Pattern Analysis and Applications
https://doi.org/10.1007/s10044-021-00971-3

THEORETICAL ADVANCES

Customizable HMM-based measures to accurately compare tree sets

Sylvain Iloga[1,2,3]

Received: 14 August 2020 / Accepted: 1 March 2021

Abstract
Trees have been topics of much interest since many decades due to various emerging applications using data represented as trees. Several techniques have been developed to compare two trees. But there is a serious lack of metrics to compare weighted trees. Existing approaches do not also allow to explicitly specify the targeted nodes properties on which the comparison should be performed. Furthermore, the problem of comparing two tree sets is not specifically addressed by existing techniques. This paper attempts to solve these problems by first proposing a distance and a similarity for the comparison of two finite sets of rooted ordered trees which can be labeled or not, as well as weighted or unweighted. To achieve this goal, a hidden Markov model is associated with each tree set for each targeted nodes property. The model associated with a tree set T for the targeted nodes property p learns how much the nodes of the trees in T verify property p. The resulting models are finally compared to derive a distance and similarity between the two sets of trees. The previous measures are then generalized for the comparison of unrooted and unordered trees. Flat classification experiments were carried out on two synthetic databases named *FirstLast-L* and *FirstLast-LW* available online. They both contain four classes of 100 rooted ordered trees whose specific and non-trivial nodes properties are clearly defined. When the distance proposed in this paper is selected as metric for the Nearest Neighbor classifier, a perfect accuracy of 100% is obtained for these two databases. This performance is 41% higher than the accuracy exhibited when the widespread tree Edit distance is selected for *FirstLast-L*.

Keywords Trees · Comparison of tree sets · Distance between trees · Hidden Markov Models

1 Introduction

Trees have been topics of much interest since many decades due to several emerging applications using data represented as trees. This involves applications in compiler design, graph transformation, automatic theorem providing, information retrieval, structured text database, pattern recognition, as well as signal processing [1, 2]. The analysis of data in these areas often requires some form of dissimilarity or distance measure to compare trees. In image processing for example, tree matching techniques are used to evaluate the similarity between 2D shapes [3]. Another interesting example is found in e-Business where tree similarity algorithms enable the match-making of agents in multi-agent systems [4]. For these reasons, developing measures for comparing two trees have become the goal of several researchers.

Depending on the category of the trees (rooted or unrooted, ordered or unordered) many metrics are available for the comparison of labeled or unlabeled trees. Most of the existing metrics are based on one of the three following widespread principles:

1. *Tree Edit* [5–25].
2. *tree Alignment* [26, 27].
3. *Tree Inclusion* [28–32].

The major differences between the existing measures based on one of these three principles are generally related to the time cost and the space complexity. These differences are analyzed in the survey proposed in [2]. Beside these three major principles, few measures based on other principles have also been proposed [3, 33–43].

✉ Sylvain Iloga
sylvain.iloga@gmail.com

[1] Department of Computer Science, Higher Teachers' Training College, University of Maroua, P.O.box 55, Maroua, Cameroon

[2] CY Cergy Paris University, ENSEA, CNRS, ETIS UMR 8051, 95000 Cergy, France

[3] University of Sorbonne, IRD, UMMISCO, 93143 Bondy, France

Published online: 31 March 2021

The first observation that can be made is the serious lack of measures for comparing weighted trees. This may be due to the fact that node and arc weights are continuous real numbers, unlike node and arc labels which always belong to finite sets.

The next drawback of existing measures is that they do not enable to explicitly specify the targeted nodes properties on which the tree comparison must be performed. Indeed, all the actual algorithms are only intended to evaluate how identical are the two trees regarding the exact values of their nodes degrees and their eventual nodes or arcs labels. This is a serious limitation because two trees can have completely different topologies, labels or weights, but still remain very similar regarding other properties verified by these same characteristics. As an example, all the labels appearing in two trees can be odd integers, but no identical label is found in both trees. Consequently, the two trees are completely different regarding the exact values of their labels, but they are very similar regarding the parity of these labels.

Another significant limitation of existing approaches is the fact that no existing work has yet specifically addressed the problem of comparing two tree sets. A coarse way to perform this task is to first consider each tree set as a cluster of trees and then to apply existing cluster distances like the *minimum*, the *maximum*, the *average* or the *centroid* linkage distances [44]. But such an approach does not consider the individual properties of the trees of each set.

This paper attempts to provide a solution to all the aforementioned limitations. Our contribution is thus composed of three main parts. In the first part, a *customizable* approach based on hidden Markov models (HMMs) is proposed for the comparison of two finite sets containing rooted ordered trees which can be labeled or unlabeled, as well as weighted or unweighted. The attribute '*customizable*' refers here to the fact that one must explicitly specify the targeted nodes properties on which the tree comparison will be performed. The principle is to traverse the trees of each set, while capturing the values of the targeted properties for each node encountered during the traversal. These values are later used to initialize, then to train one HMM per property, for each tree set. The distance and the similarity between the two tree sets are finally derived from the similarity between these HMMs. An interesting advantage of this solution is that there is no boundary on the possible choices of the targeted nodes properties. The four properties listed below are examples of properties which can be targeted for a given node y.

1. $p(y)$ is the number of children of y.
2. $p(y)$ is the label of y.
3. $p(y) = 1$ if y has more than three children, else $p(y) = 0$.
4. $p(y) = 1$ if y is not the root and the weight of the arc linking y to its father is lower than 55, else $p(y) = 0$.

In the second main part of our contribution, the former proposed distance and similarity between sets containing rooted ordered trees are generalized for the comparison of sets containing unrooted or unordered trees. In both cases, the comparison is realized after transforming each unrooted or unordered tree into one set of rooted ordered trees.

In the last main part of our contribution, the proposed distance is first experimented for the comparison of sets of text documents. Then, a performance evaluation of the proposed distance is realized by evaluating how much it can empower the performances of the *Nearest Neighbor* (*NN*) classifier [45] when suitable targeted nodes properties are selected.

The rest of this paper is organized as follows: the state of the art is presented in Sect. 2, followed by a summarized presentation of HMMs in Sect. 3. The description of the proposed approach for tree sets comparison is realized in Sect. 4, while experimental results are exhibited in Sect. 5. The last section is dedicated to the conclusion.

2 State of the art

2.1 Basic tree-related concepts

In this section, basic tree-related concepts that will be used throughout the paper are defined. Given that a tree is a specific type of graph, we start by defining few graph-related concepts that will later be required to define the tree-related concepts. A more detailed overview on graph theory and applications is available in [46]. After these definitions, the tree representation that will be used in this paper is presented. The section ends with the description of the algorithm dedicated to the traversal of a tree that has been selected in this paper.

