomorphism on which rests the ordered tree reduction.

Proposition 2.2.1 $\forall x, y \in T, T[x] \equiv T[y] \iff \sigma(x) = \sigma(y)$.

Proof *a) Let us consider two isomorphic subtrees $T[x]$ and $T[y]$, in other terms $[x] = [y]$. Then by definition $\sigma(x) = \sigma(y)$.*

b) Let us consider two vertices x and y in a tree T such that $\sigma(x) = \sigma(y)$. This implies that $|child(x)| = |child(y)|$, and $\forall x' \in child(x)$, $\forall y' \in child(y)$, where x', y' are of the same order in respectively $child(x)$ and $child(y)$, $T[x']$ and $T[y']$ are isomorphic. Thus $T[x]$ and $T[y]$ are isomorphic as well.

The signatures of the nodes of an ordered tree are computed recursively is built recursively from the bottom up. Alg.1, of Appendix.A, computes the list of signatures of all T vertices.

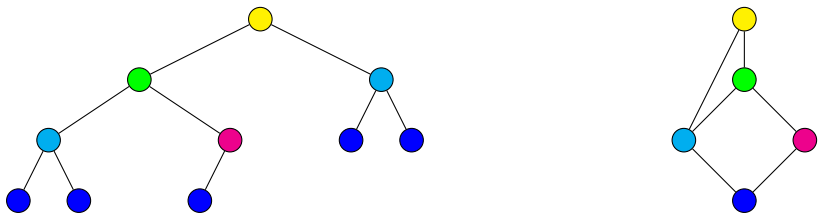
Proposition 2.2.2 *Ordered tree vertices signatures can be computed by function add_signatures in worst case time $O(|T|^2)$.*

2.3 Computing quotient graph

Now that we know how to annotate the vertices of an ordered tree by signatures and that we can determine if two subtrees are isomorphic, we exploit the equivalence relation induced by the isomorphim to build a graph where we only represent redundancy only once.

We build the quotient graph in the same way as for unordered trees. let an ordered tree $T = (V, E)$ and two vertices x and x' such that $\sigma(x) = \sigma(x')$ Consequently

$x \equiv x'$ (Proposition.2.2.1). Let $[x]$ denote the equivalence class of x, all elements of $[x]$ have the same signature as x. The equivalence classes of the isomorphism relation on the set V form a set V_Q. We form a set of pairs of equivalence class E_Q so that $(I, J) \in E_Q$ where I and J are two equivalence classes if there is a $(x, y) \in E$ and $[x] = I$ and $[y] = J$. V_Q and E_Q define the quotient graph $G_Q(T) = (V_Q, E_Q)$. Figure shows an example of quotient graph construction.



(a) (b)

Figure 2.2: (a) An ordered tree where vertices having the same signatures are colored by the same colors. (b) the resulting quotient graph with the class colors shown in the same color as their nodes.

Proposition 2.3.1 *Let T be a tree, then $G_Q(T)$ is a DAG.*

$G_Q(T)$ 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 $(x_1, ..., x_k)$ in T corresponds to a path $([x_1], ..., [x_k])$ with identical length in $G_Q(T)$.

2.4 Computing tree reduction

The quotient graph $G_Q(T)$ obviously condenses the structural contained in the tree T. However, for each vertex the children order is very important in ordered trees and $G_Q(T$ does not contain this information. We thus consider how to augment the definition of $G_Q(T)$ so that the resulting condensed representation can be used to reconstruct the original tree. For this, we shall associate an ordered list to each edge of E_Q to obtain augmented edges.

Definition 2.4.1 (Augmented edge) *In the ordered list $[a_1, a_2..a_n]$ associated to an edge (I, J) and denoted by $l_{(I,J)}$, each a_i represents the order of a child of class J among the other sibling in the initial tree T.*

To illustrate this definition, starting from the quotient graph of Fig.2.2, we present in Fig.2.3 the complete reduction including the ordered lists associated to each edge.

Definition 2.4.2 (Ordered Tree Reduction) *Given an ordered tree T and its quotient graph $G_Q(T) = (V_Q, E_Q)$ regarding the isomorphism relation on T, we define its reduction graph $R(T)$ as a DAG $D = (node_list, edge_list)$ in which $node_list = V_Q$ and $edge_list = \{(e, l_e), \forall e \in E_Q\}$.*

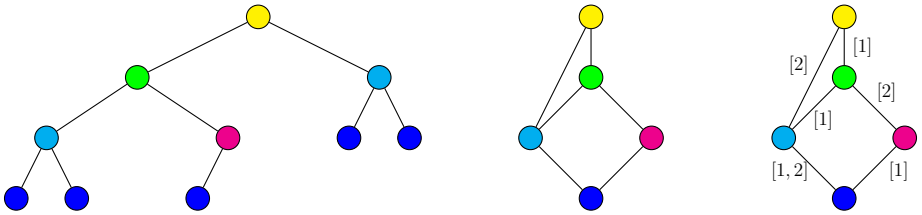


Figure 2.3: (a),(b) Fig.2.2, (c) the quotient dag with augmented edges.

Alg.3, in Appendix.A, illustrates the steps of building a reduction of an ordered tree.

Proposition 2.4.1 *The Tree reduction algorithm compresses an ordered tree in the worst case time of* $O(|T|^2)$.

The DAG contains practically all informations contained in the initial tree T that allow its reconstitution from its compression $R(T)$. Alg.2, Appendix.A, presents a recursive algorithm for the tree reconstitution.

Proposition 2.4.2 *The Ordered Tree reconstruction is performed in the worst case time of* $O(|R(T)| \times deg(R(T))$.

2.5 Properties of tree reduction

We notice that the tree and its reduction are equivalent and contain exactly the same data.

Proposition 2.5.1 *Any tree* T *can be recovered from its reduction* $R(T)$.

Proof *The proof is obvious from Alg.3 computing the reduction of a tree* T *and Alg.2 performing its reconstruction, see Appendix.A.*

Recall that, for any node I in $R(T)$, there exists a vertex x in T such that $I = c(x)$.

Proposition 2.5.2 *Any tree and its reduction have the same height.*

Proof *Let* T *be a tree and* $R(T)$ *its reduction. Since the proposed compression is performed in width, in such a way that for any vertex* x *in* T *their exists a unique node* I *in* $R(T)$ *with* $I = c(x)$ *and for which* $h(I) = h(x)$. *Then* $h(T) = h(R(T))$.

Many quantities that characterize a tree T can be directly computed on it's reduction $R(T)$, thus we define:

- $n(I) = |T[x]|$ the size of the tree rooted in x as

$$n(I) = 1 + \sum_{J \in child(I)} |l_{(I,J)}|.n(J) \tag{2.1}$$

Note that for a leaf of T, $child(I) = \phi$, and then $n(I) = 1$.
- The number of isomorphic subtrees in T rooted in x. for any I in $R(T)$:

$$m(I) = |\{x \in T | c(x) = I\}| \tag{2.2}$$

if $parent(I) = \phi$ then

$$m(I) = 1 \tag{2.3}$$

if $parent(I) \neq \phi$ then

$$m(I) = 1 + \sum_{k \in parent(I)} |l_{(K,I)}|.m(K) \tag{2.4}$$

- For a tree $T(V, E)$ and it's compression $R(T) = (V_r, E_r)$, the vertex compression factor is defined by

$$\rho_v = 1 - \frac{|V_r|}{|V|} \tag{2.5}$$

- Similarly, the edge compression factor is defined by:

$$\rho_e = 1 - \frac{|E_r|}{|E| - 1} \tag{2.6}$$

2.6 Validation

To validate the proposed solution and in the absence of a real dataset, we used the Treex library [4] to generate tree data of different sizes variant between 1000 and 10000 vertices, on which we applied our reduction algorithm. To ensure the consistency of the performance of our solution, the experiment was repeated 100 times for each tree size.

We measure and compare the performance of the semi-ordered tree compression using the following metrics:

- **Compression factors:** we have studied the evaluation of vertex and edges compression factor according to the size of the trees. Fig.2.4 and Table 2.1 show that the larger the tree size, we achieve a better compression ratio. On large trees, the patterns are more likely to have several occurrences, unlike small trees where we come across several patterns but which will not have the chance of having several occurrences, which explains the higher compression tau interested in larger trees. For edge compression ratio the figure shows that it is relatively low compared to the tau of compression of the nodes and that can be explained by the fact that even if we delete several occurrences of a pattern to keep only one on the latter there will still have all the links that were on the other occurrences. we also notice that the latter is more and more interesting on larger trees.

- **Compression time:** we have studied the time required for compression process also according to the size of trees. Fig.2.5 and Table 2.2 show that the calculation time remains reasonable even on trees of interesting size.

$size(T)$	Total number of experiments	$average(\rho_V)$	$average(\rho_E)$
1000	100	0,82	0,44
2000	100	0,83	0,47
3000	100	0,84	0,48
4000	100	0,85	0,49
5000	100	0,85	0,50
6000	100	0,85	0,51
7000	100	0,84	0,52
8000	100	0,85	0,52
9000	100	0,85	0,52
10000	100	0,85	0,53

Table 2.1: Evaluation of vertices and edges compression ratios according to the size of the trees.

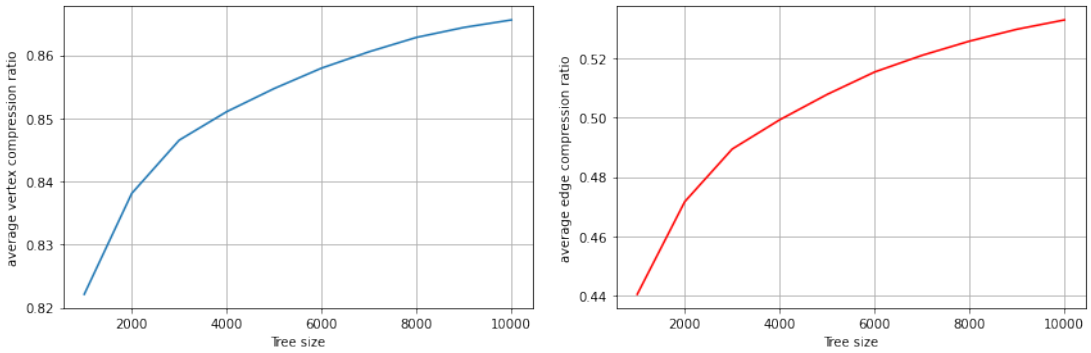


Figure 2.4: Evaluation of vertices and edges compression factors according to the size of the trees.

Tree size	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Time compression(s)	0,38	1,44	4,33	7,72	9,50	14,48	21,91	28,00	35,61	44,48

Table 2.2: Average time compression.

In a second part, we ignored the order between the siblings for applied to them a compression without loss of unordered tree and we compared the compression rates obtained in the case of unordered with the ordered case the results are exposed in Fig.2.6.

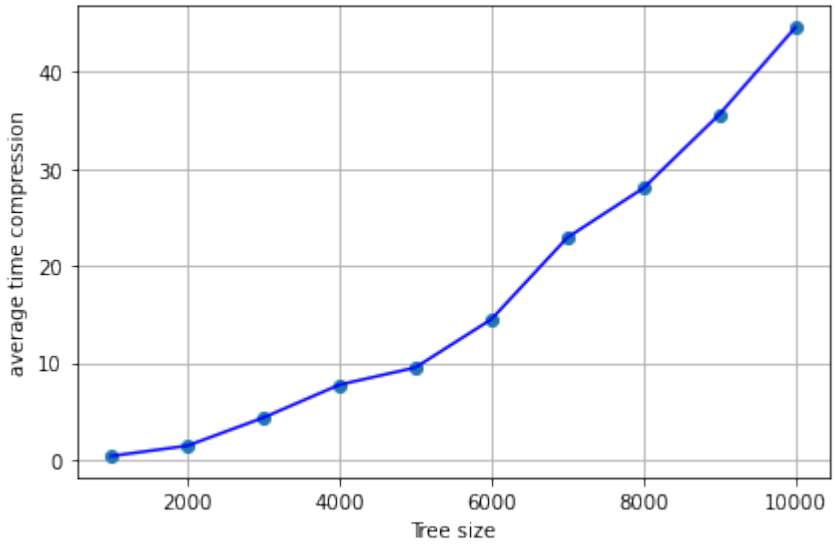


Figure 2.5: Average time compression.

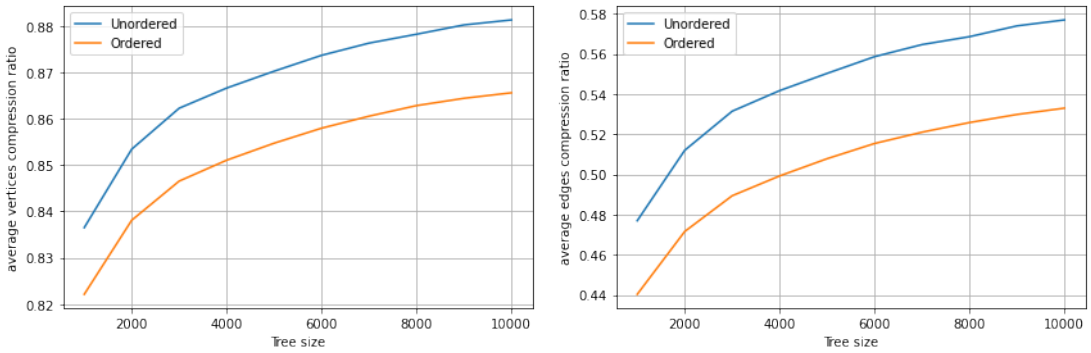


Figure 2.6: Comparison of compression rates between ordered and unordered trees.

We notice that the compression rate obtained on unordered trees are more interesting than on ordered trees and this comes down to the isomorphism of ordered trees which is more constraining

2.7 Conclusion

In this chapter we considered the problem of reduction of ordered trees where the trees are compressed in width using internal repetitions of their structures. Isomorphic subtrees are replaced by a unique class in the reduction. The efficiency of the compression algorithm is quantified by a compression factor reflecting the ratio between the size of the DAG and the size of the initial Tree. The new challenge is to adapt the solution for a new class of trees as semi-ordered trees and apply the solution for a real problem in particular the field of biology and plant biology.

Chapter 3

Lossless Compression of Semi-Ordered Trees

3.1 Introduction

Semi-ordered trees are a family of trees characterized by the fact that some siblings are equivalent where others are totally ordered.They present remarkable compression properties covering those of ordered and unordered trees. In this Chapter, we consider the problem of lossless semi-ordered tree compression. We present a solution that we published in the international Journal of Applied Research in bioinformatics [20].

To illustrate this approach, we then apply these notions to a particular class of semi-ordered trees which is the most studied branching structure particularly for a botanical motivation, namely axial trees. we evaluate the solution on artificial plant models represented by axial trees.

The proposed solution somehow unifies the compression of ordered and unordered trees. We assign to the different vertices of a semi-ordered tree a signature taking into account the configuration of its children (the order and the equivalence), this signature will make it possible to identify the isomorphic subtrees. We will use the equivalence relation induced by the isomorphism to build a quotient graph which will be annotated by labels on the edges to build a reduction of the input tree. We detail each step of the solution in the following. Note that the solution is purely structural compression and we only consider unlabeled trees in this chapter as well.

3.2 Separation of children

In this section, we will see how we go from a mixed set of children to clusters of equivalent siblings in order to prepare the computation of signatures

Definition 3.2.1 (Equivalent sibling) *Let $(T, S_{\sqsubseteq})$ be a semi-ordered tree and $\sqsubseteq_x$ a total semi-order relation on child(x). $\sqsubseteq_x$ induces an equivalence relation (i.e. reflexive, transitive and symmetric) on child(x) denoted by $=$ and defined by:*

$$\forall z, y \in child(x), z = y \Leftrightarrow z \sqsubseteq y \text{ and } y \sqsubseteq z.$$

The equivalence class of a vertex x is denoted by $[x]$ where:

$$[x] = \{y \in V | x = y\}$$

The elements belonging to the same equivalence class are said to be equivalent sibling, between which there is no order.

Definition 3.2.2 (Ordered sibling) *The total semi-order relation $\sqsubseteq_x$ induces a total order relation $<$ (i.e. reflexive, transitive and antisymmetric) on $child(x)$ such that:*

$$\forall z, y \in child(x), z < y \Leftrightarrow (z < m, \forall m \in [y] \text{ and } n < y, \forall n \in [z]).$$

To summarize, for $x_i, x_j \in child(x)$, the total semi-order relation $\sqsubseteq_x$ means that either x_i and x_j are equivalent or ordered sibling.

Definition 3.2.3 (The semi-ordered children partition) *The semi-ordered relation $\sqsubseteq_x$ induces a partitioning $\Gamma(x)$ of $child(x)$ such that:*

$$\Gamma(x) = \{P_a / \ P_a \text{ is the set of equivalent vertices of } child(x) \text{ of order } a\}.$$

Thus the semi-ordered relation $\sqsubseteq_x$ induces a partitioning $\Gamma(x)$ of $child(x)$ into classes, each class regroups equivalent sibling, while all classes form an ordered set. Note that, for any semi-ordered tree $(T, S_{\sqsubseteq})$ we have $S_{\sqsubseteq} = \{\Gamma(x), x \in T\}$, this can be shown through example of Fig.3.1.
For seek of clarity we will denote, in the sequel, a semi-ordered tree as $T = (V, E, S_{\sqsubseteq})$.

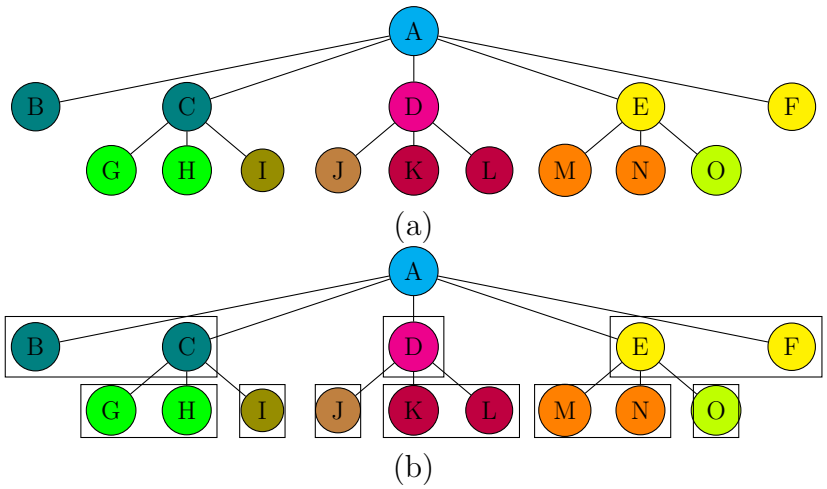


Figure 3.1: a) a semi-ordered tree in which equivalent (unordered) sibling are colored identically. (b) the ordered partition $\Gamma(A)$ induced by its total semi-order relations $\sqsubseteq_A$.

3.3 Signature of a vertex

As the semi-ordered trees present an order between some nodes while some others are equivalent, we defined the vertex signature for such trees, Definition.3.3.1, by combining vertices signatures for ordered and unordered trees.

Definition 3.3.1 (Signature of a vertex in a semi-ordered tree) *For each vertex x in a semi-ordered tree T the signature $\sigma(x)$ is defined as the corresponding semi-ordered set of all vertices signatures of $\Gamma(x)$, such that $\forall S_i, S_j \in \sigma(x)$ with $i < j$:*

- $\sigma(x_a) \in S_i$ *and* $\sigma(x_b) \in S_j \iff x_a < x_b$,
- $\forall \sigma(x_a), \sigma(x_b) \in S_i, x_a = x_b$,
- $\forall x_a \in S_i$ *such that* x_a *is a* T *leaf then* $\sigma(x_a) = \{\theta\}$.

In other terms, the signature $\sigma(x)$ of a vertex x in a semi-ordered tree is obtained from its semi-ordered child partition $\Gamma(x)$ by replacing each vertex $x_i \in child(x)$ by its signature $\sigma(x_i)$, and all leaves have automatically the same particular signature $\{\theta\}$. We keep the notation of the signatures of the sons by unique integers for the sake of simplicity.
For example, the vertices signatures corresponding to tree of Fig.3.1 are given as follow according to their bottom-up computing order:

- $\sigma(B) = \sigma(G) = \sigma(H) = \sigma(I) = \sigma(J) = \sigma(K) = \sigma(L) = \sigma(M) = \sigma(N) = \sigma(O) = \sigma(F) = \{\theta\}$ simplified 0,
- As $\Gamma(C) = \{\{G, H\}; \{I\}\}$, $\Gamma(D) = \{\{J\}; \{K, L\}\}$ and $\Gamma(E) = \{\{M, N\}; \{O\}\}$, replacing each vertex by its signature gives us:
  - $\sigma(C) = \sigma(E) = \{\{0, 0\}, \{0\}\}$ simplified 1,
  - $\sigma(E) = \{\{0\}, \{0, 0\}\}$ simplified 2,
- Finally, $\Gamma(A) = \{\{B, C\}; \{E\}; \{E, F\}\}$ thus $\sigma(A) = \{\{0, 1\}, \{2\}, \{1, 0\}\}$ simplified 3.

We denote by convention the set of all the signatures in a tree $T = (V, E, S_{\sqsubseteq})$ by $\sigma(T)$. Function $signatures(T)$ computes recursively $\sigma(T)$ in a bottom-up manner, as described in Alg.4, of Appendix.B.

Proposition 3.3.1 *Semi-ordered tree vertices signatures can be computed by function signatures(T) in worst case time $O(|T|^2)$.*

Proposition.3.3.2 gives the fundamental characteristic, relating the vertex signature concept to tree isomorphism, on which rests the semi-ordered tree reduction.

Proposition 3.3.2 $\forall x, y \in T, T[x] \equiv T[y] \iff \sigma(x) = \sigma(y)$.

Proof

(a) Let us consider two isomorphic subtrees $T[x]$ and $T[y]$, in other terms their is a bijection between $child(x)$ and $child(y)$. Then by definition $\sigma(x) = \sigma(y)$.

(b) Let us consider two vertices x and y in a tree T such that $\sigma(x) = \sigma(y)$. This implies that $|child(x)| = |child(y)|$, and $\forall x' \in child(x)$, $\forall y' \in child(y)$ where x', y' are either of the same order or belong to subsets of equivalent sibling of a same order in respectively $\sigma(x)$ and $\sigma(y)$, $T[x']$ and $T[y']$ are isomorphic. Thus $T[x]$ and $T[y]$ are isomorphic as well.

3.4 Tree reduction and reconstruction algorithms

By now, we have defined all information needed to state the compression method of a semi-ordered tree. From which, we can mention that the isomorphism relation between subtrees defines an equivalence relation between the tree vertices. We say that vertices x and y in V are equivalent if $T[x] \equiv T[y]$, and we note by extension $x \equiv y$, where $c(x)$ denotes the equivalence class of x.

We use this equivalence relation to build from a semi-ordered tree $T = (V, E, S_{\sqsubseteq})$ its quotient graph $G_Q = (V_Q, E_Q)$, such that:

- V_Q is a set of equivalence classes of $\equiv$ on V,
- E_Q is a set of pair of equivalence classes (X, Y) where $(X, Y) \in E_Q$ if and only if $\exists x \in X$ and $\exists y \in Y$ with $(x, y) \in E$.

For example, Fig.3.2 shows the quotient graph computed for tree of Fig.3.1.

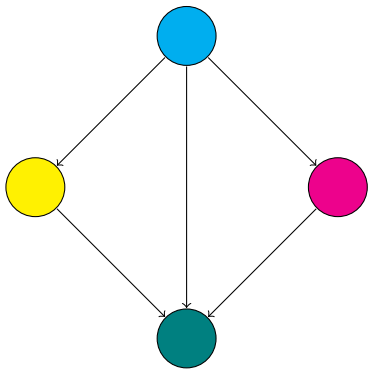


Figure 3.2: The quotient graph corresponding to tree T of Fig.3.1

The quotient graph G_Q obviously condenses the structure contained in the tree T. However, it gives no idea about the redundancies quantification of repetitive patterns nor about the position of any child among its other sibling. We thus consider how to augment the definition of G_Q so that the resulting condensed representation allows us the reach a lossless compression of the original tree. For this, we associate to each edge of E_Q a particular list as described in Definition.3.4.1.

Definition 3.4.1 *For each edge* $(c(x), c(y)) \in E_Q$ *we define the associated list:*

$$l_{(c(x),c(y))} = \{(a,b)/ for\, each\, P_b \in \Gamma(x),$$
$$a\, is\, the\, occurrences\, number\, of\, vertices\, of\, class\, c(y)\, in\, P_b\, and\, b\, defines\, the\, order$$
$$of\, subset\, P_b\, in\, among\, \Gamma(x)\}.$$

In other terms, we associate to each edge of E_Q connecting two classes I and J a list of of pair (a, b) where a is the number of occurrences of child from class J that seems in the order b among the other partitions. The resulting reduction graph $R(T)$ is thus the DAG augmented with labels on its edges, as seen in Definition.3.4.2. Fig.3.2.b depicts the reduction $R(T)$ of the semi-ordered tree T presented in Fig.3.1.

Definition 3.4.2 *Given a semi-ordered tree* $T = (V, E, S_{\sqsubseteq})$ *and its quotient graph* $G_Q = (V_Q, E_Q)$ *regarding the isomorphism relation on* T*, we define its reduction graph* $R(T) = (node_list, edge_list)$ *as a DAG in which* $node_list = V_Q$ *and* $edge_list = \{(e, l_e), \forall e \in E_Q\}$.

Fig.3.3 shows the reduction corresponding to the tree of Fig.3.1.

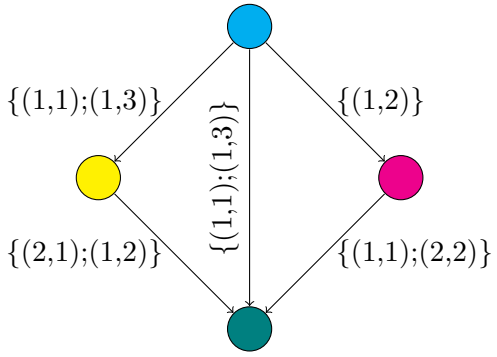


Figure 3.3: The reduction corresponding to tree T of Fig.3.1

Finally, this can be summarized through Alg.5, in Appendix.B, compressing a semi-ordered tree in the worst case time of $O(|T|^2)$, and Alg.6, in Appendix.B, decompressing a reduction $R(T)$ of a semi-ordered tree T in the worst case time of $O(|T| \times deg(T))$.

3.4.1 *Properties of tree reduction*

We notice that the tree and its reduction are equivalent and contain exactly the same data.