2.1.1 Graph-related definitions

A ***graph*** G is a pair $< V, E >$ where:

1. V is a finite set of objects called vertices.
2. $E \subseteq V \times V$ is a set of edges, which are connections between vertices. More formally, E is a set of pairs (x, y) such that there exist in graph G a connection between the vertices x and y. If the pair $(x, y) \in E$, then x is adjacent to y.

Given a finite set L of labels, a label $l \in L$ can be assigned to each vertex (resp. to each edge) of a graph G. In that case, the graph is ***vertex-labeled*** (resp. ***edge-labeled***). Similarly, a positive continuous weight w can be assigned to each vertex (resp. to each edge) of a graph G. In that case, the graph is ***vertex-weighted*** (resp. ***edge-weighted***). A graph which is neither vertex-labeled, nor edge-labeled is ***unlabeled***.

Similarly, a graph which is neither vertex-weighted, nor edge-weighted is ***unweighted***. A path in a graph G is a sequence of edges connecting a sequence of vertices which are all distinct from one another. A cycle in a graph is a path such that the first vertex of the path corresponds to the last vertex. A graph containing at least one cycle is ***cyclic***, otherwise the graph is ***acyclic***. A graph where there exist a path from any vertex to any other vertex is ***connected***, otherwise the graph is ***disconnected***.

2.1.2 Tree-related definitions

A ***tree*** is a connected acyclic graph. Generally, the edges and the vertices of a tree are respectively called arcs and nodes. In this paper, the number of nodes of a tree t is noted $|t|$. A ***rooted*** tree t is a tree in which a special node called the '*root*' and denoted $root(t)$ is singled out. If such a node is not specified, then the tree is ***unrooted***. If a node y is adjacent to another node x in a rooted tree, then y is the father of x and x is a child for y. A node without children is called a *leaf*. An ***ordered*** tree is a tree where a left-to-right order is defined among the children of each node, otherwise the tree is ***unordered***. In a tree, the depth of a node y, denoted $depth(y)$, is the number of arcs in the path from the root to y. The depth of the root is 0. The degree of a node y, denoted $deg(y)$, is its number of children. In the rest of this paper, the expression '*tree*' without any other precision refers to a rooted ordered tree.

2.1.3 Representation of a tree

There are various ways for representing trees. In this paper, only the schematic representation (i.e: a figure describing the tree) is privileged because it facilitates the explanation of the proposed approach. In this representation, nodes are represented with circles, each circle carrying the eventual attributes (label, weight) of the node it describes. Arcs are represented with lines, each line also carries the eventual attributes (label, weight) of the arc it describes.

2.1.4 Traversal of a tree

To traverse a tree t means to visit all the nodes of t in some systematic order and there exist many algorithms for this purpose. In this paper, the *Depth-First Search* (*DFS*) algorithm [47] is selected due to its simplicity. *DFS* is a recursive algorithm which traverses the tree t starting from a specific node. While traversing t, *DFS* manages an ordered list of visited nodes using a stack, this list is empty at the beginning of the algorithm and must contain all the nodes of t at the end. The formal principle of *DFS* is the following:

1. Select a starting node and put it on top of the stack. If t is rooted, then $root(t)$ is selected.
2. Remove the node x on top of the stack, use it, then insert it at the end of the visited list.
3. Browse the all the children of x and put each encountered child of x which is not yet in the visited list on top of the stack. If t is ordered, the children of x are browsed from left to right.
4. If the stack is empty, then **stop**, else **goto** step 2.

2.2 Related work

2.2.1 Tree Edit

There are three basic edit operations: node insertion, node deletion and node substitution. We assume here that each edit operation has a specific user-defined cost. An edit script between two trees t_1 and t_2 is a sequence $S(t_1, t_2)$ of edit operations required for transforming t_1 into t_2. The cost of $S(t_1, t_2)$ is the sum of the costs of the edit operations in $S(t_1, t_2)$. An optimal edit script between t_1 and t_2 is an edit script between t_1 and t_2 of minimum cost and this cost is the *tree Edit distance* (TED). The first algorithm implementing this principle for comparing ordered trees was proposed by Tai [5]. This initial version was the generalization of the well-known *string edit distance problem* addressed by Wagner and Fischer [48]. An accelerated version which reduced both space and time complexity was then proposed by Zhang and Shasha [6]. In 1998, Klein proposed a more general version which handled unrooted trees and enabled to have a better worst case time bound while preserving the same space complexity [9]. After this, a more complex version which improved the previous bounds for certain kinds of trees was proposed by Chen [10]. A version running in linear time was later proposed by Touzet [11]. In 2009, Demaine et al. proposed a decomposition algorithm for *tree Edit* distance which exhibited a low worst-case time complexity when the two trees have n nodes [12]. A Robust algorithm for the Tree Edit Distance (RTED) which was both efficient and worst-case optimal was introduced by Pawlik and Augsten [13]. The same authors also proposed another version of the former algorithm name AP-TED (All Path Tree Edit Distance) running at least as fast as RTED without trading in memory efficiency [14]. A revised version of the proposal of Chen [10] was proposed by Schwarz et al. [15]. In this last version, the time complexity has been improved and a new traversal strategy has been implemented to reduce the memory complexity.

The difficulties to solve the general ordered *tree Edit* distance problem led to the study of a restricted version of this problem named *constrained Edit distance* introduced in 1995 and 1996 by Zhang [16, 17]. Here, the *tree Edit* distance is defined under the restriction that disjoint subtrees

should be mapped to disjoint subtrees. In 1997, Richter proposed another solution to this problem which gave space improvement over the former solutions of Zhang for trees having small degrees and low depths [18]. Lu, Su, and Tang proposed in 2001 to relax the constrained mapping by introducing the *less constrained Edit distance* [19]. A quotiented tree is a tree defined with an additional equivalent relation on arcs and such that the corresponding quotient graph is also a tree. In 2007, Ouangraoua et al. proposed a constrained edit scoring scheme based on [6] for comparing ordered quotiented trees [20].

Beside these constrained versions, several other variants of the ordered *tree Edit* distance have been proposed. This was for example the case in 1977 when Selkow proposed the *1-degree Edit distance* where insertions and deletions were restricted to the leaves of the trees [21]. Another variant where edit operations worked on subtrees instead of nodes was proposed by Lu [22]. An analog implementation of this variant was also proposed by Tanaka and Tanaka [23] . An important difficulty in the use of the TED resides in the choice of the cost of each edit operation. This is why in 1990, Shasha and Zhang proposed the *unit cost Edit distance* defined as the number $|S(t_1, t_2)|$ of edit operations needed to turn t_1 into t_2 (i.e: the cost of each edit operation is 1.0) [24]. Merge trees are important for applications related to feature-directed visualization of time-varying data. In 2018, a variant of the *tree Edit* distance for comparing merge trees was proposed by Sridharamurthy et al. [25].