Proposition 3.4.1 *Any tree* T *can be recovered from its reduction* $R(T)$.

Proof *The proof is obvious from Alg.5 computing the reduction of a tree* T *and Alg.6 performing its reconstruction, see Appendix.B.*

Recall that, for any node I in $R(T)$, there exists a vertex x in T such that $I = c(x)$.

Proposition 3.4.2 *Any tree and its reduction have the same height.*

Proof *Let T be a tree and $R(T)$ its reduction. Since the proposed compression is performed in width, in such a way that for any vertex x in T their exists a unique node I in $R(T)$ with $I = c(x)$ and for which $h(I) = h(x)$. Then $h(T) = h(R(T))$.*

Many parameters that characterize a tree T can be directly computed on it's reduction $R(T)$, thus we define

- $n(I) = |T[x]|$ the size of the tree rooted in x as

$$\text{if } child(I) = \phi \text{ then } n(I) = 1 \text{ else } n(I) = 1 + \sum_{J \in child(I)} \left(n(J) \times \sum_{(a_i,b_i) \in l_{(I,J)}} a_i \right) \tag{3.1}$$

- The number of isomorphic subtrees in T rooted in x, corresponding to $m(I) = |\{x \in T | c(x) = I\}|$ for any I in $R(T)$, as

$$\text{if } parent(I) = \phi \text{ then } m(I) = 1 \text{ else } m(I) = \sum_{k \in parent(I)} \left(m(k) \times \sum_{(a_i,b_i) \in l_{(I,J)}} a_i \right) \tag{3.2}$$

- For a semi-ordered tree $T(V, E, S_{\sqsubseteq})$ and it's compression $R(T) = (V_r, E_r)$, the vertex compression factor is defined by

$$\rho_v = 1 - \frac{|V_r|}{|V|} \tag{3.3}$$

- Similarly, the edge compression factor is defined by:

$$\rho_e = 1 - \frac{|E_r|}{|E| - 1} \tag{3.4}$$

3.5 Experiments

In order to prove the efficiency of the proposed semi-ordered tree compression we present in this section the compression of simulated plants. The aim is to prove that the solution can be exploited in real problems on real data and that even large semi-ordered trees can be compressed in a reasonable time and with high compression factors, using an environment with standard computing resources.

We first present the simulated plants that we use and we describe how we will generate them before showing how to model these plants in a particular family of semi-ordered trees and we end this section with a presentation of the results obtained.

Botanical trees in nature shows a high similarity in their organs, structures and components. Repetitive patterns are readily perceptible in the growth processes and structure of many living organisms especially plants. This repetition is noticed in a lower level when the same structure is repeated periodically along an axis, such as leaves along a stem and can go further to a repetition of a whole structure in several stages.

The important role of the architecture in the understanding of plants generates a need for investigation tools. The study of tree structure has been the subject of several studies and many problematic has been treated in this context, such us: the visualization of plant architecture in 3D [30], modelization of the development of plants structure [15], comparison and calculation of similarity between plants architecture [27] and the plant structure compression who took an important place in this context going from the hypothesis that a good representation of these data ensures its efficient processing and taking advantage of the frequent redundancy in plant architecture.

Plant architecture can be modeled as tree-graphs [14], where the set of vertices V describes the plant components and the set of edges E represents the adjacency of these components. In general case, unordered trees are used to represent plants topology, as depicted in example of Fig.3.4.

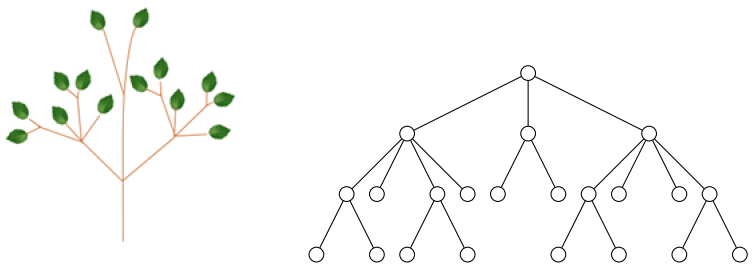


Figure 3.4: (a) Visualization of plant architecture using a schematic plant representation. (b)Visualization using the equivalent unordered tree T.

Furthermore, in branching plants all components are connected around an axis, the vertices connected directly to the axis are considered as carriers of information and therefore they are more important in some way than their siblings. The vertices which are not connected to the axis are all equivalent[13]. In this case, the plant topology is represented by an axial tree, for example tree of Fig.3.5.

3.5.1 *Creation of the plant database:*

A database of plants was created using a L-system based modeling system [6] in a similar manner to that described in [13, 16] and that we briefly recall in the next paragraph.

The apical meristem of the main axis progresses through a sequence of ordered morphological differentiation states. The initial state s_0 correspond to the germi-

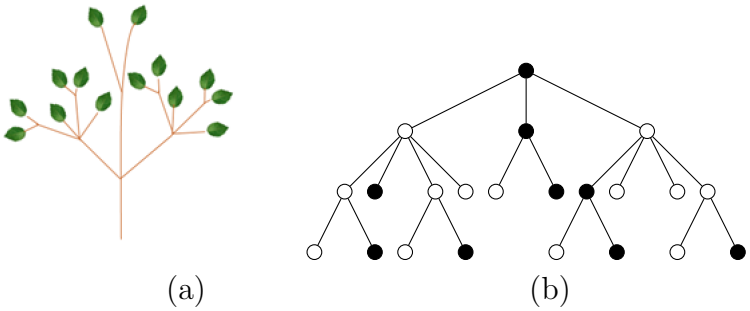


Figure 3.5: (a) Visualization of plant architecture using a schematic plant representation. (b)Visualization using the equivalent unordered tree T.

nation and the final state s_f correspond to the flowering. Each state s describe the color and the geometric characteristics of the produced branch segment at this state before passing to another state s' such that $s \leq s'$ until the apex arrive to the final state and produce a terminal organ (a flower). In addition to apical productions, at each step the apical meristem of any axis can produce lateral meristem. The initial state of the created lateral meristem must be equal to or greater than that of the apical meristem that created it.

We use differentiation graphs to visualize the above process in illustrated example of Fig 3.6 where colored circles represent differentiation states, solid arrows represent possible state transition of the apical meristem during the apical growth and the dashed arrows represent the transition of the lateral meristem during the lateral growth. The meristem stays in the same state for the number of steps indicated by the label associated with a loop, then moves to the next state.

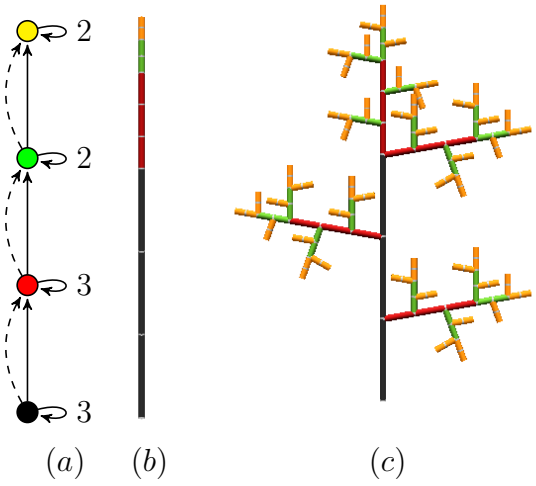


Figure 3.6: a) Differentiation graph used for the definition of a theoretical plant b) the resulting apical axes where each component has the same color of the differentiation graph state in which this component has been produced. (c)The whole resulted branching structure

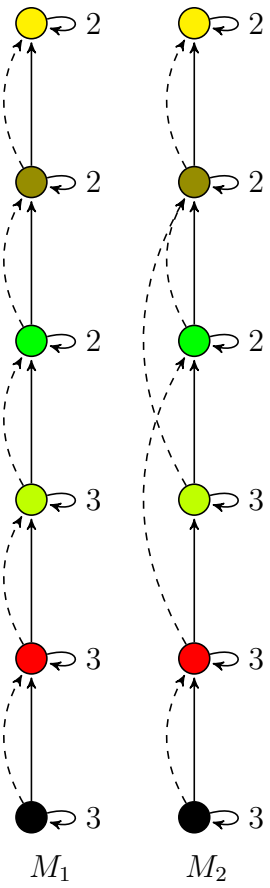


Figure 3.7: The differentiation graphs of models M_1, M_2

We used two differentiation graphs M_1, M_2 (see Fig3.7) to generate two plants families (of Fig 3.8). Each graph has six states and the two graphs differ in the lateral meristem transitions. A family F_i consists of a deterministic model M_i and five derived models labeled $M_{i-0.9}, M_{i-0.8}, M_{i-0.7}, M_{i-0.6}, M_{i-0.5}$ by randomizing a probability p (p is indicated in the label of the derived model M_{i-p}) that a lateral apex gave rise to a branch; otherwise the branch was aborted. For more diversity, 10 samples M_{i-p} were generated for each probability p.

3.5.2 *Analysis and results:*

We used the L-PY simulator [6] to generate the theoretical plants and the TREEX library [4] to model each plant by an axial tree.

All algorithms were implemented in Python and compiled with python 3.7.3 compiler. The experiments were conducted on a commodity PC with an Intel Core i5 CPU and 4 GB of DDR3 RAM, running Ubuntu Linux 18.04.

We computed the reduction of each tree in the database, then we used the compression ratio to evaluate each compression according to the probability p that a lateral branch grows (Tab 3.1).

We observe a high values of compression factors for trees T_1 and T_2 which mean

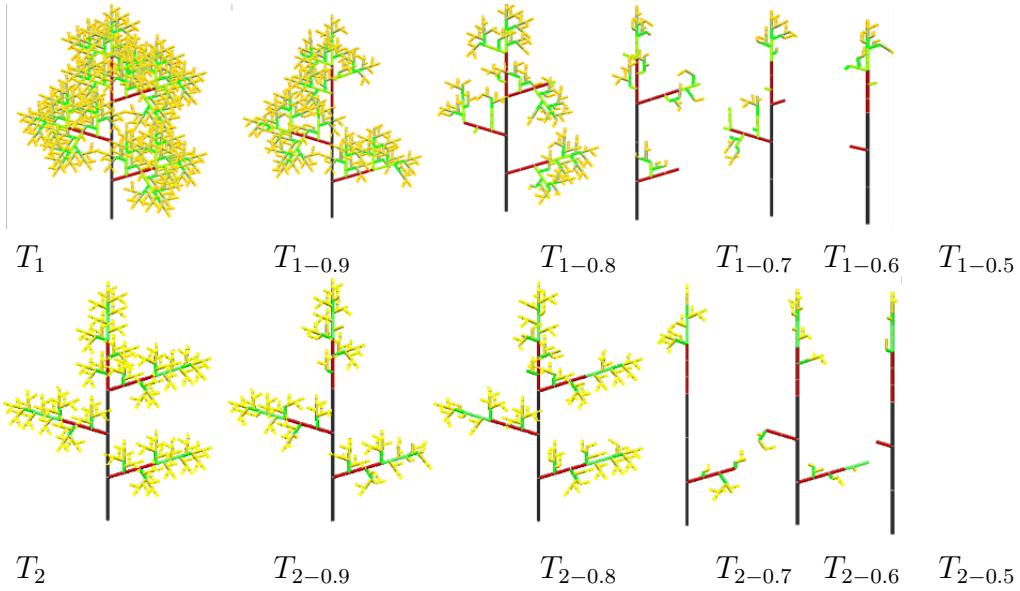


Figure 3.8: Theoretical plants: the F_1 family (first row) and the F_2 family (second row). Both of T_1 and T_2 present repetitions of the lateral branches hence a high degree of compressibility. However, for the others trees the random removal in lateral structures introduces variation that is expected to reduce their degree of compressibility.

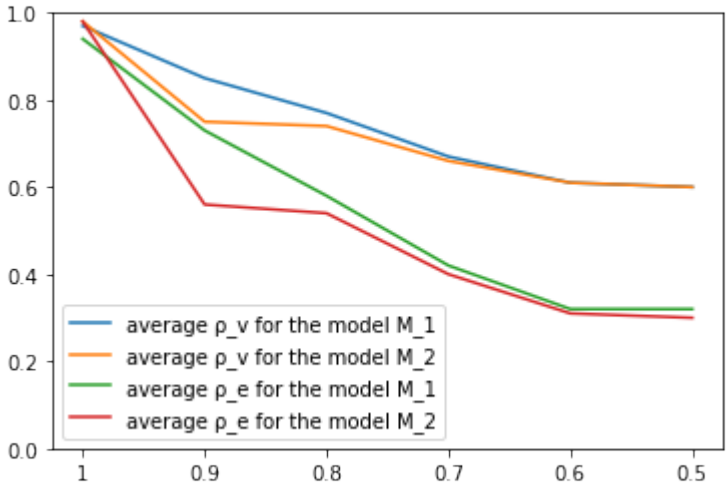


Figure 3.9: Average vertices and edges compression factors of tree families generated from the models M_1, M_2 according to the probability of lateral axis growing

that they present a high compressibility. In contrast, as it was expected the random remove of lateral axis reduce the compressibility of the trees but the results are still performing, especially in terms of vertex compression factor. The algorithm manages to capture similarities in the plants structures and uses them to build a more fluid representation while maintaining the axial aspect linked to these data.

	M_1				M_2			
p	$\|V\|$	$\|E\|$	ρ_v	ρ_e	$\|V\|$	$\|E\|$	ρ_v	ρ_e
1	1644	1645	0.97	0.94	1726	1727	0.99	0.98
0.9	808	809	0.85	0.73	183	184	0.75	0.56
0.8	385	386	0.77	0.58	170	171	0.74	0.54
0.7	157	158	0.67	0.42	105	106	0.66	0.40
0.6	92	93	0.61	0.32	71	72	0.61	0.31
0.5	75	76	0.61	0.32	60	61	0.60	0.30

Table 3.1: Average tree statistics of tree families generated from the models M_1, M_2. Representation of trees sizes variances and their compression ratio according to the probability of lateral axis growing

3.6 Conclusion

In this Chapter, we introduced the semi-ordered tree compression into directed acyclic graphs. Semi-ordered trees are nonlinear data structures that present certain structural properties such that some sibling elements are equivalent while some others are totally ordered.

Semi-ordered tree compression aims to remove internal structural redundancy by merging isomorphic subtrees in a unique class. We gave an efficient method that compress a semi-ordered tree T in $O(|T|^2)$ time complexity and we also presented a reconstruction algorithm that allows to recover the initial tree from its reduction in $O(|T| \times deg(T))$ time complexity.

We used the compression factor which reflect the ratio between the size of the reduction and the size of the initial Tree to prove that the algorithm achieve good results. Experiments on randomly generated trees have also proved the effectiveness of the algorithm even in front of large trees.

We illustrated the method on plant architecture and showed the effectiveness of the solution for axial trees. We are persuaded that the simplicity, the effectiveness, the height speed and the ability to handle large trees encourage the exploitation of this solution to solve several problems. Howbeit, significant optimizations are expected to get better result for edges compression factor. Another challenge we are aiming for is to use the compression to optimize algorithms that manipulate this class of trees.

Chapter 4

Lossy compression of labeled trees: cut-based approximation approach

4.1 Introduction

The second part of this thesis was carried out in collaboration with the Mosaic team of the RDP laboratory as part of the PROFAS B+ exchange program. The goal was to propose solutions to the lossy compression problem of unordered labeled trees. We present in this chapter our first contribution in this context.

The noisy tree structure with non-exact redundancies but also the limitation of the lossless compression model motivated our attempt to generalize this model to build a lossy compression model able to handle this kind of tree and promising better performance. In this chapter we present our first tree lossy compression method. A method that exploits the approximate self-redundancy of a tree structure but also additional information on node labels to compress it on different degrees while respecting a compromise between the compression rate and its relevance. This new compression method makes it possible to increase the efficiency of models, simulations and analyzes of particularly redundant structures by using the compressed representation instead of the initial representation.

4.2 Problem formalization

In the context of labeled trees, subtrees are not perfectly isomorphic (especially when the set of label values is continuous), and therefore, when we build the quotient graph $|V_Q| \approx |V|$ we notice in this case that these trees do not compress sufficiently. The principle of lossy compression is to group into equivalence class subtrees which are not exactly isomorphic but which resemble to a certain degree. This induces a passage through an approximation phase of the initial tree T by a tree T^* that presents better compression properties. However, for the sake of quality results, we add two constraints. The first on the size of the compression: we want to control the size of the compressed version by imposing a compression rate. The second required

constraint is to allow compression with a controlled loss. For this, we impose the fact that T and T^* are as close as possible. The main solution of the problem is then to find from a tree T, its approximation T^*, such that

$$D(T, T^*) \leq \epsilon \text{ and } 1 - \frac{size(\mathcal{R}(T^*))}{size(T)} \geq \tau.$$

In order to give a more formal definition, we need to introduce some notions. The ϵ-quasi-isomorphism on trees given in Def.3.1 is a less constrained notion than the isomorphism introduced in Def.2.2.1, in this case the distance between the two trees in question is not null but less than a positive threshold.

Definition 4.2.1 (ϵ-quasi-isomorphism) *Let T_1 and T_2 be two trees and $\epsilon \in \mathbb{R}^+$, T_1 and T_2 are ϵ-quasi-isomorphic if and only if $D(T_1, T_2) \leq \epsilon$.*

Remark *The tree ϵ-quasi-isomorphism presents some properties:*

- *for $\epsilon = 0$ the ϵ-quasi-isomorphism amounts to an isomorphism,*
- *The ϵ-quasi-isomorphism does not define an equivalence relation, indeed it is clearly not transitive.*

Tree edit-distance can be used to identify the isomorphism between subtrees [40, A Constrained Edit Distance Metric Between Unordered Trees] [**ferraro2000distance**]. A null distance between two subtrees (or trees) denote an isomorphism between the two structures where a small distance denote an ϵ-quasi-isomorphism. The use of bottom-up recursion and dynamic programming allowed these algorithms to remain in polynomial complexity. In this paper, we exploit this distance to extend the lossless DAG reduction scheme of labeled trees. The idea is to start from a distance computation between the tree and itself which will obviously be null but which computes at the same time the distance between all the subtrees of the input tree, then we gather ϵ-quasi-isomorphic subtrees in classes that replace the equivalence classes in the construction of the quotient graph.

4.3 Global clustering strategy

As the objective is to form groups of ϵ-quasi-isomorphic subtrees, we start from a *global clustering* of the subtrees set of the input tree, see example of Fig.4.1.

Using a clustering algorithm and the edit distance we partition the set of all T subtrees to groups of ϵ-quasi-isomorphic subtrees, then we build a new tree T^* such that the subtrees belonging to the same cluster in T are approached by a single model subtree, then the lossless DAG reduction scheme is applied on the resulting tree.

The *global clustering strategy* allows to form groups of trees which are not perfectly isomorphic and nevertheless present a high degree of similarity, however it does not consider the structural relationship between these subtrees. Indeed, two structures in hierarchical relation can be similar to some degree but making them in the same cluster risks disrupting the strategy. Fig.4.2 illustrates some difficulties due to hierarchical relations in trees.

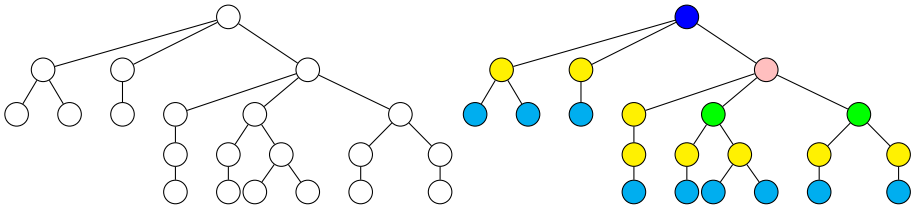


Figure 4.1: An unordered tree T (left). A global clustering of T subtrees generating 5 clusters (right) where we use the same color for roots of subtrees belonging to the same cluster.

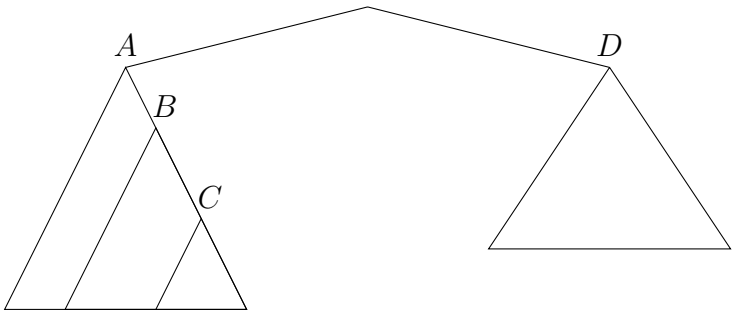


Figure 4.2: A tree in which we consider that subtrees (A, D) belong to the same cluster while (B, C) belong to an other one. The approximation of C by B does not remove the occurrence of C in the approximated tree since the latter is a subtree of B, it will also create a new class of subtrees and as a result the DAG after approximation is greater than the initial DAG. Similarly, if we approximate B by C this does not guarantee that the result on A is always similar to D.

4.4 Tree cut strategy

To avoid the problem of hierarchical relations between subtrees, whether in the same cluster or between sub-trees belonging to different clusters, the solution is then to select from each cluster a subset of sub-trees that are disjoint between them and disjoint from those belonging to the other clusters. This amounts to define the notion of a *Tree cut*, as formally described in Def.4.4.1 and visualised through example of Fig.4.3. Alg.7, given in Appendix.C, shows how to generate a cut from a global clustering of a given tree.

Definition 4.4.1 (Tree cut) *For a tree T, a cut $\mathcal{C}(T) = (\mathcal{T}, \kappa)$ is a pair composed of a rooted tree $\mathcal{T}$ and a disjointed forest κ, such that:*

- $root(T) = root(\mathcal{T})$,
- *A leaf l_i from $leave(\mathcal{T})$ can be a terminal leaf if $l_i \in leave(T)$ or an anchor leaf if $l_i \notin leave(T)$ meaning that $T[l_i] \in \kappa$.*

In other terms, a horizontal cut crosses the tree at the level of the edges (u, v) where v is an anchor leaf in $\mathcal{T}$ and $T[v] \in \kappa$.

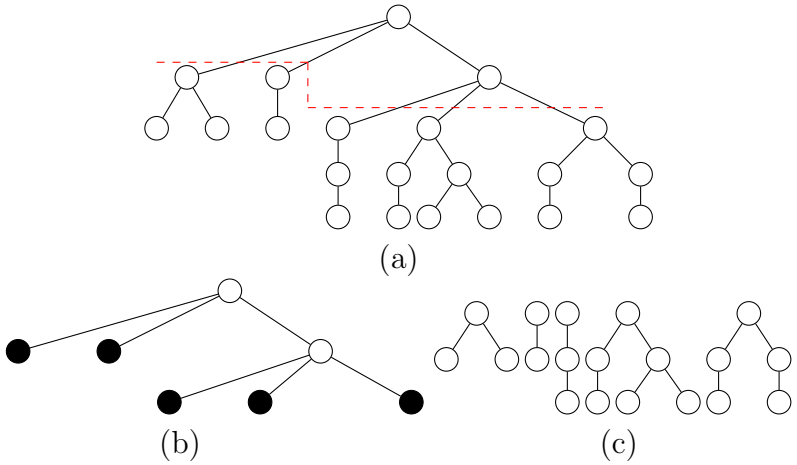


Figure 4.3: (a) A horizontal cut crosses the tree T and separate it into: (b) a rooted tree $\mathcal{T}$ at $root(T)$ with non-terminal leaves (anchors) colored in black and (c) a disjoint forest κ.

4.4.1 *Local Clustering*

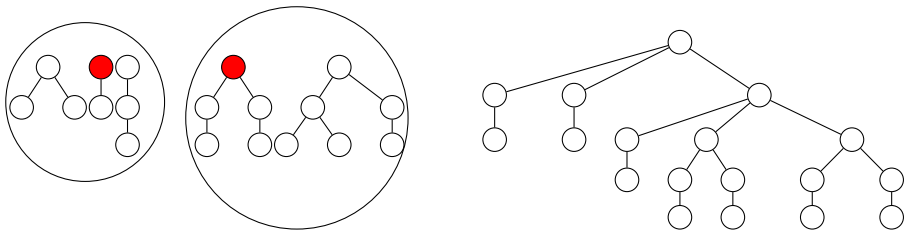
Applying a cut $\mathcal{C}(T)$ on a tree T aims to isolate a forest κ of disjointed subtrees located under the cut. To continue the spirit of the solution proposed in section 4.3, a *local clustering* is then applied on κ to form clusters of disjointed and ϵ-quasi-isomorphic subtrees. For this task, we use the k-medoids algorithm [28, Proposed K-medoids algorithm]. The k-medoids algorithm allows to split input data into k clusters, the value of k is computed in such a way to minimize the intra-cluster variation and maximize the variation between the clusters. In this context, the chosen k-medoids algorithm [28] have a complexity of $O(n \times k)$ where n is the number of data to cluster and k the number of clusters.

After the κ subtrees local clustering, the next challenge is to find a representative to each cluster. The computation of clusters representatives is also a problem that falls within the compression-loss of information compromise. Indeed, we seek that each representative is both as central as possible among the other elements of a same cluster and that the forest of representatives also compress as much as possible. We expose, in Appendix.C, the method we adopted to calculate the representative of each local cluster, and this process is formally detailed in Alg.9 of Appendix.C.

4.4.2 *Cut approximation*

Finally, we can build an approximation T^* of the initial tree T from the clustered forest κ and the representatives of the clusters. Subtrees belonging to the same cluster are both ϵ-quasi-isomorphic and disjoint trees. Subsequently, we replace the subtrees belonging to the same cluster by the representative of this cluster to get the approximated tree, as shown in Fig.4.4 representing the approximation built from the tree cut of example of Fig.4.3. This process is described in Alg.10 in AppendixC, where the evaluation of each cut depends on the Zhang edit distance computation algorithm [40], hence the complexity of the evaluation of N cuts is in

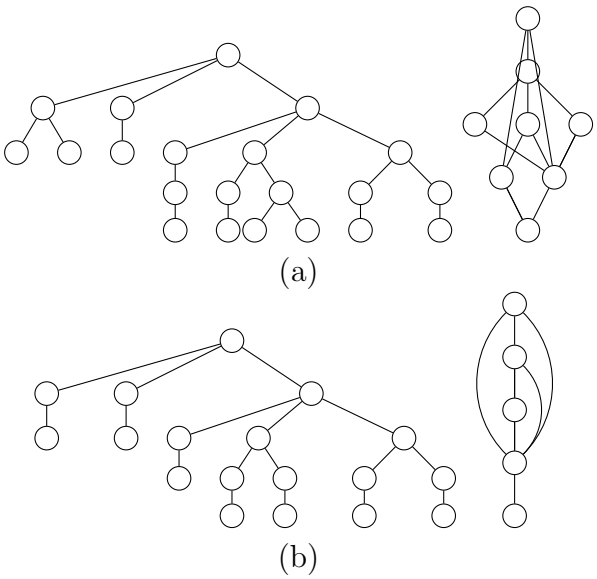
$O(N \times |T|^2 \times deg(T) \times log(deg(T)))$.