The general problem of evaluating the *tree Edit* distance for rooted unordered trees with label nodes has been shown to be NP-Complete by Zhang and Shasha [6] and to be MAX-SNP-hard by Zhang and Jiang [8]. Nevertheless, Zhang et al. proved in 1992 that for some special cases, polynomial algorithms can be implemented [7].

2.2.2 Tree Alignment

The *tree Alignment* distance is a special case of the *tree Edit* distance where all insertions must be performed before any deletion. In 1995, Jiang et al. proposed the first implementation of the *tree Alignment* distance for rooted ordered trees with labeled nodes which could be computed more efficiently than the *tree Edit* distance for some trees with small degree [26]. A faster version of *tree Alignment* designed for similar trees was later proposed by Jansson and Lingas [27].

Consider two rooted unordered trees with labeled nodes t_1 and t_2. A modified version of the algorithm proposed by Jiang et al. [26] enabled to compute the *tree alignment* distance between t_1 and t_2. Jiang demonstrated that if t_1 or t_2 has an arbitrary degree, the unordered *tree Alignment* between these trees is NP-hard. However, if they both have bounded degrees, their *tree Alignment* distance can be computed in linear time, unlike the computation of their *tree Edit* distance which remains MAX-SNP-hard in the same conditions.

2.2.3 Tree inclusion

Let t_1 and t_2 be two rooted trees with labeled nodes. t_1 is *included* in t_2 if there is a sequence $S'(t_1, t_2)$ of node deletions performed on t_2 which makes t_2 isomorphic to t_1. The tree Inclusion problem is to decide if t_1 can be included in t_2 and the *tree inclusion* distance is the sum of the costs of the deletions found in $S'(t_1, t_2)$.

The first polynomial time algorithms for computing of the *tree inclusion* distance between two rooted ordered trees with labeled nodes were proposed by Kilpeläinen et al. [28] then by Kilpeläinen and Mannila [30]. In 1997, Richter proposed a modified version which improved the time complexity if the number of pair of nodes having identical labels is small [31]. Alonso and Schott also proposed an efficient average case version in 1993 [29] while a more complex version was developed by Chen [32].

Kilpeläinen and Mannila [30] also demonstrated that the computation of the *tree inclusion* distance between two rooted unordered trees with labeled nodes is NP-complete. In 1992, Matoušek and Thomas independently exhibited another proof of this NP-completeness [49].

2.2.4 Other principles of comparison

Beside the three former principles based on edit operations, other principles have been proposed for comparing two trees. Some authors opted for *tree pattern matching* which consist in finding the instances of a given pattern tree in a given target tree [3, 33–37]. Other authors rather preferred solutions related to subtrees or to supertrees similarity. This choice led them to find the *maximum agreement subtree* [38, 39], the *largest common subtree* [40, 41] or the *smallest common supertree* [42, 43]. Table 1 summarizes the main characteristics of the related work reviewed in this paper.

2.3 Problem statement

The first limitation of the state of the art is the serious lack of techniques for comparing weighted trees. An attempt for comparing unordered node-labeled trees with labeled and weighted arcs was proposed in 2004 by Bhavsar et al. [4]. But this measure was limited because arc weights must always belong to the interval [0, 1] and arc labels must be unique on each level.

Another major drawback of existing techniques for tree comparison is that they do not enable the user to explicitly specify the set of targeted nodes properties on which the comparison should be performed. The comparison is

Table 1 Main characteristics of relevant existing techniques for comparing of two rooted or unrooted trees, that can be ordered or unordered, as well as labeled or unlabeled. For each principle, references are listed chronologically

No.	Authors [Ref.]	Years	Principle	Specificity
1	Tai [5]	1979	General tree Edit	Initial version
2	Zhang and Shasha [6]	1989		Reduces space and time complexity
3	Zhang et al. [7]	1992		NP-complete unordered versions
4	Zhang and Jiang [8]	1994		
5	Klein [9]	1998		Better worst case time bound
6	Chen [10]	2001		Complexity improvements for some trees
7	Touzet [11]	2007		Runs in linear time
8	Demaine et al. [12]	2009		Low time complexity for trees having n nodes
9	Pawlik and Augsten [13]	2015		(RTED) Efficient and worst-case optimal
10	Pawlik and Augsten [14]	2016		(AP-TED) Memory efficient version of RTED
11	Schwarz et al. [15]	2017		Improves the proposal of [10]
12	Zhang [16]	1995	Constrained tree Edit	Disjoint subtrees are mapped to disjoint subtrees
13	Zhang [17]	1996		
14	Richter [18]	1997		Gives space improvement
15	Lu et al. [19]	2001		Relaxes the constrained mapping
16	Ouangraoua et al. [20]	2007		Designed for quotiented trees
17	Selkow [21]	1977	Other variants of tree Edit	Deletions are restricted to the leaves
18	Lu [22]	1979		Edit operations work on subtrees
19	Tanaka and Tanaka [23]	1988		
20	Shasha and Zhang [24]	1990		Each edit operation costs 1.0
21	Sridharamurthy et al. [25]	2018		Edit distance between merge trees
22	Jiang et al. [26]	1995	tree Alignment	Initial version, works on unordered trees
23	Jansson and Lingas [27]	2001		Faster version
24	Kilpeläinen et al. [28]	1992	Tree inclusion	Initial version
25	Alonso and Schott [29]	1993		Efficient average version
26	Kilpeläinen and Mannila [30]	1995		Polynomial version
27	Richter [31]	1997		Improves the time complexity
28	Chen [32]	1998		More complex version
29	Hoffmann and O'Donnell [33]	1982	Tree pattern matching	Research of the instances of a pattern tree in a target tree
30	Kosaraju [34]	1989		
31	Dubiner et al. [35]	1990		
32	Ramesh and Ramakrishnan [36]	1992		
33	Zhang et al. [37]	1994		
34	Liu and Geiger [3]	1999		
35	Farach and Thorup [38]	1995	Maximum agreement subtree	Comparison related to subtrees or supertrees
36	Amir and Keselman [39]	1997		
37	Khanna et al. [40]	1995	Largest common subtree	
38	Akutsu and Halldórsson [41]	2000		
39	Gupta and Nishimura [42]	1998	Smallest common supertree	
40	Nishimura et al. [43]	2000		

generally based on the exact values of the nodes degrees and the nodes or arcs labels.