(a) (b)

Figure 4.4: (a) Local clustering of the forest κ of Fig.4.3.c, roots of clusters representatives are colored in red. (b) Approximation of subtrees belonging to the same cluster by the cluster representative in the initial tree of Fig.4.3.a.

We apply the tree to DAG compression scheme on the approximated tree T^* to build a lossy compression of the initial tree T, as seen in example of Fig.4.5 representing the compression of the tree T of Fig.4.3.a and its approximation of Fig.4.4.b.



(a)

(b)

Figure 4.5: (a) An unordered tree T (left), its DAG reduction (right), (b) An approximation T^* of T (left), the DAG reduction of T^* and the lossy compression of T (right).

The resulting lossy compression is evaluated regarding the two following criteria:

The loss of information measured by the Zhang edit distance between the approximated tree and the initial tree:

$$\sigma = \frac{\Delta(T,T^*)}{\Delta(T,\theta)}$$

The compression ratio measured by a ratio of the size of the DAG to the size of the initial tree:

$$\tau = 1 - \frac{size(D^*)}{size(T)}$$

4.4.3 *Tree approximation from an optimal cut*

Now that we have seen how to build a compression as well as a lossy compression from a cut on a tree and how to assign a score to it. Note that in reality, on a given global clustering of a tree we can generate several cuts, depending on the variation of the traversal order on the global clusters, the number of global clusters and on the clustering way of the subtrees set of the initial tree. Alg.8, in Appendix.C, shows how to generate a set of cuts from several global clustering of an initial tree. The proposed algorithm generates a set of N cuts obtained from several global clustering, where each global clustering induces n cuts. This gives a complexity of $O(\frac{N^2 \times |T|^2}{n})$ for Alg.8 of Appendix.C.

Each generated cut associates to the initial tree a particular approximation. A good choice of the cut is motivated by the fact that the compression scheme will be applied to the resulting approximation. The intuitive idea is to evaluate all generated cuts and to choose the optimal cut against a particular criterion. Except that the number of all cuts explodes rapidly depending on the number and the size of the tree global clusters, thus it becomes difficult to evaluate all the cuts. This leads us to choose the strategy based on generating cuts under constraints.

4.4.4 *Constraints on cuts*

In order to control the loss of information, we do not consider cuts which do not preserve a minimum of the the initial tree structure. For that, one defines constraints on the initial tree which specifies if a subtree can be part of a cut (eligible for a future approximation by another subtree) or not (to be preserved in the approximation). Note that, constraints are often linked to the nature of the data, as we can see in Section.4.5.2.

4.4.5 *Optimal cut selection*

The goal now is to choose the cut that best meets a particular criterion among all the generated cuts. The selection criterion can be based on the loss of information (minimization) or on the compression rate (maximization).

Consider Γ the set of cuts for which we associate to each cut $\Gamma_i \in \Gamma$ a pair $(\tau_{\Gamma_i}, \sigma_{\Gamma_i})$, where τ_{Γ_i} is the compression rate of the lossy compression built from Γ_i, and σ_{Γ_i} is the information loss resulting from this compression. Thus:

- The optimal cut maximizing a compression rate criterion τ_{min} is the cut c with a minimum of information loss and with a compression rate greater than or equal τ_{min}

$$c = \underset{\Gamma_r = \{\Gamma_i \in \Gamma | \tau_{\Gamma_i} \geq \tau_{min}\}}{argmin} (\sigma_{\Gamma_r})$$

- The optimal cut minimizing a compression rate criterion σ_{max} is the cut c with a maximum compression rate and a loss of information less or equal than σ_{max}

$$c = \underset{\Gamma_r=\{\Gamma_i \in \Gamma | \sigma_{\Gamma_i} \leq \sigma_{max}\}}{argmax} (\tau_{\Gamma_r})$$

This process is formalized in Alg.11 of Appendix.C where the step of optimal cut selection of the initial tree T takes a worst time of $O(|T|)$.

4.5 Application on virtual plants

The aim of this section is to illustrate the contribution of the proposed lossy compression model in the compression of a plant structure. We will start with a presentation of the models used for the generation of plants used for the tests, then we will see the constraint used during the generation of cuts on these plants and finally we will evaluate the contribution of this lossy compression model on the compression of the plants. We evaluated this model regarding the loss of information compared with the initial structure, the compression ratio of the resulting approximation and on the contribution in the calculation of the edit distance between two plants.

4.5.1 *Plant architecture database*

With the MOSAIC team of the RDP laboratory, we worked on new models of simulated plants by evolving the models presented in [13, 16] and that we used in 3.5.1. We transformed the deterministic passage between the states to a stochastic passage by adding probabilities of passage from one state s to several states on the apical meristem and the lateral meristem

Each model is described by two matrices A and L of size $(n \times n)$ and two vectors of size n to represent the minimum and maximum duration of each state where n is the number of differentiation states that the apex can pass through . On matrix A the value $A[s][s']$ represents the probability of passing from state s to state s' on the apical meristem and on matrix L the value $L[s][s']$ represents the probability of passing from state s to state s' on the lateral meristem.

To complete the model and give a better description to the plants, we have added functions that associate to each production (apical or lateral) attributes that describe the geometry of the branch produced: the diameter, the length, the angle of insertion and the diversion angle.

We generated 30 plants with 3 models described in the Appendix.D for test. Fig 4.6 shows examples of these plants.

4.5.2 *Constraint on plant cuts*

As we have seen in Section.4.4.4, pushing the compression to the maximum generates an uncontrolled loss of information. In plants compression, this manifests in loss of plant architecture, see example of Fig.4.7.

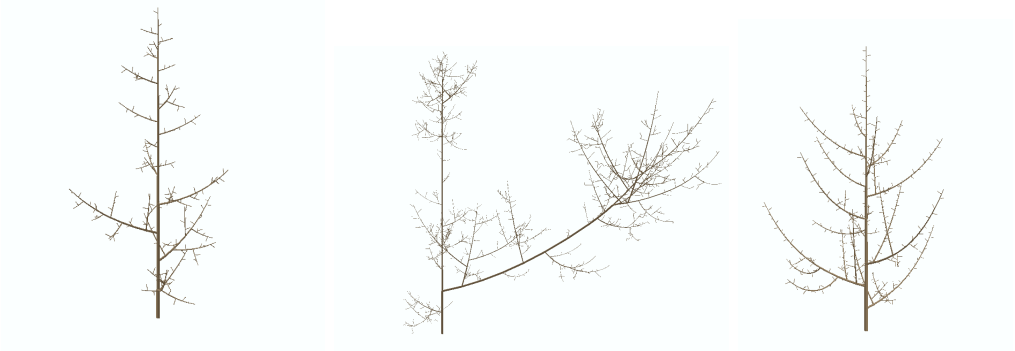


Figure 4.6: Theoretical plants generate with models M_1, M_2 and M_3 in that order.

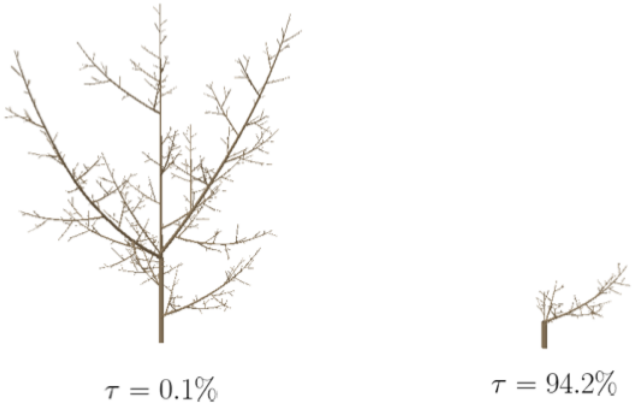


Figure 4.7: An artificial plant with a compression rate of 0.1% (left), and its approximation with a compression rate of 94.2% (right).

To control the loss of information we introduce a constraints on the cut related to the plants architecture. The aim is to determine a minimal part of the plant that is considered elementary and that must be preserved during the approximation and a dynamic part that can changes during the approximation. For this, for each element e of the plant we note by λ_e the ratio $\frac{d_e}{d_r}$ where d_e is the diameter of an element e and d_r is the diameter of the root of the plant. An element is considered as elementary if λ_e exceeds a certain threshold, otherwise the element e is small enough in the plant that it can be changed and therefore this element is considered as a dynamic part of the plant. The cuts generated will only be performed on the dynamic parts of the plant. Fig.4.8 shows a plant where we have differentiated between the elementary part and the dynamic part (the threshold used is 0.3).

We varied the threshold on λ_e to change each time the elementary part of the plant before each experiment. Fig.4.9 shows an example of a plant generated with the model M_1 and its different elementary parts corresponding to different thresholds.

As we can see, the greater the threshold on λ_e, the part to be preserved in the approximation is smaller and at the same time the dynamic part is greater which allows to generate a greater number of cuts.

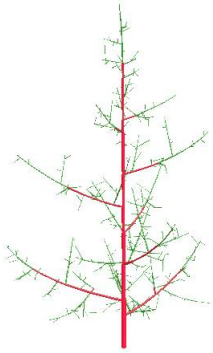


Figure 4.8: In red the elementary part of the plant to be preserved during the approximation and in green the dynamic part on which the cuts are generated.

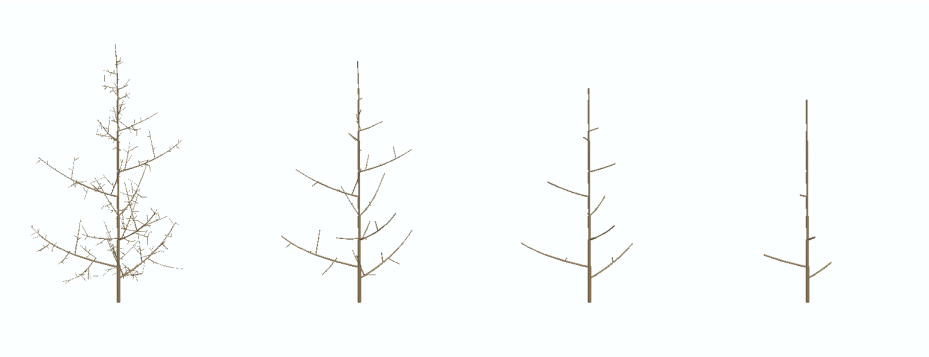


Figure 4.9: From left to right a plant generated with the M_1 model followed by the elementary parts corresponding to a threshold of 0.2, 0.3, and 0.4 on λ_e.

.

4.5.3 *Analysis and performance*

Detailed geometric representations of plant architecture, plant modeling [25] and the structural redundancies in plants which only become to a certain degree when we consider this geometric information motivated our choice to test the presented method on simulated plants.

4.5.3.1 Maximum compression with minimum loss of information

At this level, we seek to find a compression at a rate τ_{min} with minimum loss of information. For that, we sort down all the approximations generated by the compression rate then we exclude all approximations with a compression rate lower than τ_{min}. Next from the remaining set, we return the compression with the minimum loss of information, as described in Alg.11 in the AppendixC.

Fig.4.10 shows the different approximations of a plant generated with the model M_1 for different value of τ_{min} and for different threshold of λ_e.

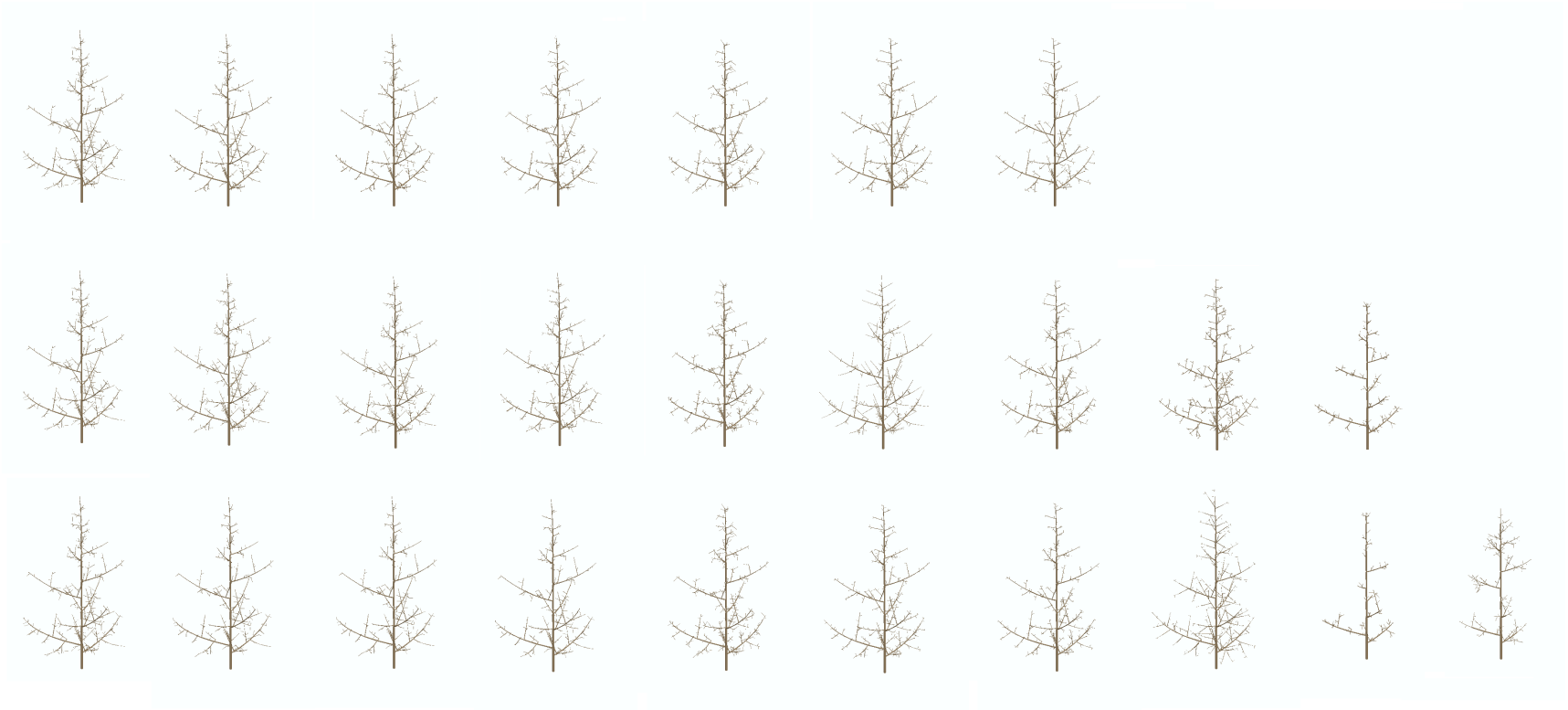


Figure 4.10: The lines corresponding to experiments with thresholds of 0.2, 0.3 and 0.4 on λ_e, in each line from left to right the initial tree followed by the approximations corresponding to values of τ_{min} among [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], the absence of the plant for a specific value of τ_{min} means that no approximation with a compression rate greater than or equal to τ_{min} was found.

We notice that the visual aspect of the loss of information is very weak against the important compression rates that we can reach, the approximations are made on the details while the elementary parts of the plants are preserved.

The constraint on λ_e determines the possible approximation space, the latter is quickly exhausted when threshold of λ_e is small enough and not offering the possibility of reaching high compression ratios (Fig.4.10 line 1 maximum rate 60%). In this case, the solution to push the compression to a greater τ_{min} is to combine two (or more) cuts while releasing the constraint on λ_e. Fig.4.11 illustrates the case of a sequence of two cuts for the same plant in Fig.4.10 for thresholds of 0.2 and 0.3 on λ_e relaxed respectively to 0.3 and 0.4 for the second cut.



Figure 4.11: On the left the initial plant followed by: on the first line the result of the first cut with a threshold of 0.2 on λ_e then that of the second after threshold relaxation to 0.3, on the second line the result of the first cut with a threshold of 0.2 then that of the second after threshold relaxation to 0.4.

As we can see, boosting a second cut on the result of a first one allows to push the compression even more but this is quickly paid for by a significant jump in the loss of information.

On the same plants of the first experiment on model M1 and with the different thresholds on λ_e (0.2, 0.3 and 0.4) we generated the approximation corresponding to several values of τ_{min} among [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] and we measured each time the loss of information which goes with the results. Fig.4.12 represents the obtained loss of information regarding the different values of τ_{min}.

Experiments show that, we can gain up to 60% of compression rate for a very low loss of information which does not exceed 20% of the information contained in the initial plant. For greater compression rate (more than 70%) we notify a significant

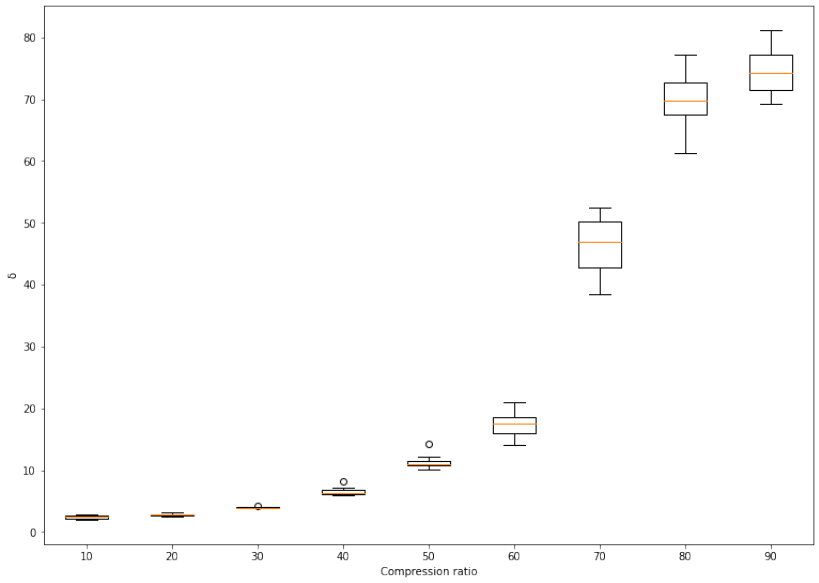


Figure 4.12: The loss of information due to the approximation against τ_{min}.

jump in the loss of information due to high cuts or a combination of several cuts.

4.5.3.2 Edit distance

This part illustrates the contribution of the lossy compression model proposed in the optimization of the edit distance computation. The idea is to calculate the distance on the compression DAGs directly instead of doing it on the trees.

The distance calculation on the DAGs makes it possible to replace all the distances calculations between the sub-trees x_i belonging to the equivalence class X and the sub-trees y_j belonging to the equivalence class Y by a single distance calculation between an instance of X and an instance of Y then reuse the result for other calculations. Formally :

$\forall x_1, x_2 \in X \times X$ and $\forall y_1, y_2 \in Y \times Y : D(x_1, y_1) = D(x_2, y_2)$ where X and Y are equivalences classes of the isomorphic relation.

Therefore, the less the size of the compression it is, the more there will be simplified calculations. We use two measures to evaluate the model in this context: the gain in computation time that we seek to maximize and the error on the distance that we seek to minimize.

Let us consider two input trees T_1 and T_2 and let $T_1(\tau_i)$ be the approximation corresponding to the compression rate τ_i, $D(T_1, T_2)$ the edit distance between T_1 and T_2, $D(T_1(\tau_i), T_2)$ the edit distance between $T_1(\tau_i)$ and T_2, t_i the time needed to calculate $D(T_1, T_2)$ and t_f the time needed to calculate $D(T_1(\tau_i), T_2)$ then:

- The error on the distance $\delta = \frac{|D(T_1,T_2) - D(T_1(\tau_i),T_2)|}{D(T_1,T_2)}$.
- The gain in calculation time $\gamma = 1 - \frac{t_f}{t_i}$.

With the same 3 models used in the previous experiments we made a random choice to choose a model that we use to generate 2 plants T_1 and T_2 then we build different approximations of T_1 corresponding to different compression rates τ_{min} among $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$. We calculated each time the distances $D(T_1, T_2)$ and $D(T_1(\tau_i), T_2)$ as well as the measures δ and γ. We perform this 10 times and the results are shown in figures Fig.4.13 and Fig.4.14. As we can see, the more we compress the greater the error rate on the calculation of the edit distance and at the same time the gain in calculation time is also significant. The error on the edit distance calculation increases the more we compress but remains around 40% even when reaching high compression rates (90%). On the other hand, the calculation time gain also increases to reach high rates for a low loss of information.

All experiments were carried out on a 2.3 GHz Intel core I7 quad core with 16 GB of DDR3 ram.

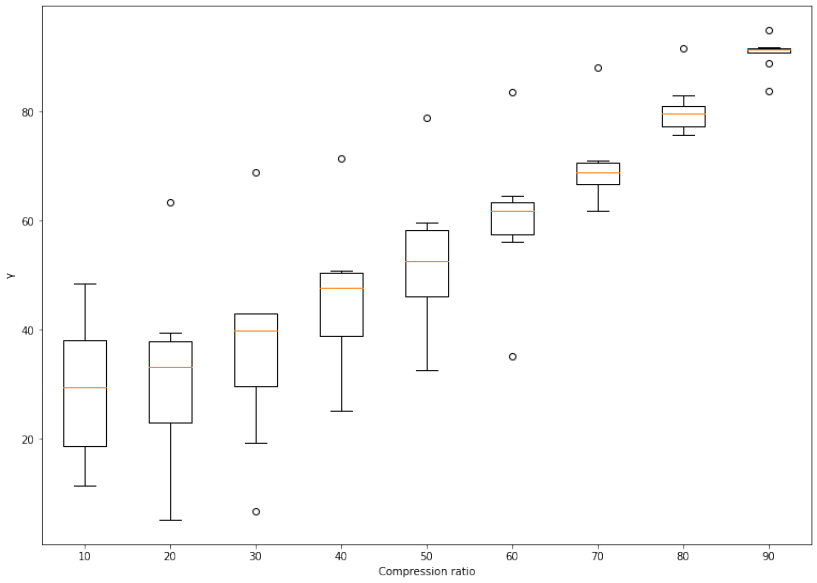


Figure 4.13: Evolution of the gain in the edit distance computation time (on the y-axis) against the minimum compression rate (on the x-axis)

4.6 Towards an exploration of symmetries in plant architectures

xThe introduction of the notion of cuts in the study of tree architectures has awakened our intention to another property that characterises these structures and that can be observed very clearly in the architectures of plants.

Plant architectures are characterised by symmetries present at several levels and often reflect the repetition of the mechanisms of morphogenesis. Symmetries are closely related to the mathematical concept of invariance, an object x is said to be invariant under a certain operator T when it remains unchanged under the action of the operator, i.e. $T(x) = x$. Observing symmetries means identifying the operators

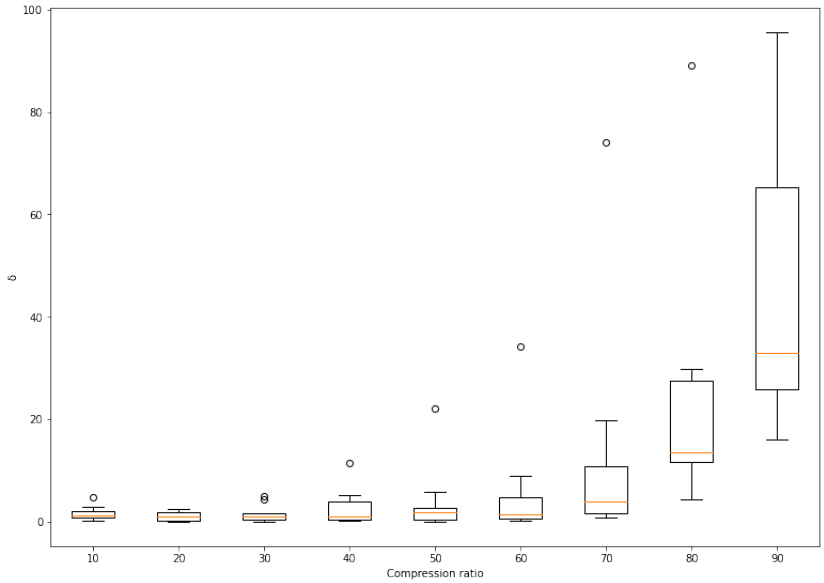


Figure 4.14: Evolution of the error rate (on the y-axis) against the minimum compression rate (on the x-axis)

under whose action an object remains unchanged. However, in plant architecture, due to biological noise and environmental effects, symmetries are often imperfect and correspond to $T(x) \approx x$. Here we investigate how these concepts can be applied to identify symmetries in plant architectures. symmetries in plant architectures.

4.6.1 *From cuts to antichains and Maximal symmetry antichain*

As we have seen in this chapter how cuts allow us to isolate a set of subtrees that are not in a parent-child relationship, we introduce in this section another way to isolate subtrees that are not in a parent-child relationship. An antichain is a set of subtrees that can be swapped pairwise (one can replace the other). We define a symmetry antichain as an antichain of subtrees that can be swapped without losing the plant's architecture, i.e. for any pair of subtrees t and s in the antichain, the tree $T^{'}$ obtained after their swap is equal (or at least very similar) to the original tree T. Such a symmetry antichain is said maximal when there is no other subtree within T that can be swapped with one of the antichain while preserving the general architecture. Detecting the symmetries actually means identifying all the maximal symmetry antichains.In a real plant architecture, one does not expect to observe exact symmetries. One must therefore be able to quantify the similarity between subtrees hence the interest of using the tree edit distance [40].