Additionally, none of the existing techniques has yet been specifically dedicated for the comparison of tree sets. In 2006, Torsello and Hancock developed a clustering method for learning the class prototype and class structure of a set of unlabeled trees [50], which was generalized to graphs in 2011 [51]. In [50], a probabilistic model for describing the distribution of tree data in each cluster (tree set) was constructed under several simplifying assumptions. It is possible to use this method as a proxy for comparing two tree sets by evaluating the similarity between their associated probabilistic models. But this will be done without having the possibility of capturing specific

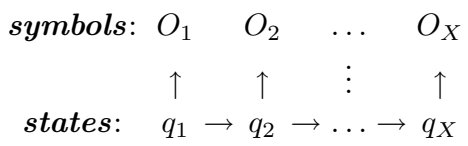


Fig. 1 HMM used as sequence generator

user-defined targeted nodes properties during the modeling process.

The goal of this paper is to attempt to provide solutions for these problems by initially proposing one distance and one similarity, both based on HMMs, to compare two finite sets containing rooted ordered trees which can be labeled or unlabeled, as well as weighted or unweighted. These measures are finally generalized for the comparison of two finite sets containing unrooted or unordered trees.

3 Hidden Markov Models

3.1 HMM definition

A HMM $\lambda = \{A, B, \pi\}$ is fully characterized by [52]:

1. The number N of states of the model. The set of states is $S = \{s_1, s_2, \ldots, s_N\}$. The state of the model at time x is generally noted $q_x \in S$.
2. The number M of symbols. The set of symbols is $\vartheta = \{v_1, v_2, \ldots, v_M\}$. The symbol observed at time x is generally noted $O_x \in \vartheta$.
3. The state transition probability distribution $A = \{a_{ij}\}$ where each $a_{ij} = P(q_{x+1} = s_j | q_x = s_i)$ with $1 \leq i, j \leq N$.
4. The symbols probabilities distributions $B = \{b_i(k)\}$ in each state s_i where each distribution $b_i(k) = P(v_k \text{ at time } x | q_x = s_i)$ with $1 \leq i \leq N$ and $1 \leq k \leq M$.
5. The initial state probability distribution $\pi = \{\pi_i\}$ where $\pi_i = P(q_1 = s_i)$ with $1 \leq i \leq N$.

3.2 HMM used as sequence generator

A HMM $\lambda = \{A, B, \pi\}$ can be used to generate a sequence $O = O_1 O_2 \ldots O_X$ composed of X symbols observed by the sequence of states $q = q_1 q_2 \ldots q_X$ as described in the *Markov chain* (MC) shown in Fig. 1.

In order to obtain the MC presented in Fig. 1, the following algorithm is executed:

1. Select the initial state $s_j \in S$ according to the distribution π and set $x = 0$.
2. Set $x = x + 1$ and change the current state to $q_x = s_j$
3. Select the symbol $O_x \in \vartheta$ to be observed at state q_x according to the distributions in B.
4. If $(x < X)$ **go to** step 5, else **terminate**.
5. Select the state transition to be realized from the current state q_x to another state $s_j \in S$ according to the distribution A, then **go to** step 2.

3.3 Manipulation of a HMMs

Consider a sequence of symbols $O = O_1 O_2 \ldots O_X$ and a HMM $\lambda = \{A, B, \pi\}$. The probability $P(O|\lambda)$ to observe O given λ is efficiently calculated by the *Forward-Backward* algorithm [52] which runs in $\theta(X.N^2)$. Given a sequence of symbols $O = O_1 O_2 \ldots O_X$, it is possible to iteratively re-estimate the parameters of a HMM $\lambda = \{A, B, \pi\}$ in order to maximize the value of $P(O|\overline{\lambda})$, where $\overline{\lambda} = \{\overline{A}, \overline{B}, \overline{\pi}\}$ is the re-estimated model. The *Baum-Welch* algorithm [52] is generally used to perform this re-estimation. This algorithm runs in $\theta(\gamma.X.N^2)$ where γ is the user-defined maximum number of iterations. In this paper, the value $\gamma = 100$ is selected following [53]. The *Baum-Welch* algorithm can also train a HMM for multiple sequences. The algorithm maximizes the value of $P(O|\overline{\lambda}) = \sum_{k=1}^{K} P(O^{(k)}|\overline{\lambda})$ where $O = \{O^{(1)}, \ldots, O^{(K)}\}$ is a set of K sequences of symbols and $O^{(k)} = O_1^{(k)} \ldots O_{X_k}^{(k)}$ is the k^{th} sequence of symbols of O. In the case of multiple sequences, this algorithm runs in $\theta\left(\gamma.(\sum_{k=1}^{K} X_k).N^2\right)$.

3.4 HMMs similarity measure

Several measures have yet been proposed for comparing two HMMs $\lambda = \{A, B, \pi\}$ and $\lambda' = \{A', B', \pi'\}$ [54–58]. Further details related to these measures and their drawbacks are available in [53, 59]. In this paper, we selected the accurate low-complexity similarity measure between two HMMs λ and λ' originally proposed in [60] and later used in [53, 59]. In the current paper, this measure is noted $d_{sim}(\lambda, \lambda')$ and its summarized computation scheme can be found in [59]. $d_{sim}(\lambda, \lambda')$ evaluates the probability that λ and λ' observe identical sequences of symbols.

4 The proposed approach

4.1 Main idea

The main idea for comparing tree sets using HMMs in this work arises from the following observation related to the traversal of a tree using *DFS*: When *DFS* is executed on a tree t, the algorithm sequentially transits from one visited node located at a specific depth, to another visited node located at another depth, starting from the root node until the last node is visited. In other words, *DFS sequentially transits from*

$$\begin{array}{lcccc} \textbf{\textit{symbols}}: & p(y_1) & p(y_2) & \dots & p(y_k) \\ & \uparrow & \uparrow & \vdots & \uparrow \\ \textbf{\textit{states}}: & depth(y_1) & \rightarrow depth(y_2) & \rightarrow \dots \rightarrow & depth(y_k) \end{array}$$

Fig. 2 MC $\delta(t)$ of tree t considering property p

one node depth to another and at each step, the algorithm observes the properties of the visited node. This observation enables us to transform t into a MC resulting from the traversal of t using *DFS*. In this transformation, the hidden states are the depths of the visited nodes and the observed symbols are the values of a specific targeted nodes property p. If we note $depth(t) = \max\{depth(y)|y \in t\}$ the depth of the deepest node of t, then the depth of the node visited by *DFS* at each step of the traversal is always between 0 (which is the depth of the root node) and $depth(t)$. Therefore, the hidden states belong to $\{0, 1, \dots, depth(t)\}$. In practice, if $y_1 y_2 \dots y_k$ is the ordered list of visited nodes outputted by *DFS* (with $root(t) = y_1$), then Fig. 2 shows the MC named $\delta(t)$ generated for t considering the property p.

The MC of Fig. 2 is later used to initialize and to train a HMM which learns how much the nodes of t verify property p. Given that HMMs can be trained for multiple sequences, this principle can be generalized to a tree set. Consider a finite set T composed of $|T| \geq 1$ trees. We transform the trees in T into MCs, before training one HMM for T using these MCs.

4.2 Methodology

Let $T = \{t_1, \dots, t_n\}$ and $T' = \{t'_1, \dots, t'_k\}$ be two tree sets. Consider now the set $P = \{p_1, p_2, \dots, p_m\}$ composed of m user-defined targeted nodes properties. As it is shown in Fig. 3, the methodology applied in the proposed approach to compare T and T' can be summarized in the two following steps:

1. **Tree modeling:** For each property p_i $(1 \leq i \leq m)$, the principle presented in Sect. 4.1 is applied to obtain the two sets $\Delta_i(T) = \{\delta_i(t_1), \dots, \delta_i(t_n)\}$ and $\Delta_i(T') = \{\delta_i(t'_1), \dots, \delta_i(t'_k)\}$ containing the MCs associated with the trees in T and T'. Thereafter, two HMMs $\lambda_i(T)$ and $\lambda_i(T')$ are respectively initialized then trained according to the contents of $\Delta_i(T)$ and $\Delta_i(T')$. Figure 4 describes the principle of tree modeling for a tree set T.
2. **Tree comparison:** The point $\Omega_P(T, T') \in \mathbb{R}^m$ whose i^{th} component is the similarity rate between T and T' according to property p_i $(1 \leq i \leq m)$ is initially calculated. Finally, the distance $d_P(T, T')$, then the similarity rate $\varphi_P(T, T')$ between T and T' are computed based on the components of $\Omega_P(T, T')$. Figure 5 summarizes the principle of tree comparison.

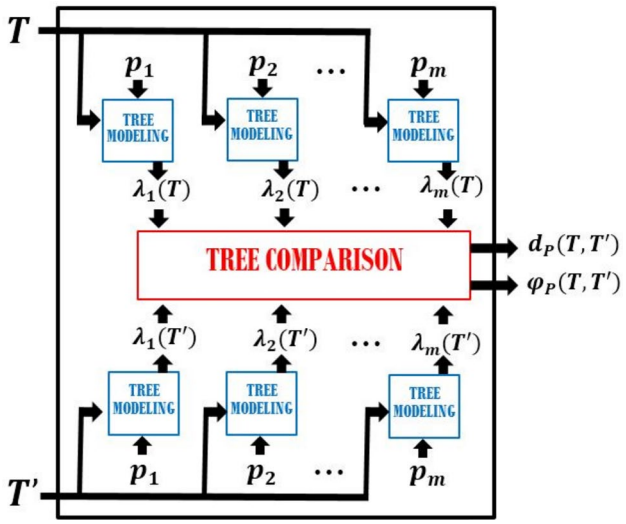


Fig. 3 Proposed methodology for comparing tree sets

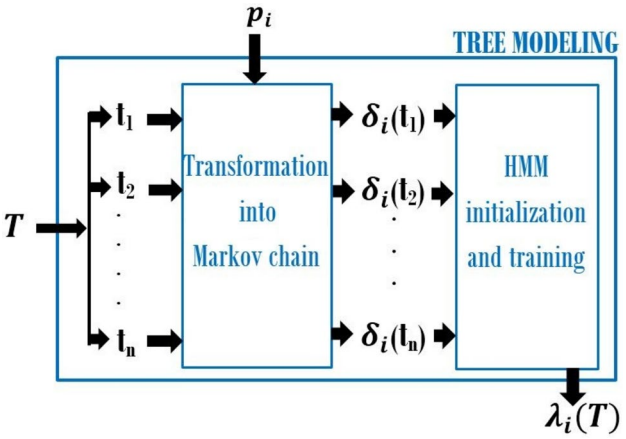


Fig. 4 The tree modeling principle

4.3 Tree modeling

4.3.1 Example of transformation into MC

Consider the singleton $T = \{t\}$ where t is the node-labeled and arc-labeled tree presented in Fig. 6a. The node labels of t belong to the set $\{1, 2, 3, 4, 5, 7\}$ and the arc labels belong to the set $\{a, b, c, e, f\}$. Given that the root of a tree has no father, we conventionally assume that if a tree is arc-weighted, then the weight of the arc linking the root to its father is 0.0. If a tree is arc-labeled, we also

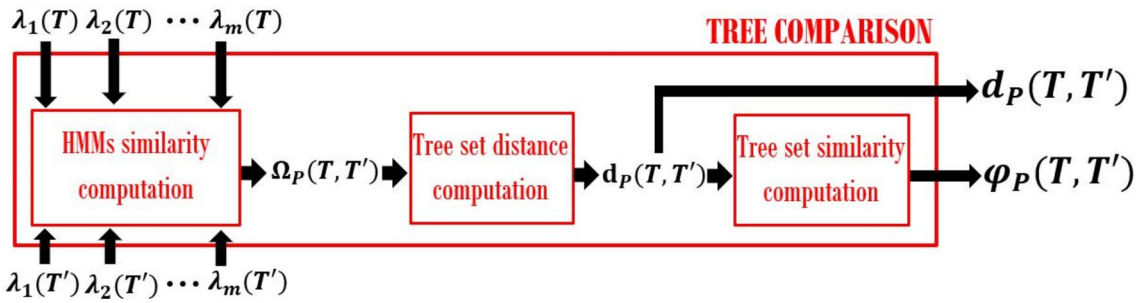


Fig. 5 The principle of tree comparison for two tree sets T and T'