4.6.2 *The graph of maximal symmetry antichains*

It is clear that on a tree we can identify several maximal symmetry antichains and even if the subtrees within a same antichain are not related it is not the case from one antichain to another hence the idea to consider the graph of maximum

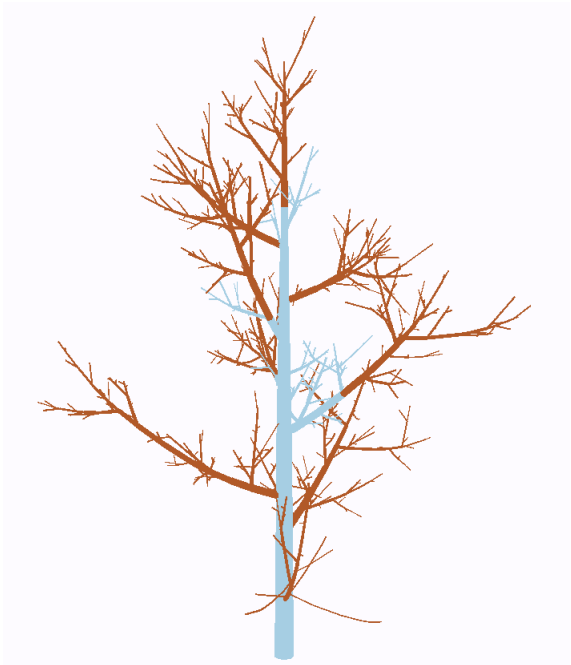


Figure 4.15: A plant architecture where we show the subtrees of an antichain colored with the same color

symmetry: antichains: two maximal symmetry antichains are linked by an edge when subtrees of one appear as descendants of subtrees of the other, and the weight of the edge corresponds to the proportion of such links Fig 4.16 shows an example of plant architecture, its maximal symmetry antichains and the graphs. Using this graph, we can therefore not only observe all the symmetries of a plant architecture but also their nesting. We have thus developed an algorithm to build all maximal symmetry antichains of the plant architecture that proceeds through a clustering of all its subtrees based on the above tree edit distance, instead of an exact match. The project is still being published, part of the work will be presented at the next edition of the Functional-Structural Plant Models conference (FSPM March 27-31, 2023) and more details will be presented in other manuscripts.

4.7 Conclusion

In this chapter, we introduced a lossy compression model of trees. The model is a continuity of a previous work done in the context of lossless compression of unordered trees [16]. The idea is to go through an approximation of the initial tree by another with better compression properties before applying the lossless compression model on the resulted approximation. The approximation consists in replacing the subtrees having a certain similarity by a single model in order to multiply its occurrence in the tree to achieve a better compression rate. The efficiency of the model is quantified by information loss rate due to compression measured by the ratio of the distance between the initial tree and its approximation to the distance between the initial

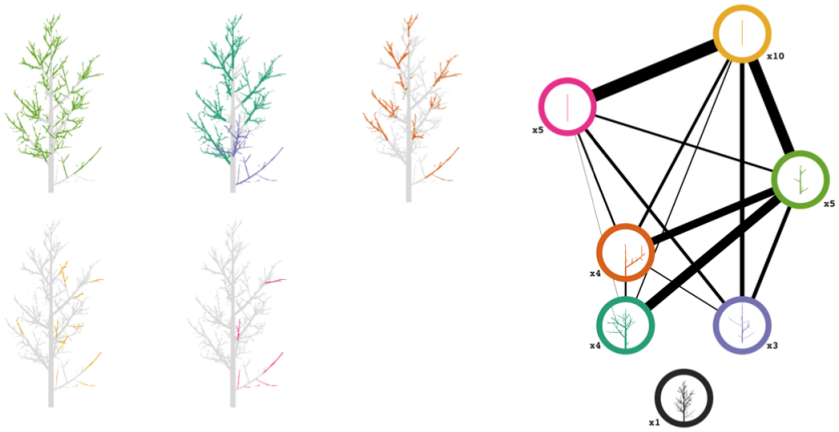


Figure 4.16: A plant architecture with 6 maximal symmetry antichains (identically colored subparts of the plant) which nesting links are given by the graph on the right.

tree and the empty tree, as well as the compression ratio measured by the ratio between the size of the DAG and the size of the initial tree. We then illustrated this model on artificial tree compression. We also evaluated the contribution of the model in the calculation of the edit distance between these plants.

Conclusion and perspectives

Starting from the fact that the size of the trees is exploding in front of the high complexity of the algorithms of edit distance calculation, we presented in this thesis methods of optimization of these algorithms by compressing the trees in DAGs to make the calculation afterwards on the DAGs instead of on the trees.

We have studied the work done in the context of lossless compression of unordered trees and have proposed a generalization of these methods for ordered and semi-ordered trees. For that, it was necessary to determine the similar subtrees using the calculation of a distance of edition. These last ones define a relation of equivalence, we thus calculated the classes of equivalence to build a quotient graph. We showed how to augment in each case this quotient graph to build a compression as well as the reconstruction of the initial tree starting from its compression. The proposal was implemented and tested on representations of theoretical plant architecture. Although our method has been tested on a few simulated plants, in order to analyse the robustness of the method, the implementation of larger tests on different data would allow us to study the influence of different plant architectures on the compression. But also the consideration of the information carried on the nodes of the trees adds interest to the study.

In the second part of the manuscript, we considered the problem of lossy compression of trees, we saw that it is a more pushed compression but which does not allow the reconstruction of the initial tree from its compression,a quantity of information is thus lost and the challenge was to find a happy medium between what we gain in compression and what we lose in data. Starting from a distance calculation between the tree and itself to determine the close sub-trees, the idea is to approximate the close sub-trees by a single representative to build an approximation of the input tree. The lossless compression of the resulting approximation is the lossy compression of the initial tree. We have presented methods for the grouping of similar subtrees, the choice of representative of each group and the approximation of subtrees of the same group by their representative. to remedy the problem of hierarchies in the tree we proposed to make an horizontal cut on the tree and to perform the procedure (clustering, selection of representatives and approximation) only on isolated subtrees. We finally proposed our lossy compression technique which was implemented and tested on plant architecture representations. A more extensive test and even on real plants as well as the consideration of other parameters (symmetry, architecture...) and more real applications are desired to complete this work.

Appendix A

Ordered Tree Compression pseudo algorithms

Algorithm 1: Vertices signature computation.

```
Data: Ordered Tree T = (V, E)
Result: List of signatures of T vertices
Function add_signatures(T);
foreach leaf l ∈ V do
    l.signature = [];
    l.signaturelabel = 0;
list_signature = [[]];
n_signature = 1;
N:=the list of leafs from V;
while N is not empty do
    foreach vertex n ∈ N do
        foreach vertex m ∈ child(n) do
            n.signature+ = [m.signaturelabel];
        if (list_signature.contain(n.signature));
         then
            n.signaturelabel = list_signature[n.signature];
        else
            list_signature+ = [n.signature];
            n.signaturelabel = n_signature;
            n_signature+ = 1;
    empty N ;
    add to N all not signed vertices for which all children have a signature;
return list_signature;
```

Algorithm 2: Tree Reconstitution.

Data: DAG $D = (V, E)$
Result: Tress $T = (node_list, edge_list)$
Function $Reconstitution(Dag_node\ m, Tree_node\ father)$::;
$node_list = []$;
$edge_list = []$;
if *(m is a leaf);*
then
    create $Tree_node\ n$;
    $node_list.add(n)$;
else
    create $Tree_node\ n$;
    $node_list.add(n)$;
    foreach $M_i \in child(M)$ **do**
        foreach $i \in e(M, M_i).list()$ **do**
            create $Tree_edge\ e_t$=$(n, Reconstitution(M_i, n, i))$;
            $edge_list.add(e_t)$;

Algorithm 3: Tree reduction algorithm.

```
Data: Ordered Tree T = (V, E)
Result: DAG D = (node_list, edge_list)
node_list = [];
edge_list = [];
list_signatures = add_signatures(T);
foreach node x ∈ V do
    n = new Node(x.signaturelabel) ;
    if (!node_list.contain(n));
     then
        node_list+ = [n];
    else
foreach node x ∈ V do
    for i = 1; i ≤ child(x); i = i + 1 do
        e = newedge;
        e.begin = x.signaturelabel;
        e.end = child(i).signaturelabel;
        if (edge_list.contain(e));
         then
            edge_list[e].list+ = [i];
        else
            e.list = [i];
            edge_list+ = e;
return node_list, edge_list;
```

Appendix B

Lossless Compression of Semi-Ordered Trees pseudo algorithms

Algorithm 4: Vertices signature computation.

Function *signatures*(T)**:**
- **Input :** Semi-ordered Tree $T = (V, E, S_{\sqsubseteq})$
- **Output:** $\sigma(T)$
- $\sigma(T) \leftarrow \emptyset$
- $N \leftarrow leaves(T)$
- **while** N *is not empty* **do**
  - **foreach** *vertex* $n \in N$ **do**
    - $\sigma(n) \leftarrow \emptyset$
    - **foreach** $P \in \Gamma(n)$ **do**
      - $d \leftarrow \emptyset$
      - **foreach** *vertex* $x \in P$ **do**
        - $d \leftarrow d + \sigma(x)$
      - $\sigma(n) \leftarrow \sigma(n) + d$
    - **if** $\sigma(n)) = \emptyset$ **then** $\sigma(n) \leftarrow \theta$
    - **if** $\sigma(n)) \notin \sigma(T)$ **then** $\sigma(T) \leftarrow \sigma(T) + \sigma(n)$
  - $V \leftarrow V - N$
  - $N \leftarrow \emptyset$
  - $N \leftarrow \{n \in V / \forall n_i \in child(n), \sigma(n_i) \in \sigma(T)\}$ `// add to` N `all not signed vertices for which all children have a signature`
- Return $\sigma(T)$

Algorithm 5: Tree reduction.

Function *Reduction*(T)**:**

Input : Semi-ordered Tree $T = (V, E, S_{\sqsubseteq})$

Output: its reduction $R(T) = (node_list, edge_list)$

$node_list \leftarrow \sigma(T)$
$edge_list \leftarrow \emptyset$
$C \leftarrow \emptyset$
foreach *node* $x \in V$ *with* $x \notin C$ **do**
  foreach $P_b \in \Gamma(x)$ **do**
    foreach *vertex* $y \in P_b$ **do**
      $e \leftarrow (\sigma(x), \sigma(y))$ `// e is a new edge`
      if $e \notin edge_list$ **then**
        $a \leftarrow 1$
        $l_e \leftarrow \{(a, b)\}$
        $edge_list \leftarrow edge_list \cup \{(e, l_e)\}$
      else
        if $\exists (a, b) \in l_e$ **then**
          $a \leftarrow a + 1$
        else
          $a \leftarrow 1$
          $l_e \leftarrow l_e \cup \{(a, b)\}$
  $C \leftarrow C \cup \{x\}$
  foreach $x' \in V$ *such that* $\sigma(x) = \sigma(x')$ **do**
    $C \leftarrow C \cup \{x'\}$ `// Avoid repeating calculation for isomorphic parts`
- Label each signature of $\sigma(T)$ with a unique integer value, starting from 0 for leaves signature
- Replace each signature in $node_list, edge_list$ by its label
Return $(node_list, edge_list)$

Algorithm 6: Tree Reconstruction.

Function *Reconstruction*$(R(T))$**:**

Input : The reduction $R(T) = (node_list, edge_list)$ of a tree T
Output: The semi-ordered Tree $T = (V, E, S_{\sqsubseteq})$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
$S_{\sqsubseteq} \leftarrow \emptyset$
Create $Tree_node$ r // r is the root of the new tree
$V \leftarrow V + r$
Let d the root node of the DAG $R(T)$
$(V, E, S_{\sqsubseteq}) \leftarrow Create_Tree(R(T), d, r, V, E, S_{\sqsubseteq})$
Return $(V, E, S_{\sqsubseteq})$

Function $Create_Tree(R(T), I, x, V, E, S_{\sqsubseteq})$**:**

Input : - the reduction DAG $R(T) = (node_list, edge_list)$,
- the node I the root of the subDAG of $R(T)$ to decompress,
- the tree node x such that $I = c(x)$,
- tree T elements $(V, E, S_{\sqsubseteq})$ in recursive computation.
Output: Tree $T = (V, E, S_{\sqsubseteq})$.

$\Gamma(x) \leftarrow \emptyset$
foreach $J \in child(I)$ **do** // Creation of empty partitions P_b of $\Gamma(x)$
  foreach $(a, b) \in l_{(I,J)}$ **do**
    if $P_b \notin \Gamma(x)$ **then**
      $P_b \leftarrow \emptyset$
      $\Gamma(x) \leftarrow \Gamma(x) + P_b$
foreach $J \in child(I)$ **do** // Construction of partitions P_b of $\Gamma(x)$
  foreach $(a, b) \in l_{(I,J)}$ **do**
    $i \leftarrow 1$
    while $i \leq a$ **do**
      Create $Tree_node$ x_i
      $V \leftarrow V + x_i$
      $P_b \leftarrow P_b + x_i$
      $E \leftarrow E + (x, x_i)$
      $(V, E, S_{\sqsubseteq}) \leftarrow Create_Tree(R(T), J, x_i, V, E, S_{\sqsubseteq})$
      $i \leftarrow i + 1$
$S_{\sqsubseteq} \leftarrow S_{\sqsubseteq} + \Gamma(x)$
Return $(V, E, S_{\sqsubseteq})$

Appendix C

Lossy compression of labeled trees pseudo algorithms

Algorithm 7: Get a tree cut from a global clustering

Function $cut_from_clustering(T, \mathcal{G})$**:**

Input : - a tree T, - a set of T global clusters $\mathcal{G}$.

Output: a cut $\mathcal{C}(T) = (\mathcal{T}, \kappa)$

$shuffle(\mathcal{G})$
$\mathcal{T} \leftarrow T$
$\kappa \leftarrow \phi$
for $\mathcal{G}_i \in \mathcal{G}$ **do**
    $shuffle(\mathcal{G}_i)$
    for $t_j \in \mathcal{G}_i$ **do**
        if *t_j is disjoint with all κ trees* **then**
            Add t_j to κ
            Replace t_j in $\mathcal{T}$ by an anchor node
Return $\mathcal{T}, \kappa$

Algorithm 8: Generate a set of cuts for a tree.