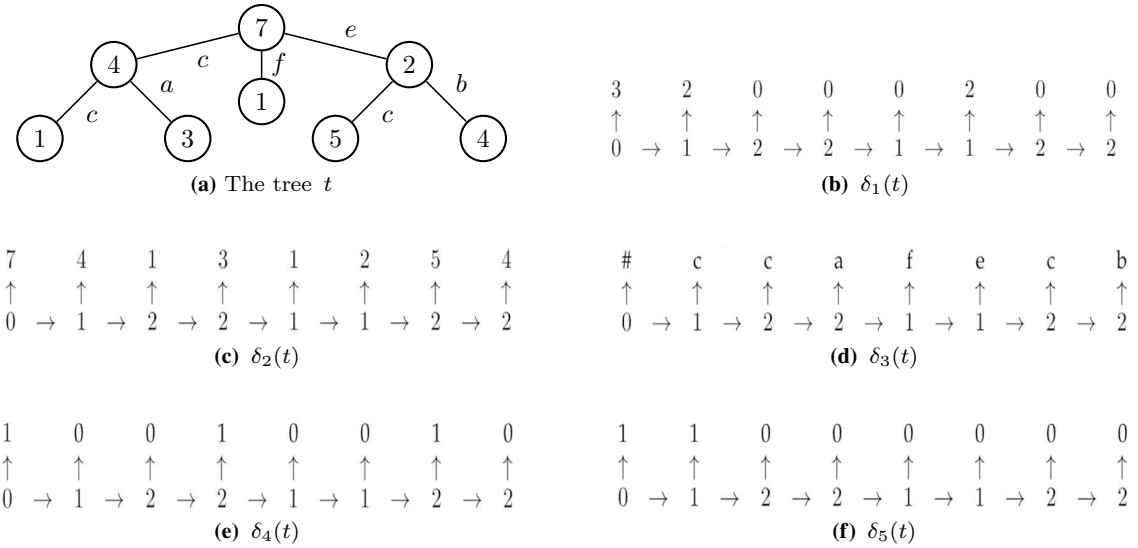


Fig. 6 Transformation of the tree set $T = \{t\}$ into a MC depending on the targeted nodes property

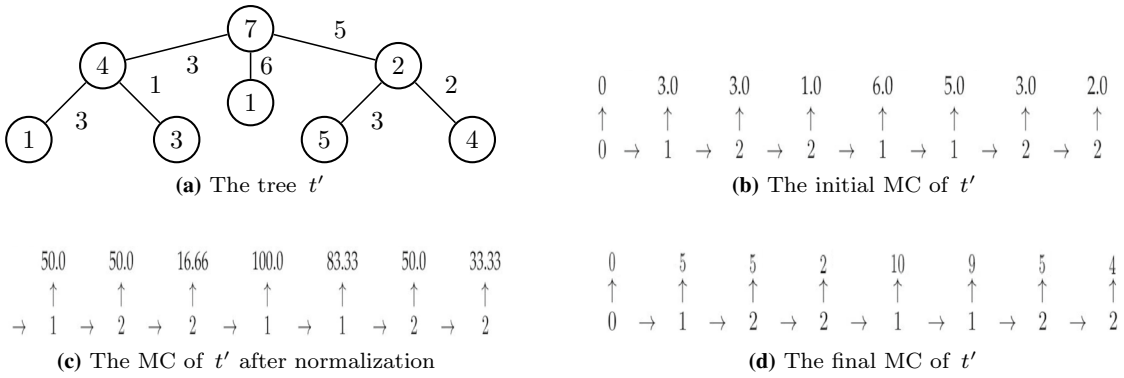
conventionally assume that the label of the root is any valid label (symbol, integer, etc.) that does not belong to the initial set of labels. In this paper, the symbol $'\#'$ is selected as conventional label for this purpose. Thus, the set of labels becomes $\{a, b, c, e, f\} \cup \{\#\}$. For each node y of t, if we consider that the targeted nodes property is:

1. $p_1(y)$: The degree of y, then the resulting MC is $\delta_1(t)$ presented in Fig. 6a.
2. $p_2(y)$: The label of y, then the resulting MC is $\delta_2(t)$ presented in Fig. 6b.
3. $p_3(y)$: The label of the arc linking y to its father, then the resulting MC is $\delta_3(t)$ shown in Fig. 6c.
4. $p_4(y)$: A Boolean which is true when the label of y is an odd number greater or equal to 3, then the resulting MC is $\delta_4(t)$ presented in Fig. 6d.
5. $p_5(y)$: A Boolean which is true when y is not a leaf and the label of y is the sum of the labels of its children, then the resulting MC is $\delta_5(t)$ presented in Fig. 6e.

Let us now explain how to handle properties whose values are positive continuous numbers like nodes or arcs weights. Indeed, for such properties, the set of possible values is infinite. This is a serious obstacle because the set ϑ of symbols of a HMM must be finite. Let us note ρ the set of nodes properties whose values are positive continuous numbers. Consider now a tree set T and a property $p \in \rho$. To overcome the former obstacle, the algorithm below composed of four main steps is applied following an analog algorithm used in [59] for a similar purpose:

Step 1: Compute the MCs of the trees in T using property p.

Step 2: Modify the MCs obtained at step 1 by normalizing the values of $p(y)$ in order the move them in the interval [0, 100]. This is done by replacing the value of $p(y)$ in each MC by $\hat{p}(y)$ calculated in Eq. 1. The effect of this normalization will be canceled when the distance will be computed.



(a) The tree t' **(b)** The initial MC of t'

(c) The MC of t' after normalization **(d)** The final MC of t'

Fig. 7 MC of the tree set $T' = \{t'\}$ for a property $p \in \rho$. Here, $p(y)$ is the weight of the arc linking y to its father

$$\hat{p}(y) = 100 \times \frac{p(y)}{\alpha(T,p)} \quad \text{where} \quad \alpha(T,p) = \max\{p(z) | \forall z \in t \text{ and } \forall t \in T\}. \tag{1}$$

Step 3: Split the interval [0, 100] into $(N+1)$ slices $\{v_0, v_1, \ldots, v_N\}$ as shown in Eq. 2, where N is a user-defined integer.

$$v_0 = \{0\} \text{ and } v_k = \left]\frac{100}{N} \times (k-1), \frac{100}{N} \times k\right], 1 \leq k \leq N. \tag{2}$$

Step 4: If the value of N used at step 3 to split the interval [0, 100] is very high, then the width of each slice v_k becomes tiny and all the elements in v_k become very near to one unique value which is $\frac{100}{N} \times k$. In that case, the elements of v_k can be approximated by this single value which we identify here by the index k of the slice v_k. This reasoning enables to define a new MC by replacing $\hat{p}(y)$ in the MC obtained at step 2 with the value $\tilde{p}(y)$ which is the index k the slice v_k containing $\hat{p}(y)$ as shown in Eq. 3.