Function $cut_generator(T, n, k_{min}, k_{max})$**:**

Input : - Tree T,
- n the number of cuts to generate from each global clustering of T,
- k_{min}, k_{max} parameters for the k-medoid clustering method.

Output: Γ a set of T cuts

$\Gamma \leftarrow \emptyset$
$\mathcal{S} \leftarrow set\ of\ all\ T\ subtrees$
for $k \in [k_{min}, k_{max}]$ **do**
  $\mathcal{G} \leftarrow clustering(\mathcal{S}, k)$ // $\mathcal{S}$ clustering by the k-medoid method
  for $i \in [\![1, n]\!]$ **do**
    $C \leftarrow cut_from_clustering(T, \mathcal{G})$
    Add C to Γ
Return Γ

Algorithm 9: Compute the representatives of all local clusters.

Function *get_representatives_of_clusters*$(M, \mathcal{L})$**:**

Input : - M the subtrees distances matrix of the initial Tree T,
- $\mathcal{L}$ list of T local clusters

Output: list of local clusters representatives $\mathcal{R}$

$\mathcal{R} \leftarrow \emptyset$
for $\mathcal{L}_i \in \mathcal{L}$ **do**
  for $e \in \mathcal{L}_i$ **do**
    Compute $\zeta(e, \mathcal{L}_i)$
  Sort the j's in $\mathcal{L}_i$ in increasing order by of $\zeta(\mathcal{L}_i[j], \mathcal{L}_i)$
  Add $\mathcal{L}_i[1]$ to $\mathcal{R}$

$R^* \leftarrow \phi$
do
  $R^* \leftarrow \mathcal{R}$
  $\tau_0 \leftarrow \Upsilon(R^*)$ `// The list of trees taken as a forest`
  $i \leftarrow$ Choose a random in $[\![1, |R^*|]\!]$
  $j \leftarrow 1+$The index of $R^*[i]$ in $\mathcal{L}[i]$
  $\tau_j \leftarrow \Upsilon(\{R^* \setminus R^*[i]\} \cup \{\mathcal{L}[i][j]\})$
  do
    replace $R^*[i]$ by $\mathcal{L}[i][j]$ in R^*
    $\tau_0 \leftarrow \tau_j$
    $j \leftarrow j + 1$
    $\tau_j \leftarrow \Upsilon(\{R^* \setminus R^*[i]\} \cup \{\mathcal{L}[i][j]\})$
  while $\tau_j \geq \tau_0$
  $\mathcal{R} \leftarrow R^*$
while $\mathcal{R} \neq R^*$
Return $\mathcal{R}$

Algorithm 10: Compute an approximation from a cut.

Function $approximation_from_cut(T, \mathcal{C}(T), M)$**:**

Input : - T the initial Tree,
- $\mathcal{C}(T) = (\mathcal{T}, \kappa)$ a T cut,
- M the matrix of distances between T subtrees

Output: - T^* an approximation of T,
- D^* its lossless compression,
- σ,τ the associated loss of information and compression ratio.

$\mathcal{L} \leftarrow clustering(\kappa, M)$ `// k-medoid method perform the` κ `clustering`
$\mathcal{R} \leftarrow get_representatives_of_clusters(M, \mathcal{L})$
$T^* \leftarrow \mathcal{T}$ `// start the approximation`
for $i \leftarrow 1$ **to** $|\mathcal{L}|$ **do**
  for $d \in \mathcal{L}[i]$ **do**
    - d is replaced by $\mathcal{R}[i]$ so that $\mathcal{R}[i]$ is connected to the anchor of d in T^*
$D^* \leftarrow Reduction(T^*)$ `// As in the lossless compression`
$\sigma = \Delta(T, T^*)/\Delta(T, \theta)$
$\tau = 1 - (size(D^*)/size(T))$
Return T^*, D^*, σ, τ

Algorithm 11: Lossy compression of a labelled tree.

Function $Lossy_compression(T, p, N, k_{min}, k_{max})$**:**

Input : - Tree $T = (V, E)$,
- The parameter p representing either the target compression ratio τ_{min} or the allowed loss ratio σ_{max},
- Number of cuts N,
- k_{min}, k_{max} parameters for the k-medoid clustering method.

Output: - A set of T approximations according to the parameter p.

$M \leftarrow [\Delta(T[i], T[j])]_{\forall (i,j) \in [1,|T|] \times [1,|T|]}$
// compute the distance between all T subtrees
$\Gamma \leftarrow cut_generator(T, N, k_{min}, k_{max})$
$X \leftarrow \emptyset$
foreach $C \in \Gamma$ **do**
  $T^*, D^*, \sigma, \tau \leftarrow approximation_from_cut(T, C, M)$
  $X = X \cup \{(T^*, D^*, \sigma, \tau)\}$
if *p is a target compression ratio τ_{min}* **then**
  $Y \leftarrow \{x \text{ for } x \in X \text{ and } x[3] \geq \tau_{min}\}$
  $\sigma_{min} \leftarrow min\{x[2] \text{ for } x \in Y\}$
  $Z \leftarrow \{x \in Y \text{ where } x[2] = \sigma_{min}\}$
if *p is a the allowed loss ratio σ_{max}* **then**
  $Y \leftarrow \{x \text{ for } x \in X \text{ and } x[2] \leq \sigma_{max}\}$
  $\tau_{max} \leftarrow max\{x[3] \text{ for } x \in Y\}$
  $Z \leftarrow \{x \in Y \text{ where } x[3] = \tau_{max}\}$
Return Z

Appendix D

Stochastic plant models

D.1 Model M_1

$$A = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \quad L = \begin{pmatrix} 0.0 & 0.83 & 0.17 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.3 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.3 & 0.7 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

$$min = \begin{pmatrix} 6 & 4 & 4 & 2 & 1 \end{pmatrix} \qquad max = \begin{pmatrix} 8 & 8 & 4 & 2 & 1 \end{pmatrix}$$

D.2 Model M_2

$$A = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \quad L = \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.0 & 0.0 & 0.0 & 0.1 & 0.2 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.2 \end{pmatrix}$$

$$min = \begin{pmatrix} 8 & 7 & 6 & 5 & 2 & 1 \end{pmatrix} \qquad max = \begin{pmatrix} 8 & 7 & 6 & 5 & 2 & 1 \end{pmatrix}$$

D.3 Model M_3

$$A = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \quad L = \begin{pmatrix} 0.07 & 0.83 & 0.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.3 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.7 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

$$min = \begin{pmatrix} 6 & 4 & 4 & 2 & 1 \end{pmatrix} \qquad max = \begin{pmatrix} 8 & 8 & 4 & 2 & 1 \end{pmatrix}$$

Bibliography

[1] J. Allali. "Comparaison de structures secondaires d'ARN". PhD thesis. Université de Marne la Vallée, 2004.

[2] R. Azaıs and F. Ingels. "The weight function in the subtree kernel is decisive". In: *Journal of Machine Learning Research* 21 (2020), pp. 1–36.

[3] R. Azaıs, J. Durand, and C. Godin. "Approximation of trees by self-nested trees". In: *2019 proceedings of the twenty-first workshop on algorithm engineering and experiments (alenex)*. SIAM. 2019, pp. 39–53.

[4] R. Azaıs et al. "Treex: a Python package for manipulating rooted trees". In: *Journal of Open Source Software* 4.38 (2019), p. 1351.

[5] P. Bille. "A survey on tree edit distance and related problems". In: *Theoretical computer science* 337.1-3 (2005), pp. 217–239.

[6] F. Boudon et al. "L-Py, an open L-systems framework in Python". In: 2010.

[7] Y. Cheng and S. Lu. "Waveform correlation by tree matching". In: *IEEE transactions on pattern analysis and machine intelligence* 3 (1985), pp. 299–305.

[8] M. Chodorow and J.L. Klavans. "Locating syntactic patterns in text corpora". In: *Manuscript, Lexical systems, IBM Research, TJ Watson Research Center, Yorktown Heights, New York* (1990).

[9] E.D. Demaine et al. "An optimal decomposition algorithm for tree edit distance". In: *ACM Transactions on Algorithms (TALG)* 6.1 (2009), pp. 1–19.

[10] S. Dulucq and H. Touzet. "Analysis of tree edit distance algorithms". In: *Annual Symposium on Combinatorial Pattern Matching*. Springer. 2003, pp. 83–95.

[11] J. Edmonds and R.M. Karp. "Theoretical improvements in algorithmic efficiency for network flow problems". In: *Journal of the ACM (JACM)* 19.2 (1972), pp. 248–264.

[12] L. Euler. "Solutio problematis ad geometriam situs pertinentis". In: *Commentarii academiae scientiarum Petropolitanae* (1741), pp. 128–140.

[13] P. Ferraro, C. Godin, and P. Prusinkiewicz. "Toward a quantification of self-similarity in plants". In: *Fractals* 13.02 (2005), pp. 91–109.

[14] C. Godin and Y. Caraglio. "A multiscale model of plant topological structures". In: *Journal of theoretical biology* 191.1 (1998), pp. 1–46.

[15] C. Godin, E. Costes, and H. Sinoquet. "Plant architecture modelling: virtual plants and complex systems". In: *Annual Plant Reviews online* (2018), pp. 238–287.

[16] C. Godin and P. Ferraro. "Quantifying the degree of self-nestedness of trees: application to the structural analysis of plants". In: *IEEE/ACM transactions on computational biology and bioinformatics* 7.4 (2009), pp. 688–703.

[17] J.L. Gross and J. Yellen. *Handbook of graph theory*. CRC press, 2003.

[18] J.L. Gross, J. Yellen, and M. Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[19] S. Habibeche and F. Ben-Naoum. "Ordered Tree Compression". In: *ICA2IT19: International Conference on Artificial Intelligence and Information Technology*. Université Kasdi Merbah Ouargla, Algeria, March. 2019.

[20] S. Habibeche and F. Ben-Naoum. "Lossless Compression of Semi-Ordered Trees". In: *International Journal of Applied Research in Bioinformatics (IJARB)* 12(1).8 (2022).

[21] C. Herrbach. "Etude algorithmique et statistique de la comparaison des structures secondaires d'ARN". PhD thesis. 2007.

[22] T. Jiang et al. "A general edit distance between RNA structures". In: *Journal of computational biology* 9.2 (2002), pp. 371–388.

[23] P.N. Klein. "Computing the edit-distance between unrooted ordered trees". In: *European Symposium on Algorithms*. Springer. 1998, pp. 91–102.

[24] V. Levenshtein et al. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics doklady*. Vol. 10. 8. Soviet Union. 1966, pp. 707–710.

[25] S. Mondet et al. "Streaming of plants in distributed virtual environments". In: *Proceedings of the 16th ACM international conference on Multimedia*. 2008, pp. 1–10.

[26] A. Ouangraoua and P. Ferraro. "A constrained edit distance algorithm between semi-ordered trees". In: *Theoretical computer science* 410.8-10 (2009), pp. 837–846.

[27] A. Ouangraoua et al. "Plant architecture comparison methods: A review of existing algorithms and examples of application". In: 2007.

[28] H. Park and C. Jun. "A simple and fast algorithm for K-medoids clustering". In: *Expert systems with applications* 36.2 (2009), pp. 3336–3341.

[29] F.P. Preparata and R.T. Yeh. *Introduction to Discrete Structures for Computer Science and Engineering*. USA: Addison-Wesley Longman Publishing Co., Inc., 1973. ISBN: 0201059681.

[30] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

[31] K. Sayood. *Introduction to data compression*. Morgan Kaufmann, 2017.

[32] B.A. Shapiro and K. Zhang. "Comparing multiple RNA secondary structures using tree comparisons". In: *Bioinformatics* 6.4 (1990), pp. 309–318.

[33] F.Y. Shih. "Object representation and recognition using mathematical morphology model". In: *Journal of Systems Integration* 1.2 (1991), pp. 235–256.

[34] J.J. Sylvester. "Chemistry and algebra". In: *Nature* 17.432 (1878), p. 284.

[35] K. Tai. "The tree-to-tree correction problem". In: *Journal of the ACM (JACM)* 26.3 (1979), pp. 422–433.

[36] E. Tanaka and K. Tanaka. "The tree-to-tree editing problem". In: *International Journal of pattern recognition and artificial intelligence* 2.02 (1988), pp. 221–240.

[37] R.E. Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[38] R. A Wagner and M.J. Fischer. "The string-to-string correction problem". In: *Journal of the ACM (JACM)* 21.1 (1974), pp. 168–173.

[39] K. Zhang. "Algorithms for the constrained editing distance between ordered labeled trees and related problems". In: *Pattern recognition* 28.3 (1995), pp. 463–474.

[40] K. Zhang. "A constrained edit distance between unordered labeled trees". In: *Algorithmica* 15.3 (1996), pp. 205–222.

[41] K. Zhang and T. Jiang. "Some MAX SNP-hard results concerning unordered labeled trees". In: *Information Processing Letters* 49.5 (1994), pp. 249–254.

[42] K. Zhang and D. Shasha. "Simple fast algorithms for the editing distance between trees and related problems". In: *SIAM journal on computing* 18.6 (1989), pp. 1245–1262.

[43] K. Zhang, R. Statman, and D. Shasha. "On the editing distance between unordered labeled trees". In: *Information processing letters* 42.3 (1992), pp. 133–139.