$$\tilde{p}(0) = 0 \text{ and } (\tilde{p}(y) = k) \Leftrightarrow \left(\hat{p}(y) \in v_k\right), 1 \leq k \leq N \tag{3}$$

Following this algorithm, property p whose values are continuous numbers is finally replaced by property $\tilde{p}$ whose values belong to the finite set $\vartheta = \{0, 1, \ldots, N\}$. Consider for example the singleton $T' = \{t'\}$ containing the node-labeled and arc-weighted tree t' presented in Fig. 7a. The tree t' is quite similar to the tree t of Fig. 6a, the only difference is that the arc labels are transformed into arc weights as follows: # becomes 0.0, a becomes 1.0, b becomes 2.0, $\cdots$ and f becomes 6.0 . If we consider that for each node $y \in t'$, $p(y)$ is the weight of the arc linking y to its father, then the former algorithm is executed on T' as follows:

1. The MC of Fig. 7b is computed for t' using p.
2. The arc weights are replaced by their normalized values using $\alpha(p, T') = 6$. The resulting MC is presented in Fig. 7c.
3. If we select for this example the value $(N = 10)$, then the interval [0, 100] is split to generate the set $\{v_0, v_1, \ldots, v_{10}\}$ of slices where: $v_0 = \{0\}, v_1 =]0, 10], v_2 =]10, 20], \ldots, v_{10} =]90, 100]$.
4. Each normalized arc weight is replaced by the index of the slice to which it belongs. The resulting MC is presented in Fig. 7d.

One can remark that the choice of N is crucial in this algorithm. On the one hand, low values of N cannot enable to accurately separate the property values. But on the other hand, huge values of N induce a considerable augmentation of the HMM training time due to the elevated number of symbols. In this paper, the value $N = 50$ is selected following a recommendation made in [59].

Another important remark is that the proposed approach can be used for comparing two tree sets containing trees which are both, unlabeled and unweighted. In that case, the set P of targeted nodes properties must only contain properties related to the tree topologies like the value of the degree of each node or the parity of this degree for example.

4.3.2 HMM initialization

Consider a tree set $T = \{t_1, \ldots, t_n\}$ and a nodes property p_i. The parameters of the initial HMM $\lambda_{i_0}(T) = (A^T_{i_0}, B^T_{i_0}, \pi^T_{i_0})$ associated with T are calculated as described below to statistically capture the states transitions and the symbols probabilities distributions from the content of $\Delta_i(T) = \{\delta_i(t_1), \ldots, \delta_i(t_n)\}$:

1. Given that the states are the node depths, the number of states is $(depth(T) + 1)$ where $depth(T) = \max\{depth(t) | t \in T\}$. Therefore, the set of states is $S = \{0, 1, \ldots, depth(T)\}$.

2. The content of the set ϑ of symbols depends on the values that can be taken by the property p_i. Some particular cases are listed below:

   Case 1: If p_i is a Boolean, then $\vartheta = \{0, 1\}$.
   Case 2: If $p_i \in \rho$, the algorithm presented in Sect. 4.3.1 is applied to obtain the finite set $\vartheta = \{0, 1, \ldots, N\}$, where N is a user-defined integer.
   Case 3: If p_i is the degree of each node, then $\vartheta = \{0, 1, \ldots, deg_{max}\}$ where deg_{max} is a user-defined constant representing the maximum possible degree. In this paper, the value $deg_{max} = 30$ is selected.
   Case 4: If p_i is the value of the label of a node (resp. arc), ϑ must contain all the node (resp. arc) labels appearing in T, augmented by all the possible labels that can appear in any other tree set that can be compare to T. In other words, if we want to compare two tree sets T and T', ϑ must contain at least all the labels appearing in both, T and T'.

3. The probability of transiting from state j to state k is calculated in Eq. 4. In that equation, the expression $transit(j, k, \Delta_i(T))$ is the number of transitions from state j to state k in $\Delta_i(T)$ and the expression $transit(j, -, \Delta_i(T))$ is the number of transitions from state j to any destination in $\Delta_i(T)$.

$$A_{i_0}^T[j,k] = \frac{transit(j,k,\Delta_i(T))}{transit(j,-,\Delta_i(T)) + 1}. \tag{4}$$

4. The probability of observing symbol k at state j is calculated in Eq. 5 where $observe(k, j, \Delta_i(T))$ is the number of times where symbol k is observed at state j in $\Delta_i(T)$, and $observe(-, j, \Delta_i(T))$ is the number of occurrences of state j in $\Delta_i(T)$, whatever is the symbol observed.

$$B_{i_0}^T[j,k] = \frac{observe(k,j,\Delta_i(T))}{observe(-,j,\Delta_i(T)) + 1}. \tag{5}$$

5. The probability that a sequence starts with state j is given by Eq. 6 where $start(j, \Delta_i(T))$ is the number of elements in $\Delta_i(T)$ starting with state j.

$$\pi_{i_0}^T[j] = \frac{start(j,\Delta_i(T))}{|\Delta_i(T)| + 1}. \tag{6}$$

The parameters of $\lambda_{i_0}^T$ are not probability distributions due to the constant 1.0 added to the denominator of their components. This value is intentionally introduced to avoid eventual divisions by zero in $A_{i_0}^T$ and $B_{i_0}^T$. Zero probabilities are also avoided in $\pi_{i_0}^T$. The model $\lambda_{i_0}^T$ is then readjusted by equitably redistributing the missing quantity in each line to obtain the model $\lambda_{i_1}(T) = (A_{i_1}^T, B_{i_1}^T, \pi_{i_1}^T)$ as follows:

1. $A_{i_1}^T[j,k] = A_{i_0}^T[j,k] + \frac{1}{|S|}\left(1 - \sum_{l=0}^{|S|-1} A_{i_0}^T[j,l]\right)$
2. $B_{i_1}^T[j,k] = B_{i_0}^T[j,k] + \frac{1}{|\vartheta|}\left(1 - \sum_{l=0}^{|\vartheta|-1} B_{i_0}^T[j,l]\right)$
3. $\pi_{i_1}^T[j] = \pi_{i_0}^T[j] + \frac{1}{|S|}\left(1 - \sum_{l=0}^{|S|-1} \pi_i^0[l]\right)$

4.3.3 HMM training phase

If $|T| = 1$, then the readjusted initial HMM $\lambda_{i_1}(T)$ is trained with the *Baum-Welch* algorithm for one single sequence, otherwise $\lambda_{i_1}(T)$ is trained by the same algorithm for multiple sequences. In both situations, the resulting HMM is $\lambda_i(T) = (A_i^T, B_i^T, \pi_i^T)$ and the training sequences are the sequences of symbols appearing in $\Delta_i(T)$. For example, the initial HMMs associated with the MCs presented in Fig. 6a and b will respectively be trained with the following sequences of symbols: 3, 2, 0, 0, 0, 2, 0, 0, 0 and 7, 4, 1, 3, 1, 2, 5, 4.

4.4 Tree comparison

Let T and T' be two finite tree sets, and consider a finite set P of nodes properties. In order to compare T and T' according to the properties in P, the similarity rate $\omega_i(T, T')$ between T and T' according to each property $p_i \in P$ ($1 \leq i \leq |P|$) is initially computed. The $|P|$ resulting similarity rates are then saved as the coordinates of a point $\Omega_P(T, T')$ of $\mathbb{R}^{|P|}$. This point will be later used to derive the distance and the similarity rate between T and T'.

4.4.1 Models similarity computation

In this work, $\omega_i(T, T')$ is obtained by calculating the value of $d_{sim}\left(\lambda_i(T), \lambda_i(T')\right)$. Unlike most of the existing measures which are based on the comparison of the exact visual content of the two trees, the measure proposed in this paper is finer because d_{sim} rather evaluates the probability that $\lambda_i(T)$ and $\lambda_i(T')$ observe an identical sequence of symbols. In our context, this means that d_{sim} evaluates the probability that $\lambda_i(T)$ and $\lambda_i(T')$ generate new trees with identical values of $p_i(y)$ for every node y. Therefore, the trees of two tree sets may not be too much similar visually, but their similarity rate can still be high. This is possible because even when two HMMs look ostensibly very different, their statistical equivalence can still occur [52] [1].

Given that the values of $p_i(y)$ are normalized when $p_i \in \rho$, the similarity rate must in that case be weighted by a coefficient $\theta_i(T, T') \leq 1$ that will cancel the effects of normalization. Such a coefficient was also used in [59] to cancel the effects of histogram normalization. Equation 7 shows how to compute this coefficient and Eq. 8 describes how to compute

[1] See page 15, Section F

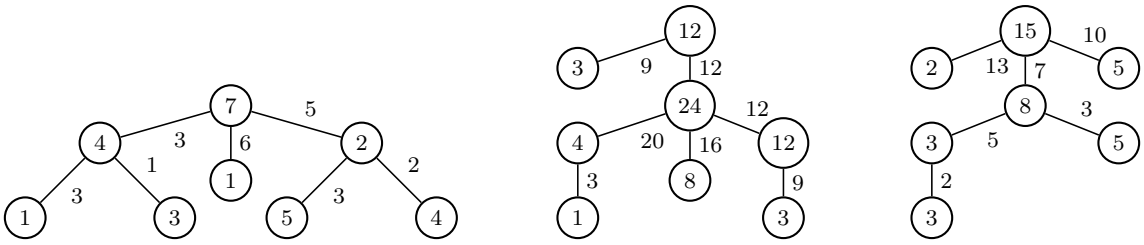


Fig. 8 The node-labeled and arc-weighted trees t_1, t_2 and t_3

$\omega_i(T,T')$. Finally, the point $\Omega_P(T,T')$ whose ith component is $\omega_i(T,T')$ is generated as shown in Eq. 9.

$$\theta_i(T,T') = \frac{\min\left(\alpha(T,p_i),\alpha(T',p_i)\right)}{\max\left(\alpha(T,p_i),\alpha(T',p_i)\right)} \text{if } (p_i \in \rho)$$
$$\theta_i(T,T') = 1 \text{ otherwise,} \tag{7}$$

$$\omega_i(T,T') = 100 \times \theta_i(T,T') \times d_{sim}\left(\lambda_i(T),\lambda_i(T')\right)(in\%), \tag{8}$$

$$\Omega_P(T,T') = [\omega_1(T,T'),\omega_2(T,T'),\ldots,\omega_{|P|}(T,T')]. \tag{9}$$

4.4.2 The proposed distance between tree sets

The $|P|$ components of $\Omega_P(T,T')$ are percentages. Consequently, their maximum value is 100%. This means that the more the components of $\Omega_P(T,T')$ are distant from 100%, the more T and T' are distant. This enables to calculate the distance $d_P(T,T')$ between T and T' according to the properties in P by computing any valid distance between the point $\Omega_P(T,T')$ and the point of $\mathbb{R}^{|P|}$ whose components are all equal to 100. In this paper, the *Euclidean* and the *Manhattan* distances whose formulas are presented in Eq. 10 have been selected for this purpose. In unreported simulations, the *Tchebychev* distance was also selected, but we finally excluded it because it exhibited extremely poor classification results (Cf. Section 5.2).

$$d_P(T,T') = \sqrt{\sum_{i=1}^{|P|}(100-\omega_i(T,T'))^2} \quad (Euclidean)$$
$$d_P(T,T') = \sum_{i=1}^{|P|}|100-\omega_i(T,T')| \quad (Manhattan). \tag{10}$$

4.4.3 The proposed similarity between tree sets

If T and T' are strongly similar (i.e: each component of $\Omega_P(T,T')$ is near to 100), the distance $d_P(T,T') \approx 0$ irrespective of the selected distance between two points in $\mathbb{R}^{|P|}$. In a similar way, when T and T' are strongly distant (i.e: each component of $\Omega_P(T,T')$ is near to 0), the upper-bound $\tilde{d}_P$ of the distance $d_P(T,T')$ presented in Eq. 11 is obtained depending on the selected distance between two points in $\mathbb{R}^{|P|}$.

$$\tilde{d}_P = 100\sqrt{|P|} \quad (Euclidean)$$
$$\tilde{d}_P = 100.|P| \quad (Manhattan) \tag{11}$$

The existence of this upper-bound is an asset because if we divide $d_P(T,T')$ by $\tilde{d}_P$ as shown in Eq. 12, we obtain the dissimilarity rate $\overline{\varphi}_P(T,T')$ between T and T' according to the properties in P. The values computed in Eq. 12 are in (%). This finally enables to compute the similarity rate $\varphi_P(T,T')$ between T and T' according to the properties in P in Eq. 13.

$$\overline{\varphi}_P(T,T') = 100 \times \left(\frac{d_P(T,T')}{100\sqrt{|P|}}\right) = \frac{d_P(T,T')}{\sqrt{|P|}} \quad (Euclidean)$$
$$\overline{\varphi}_P(T,T') = 100 \times \left(\frac{d_P(T,T')}{100.|P|}\right) = \frac{d_P(T,T')}{|P|} \quad (Manhattan), \tag{12}$$

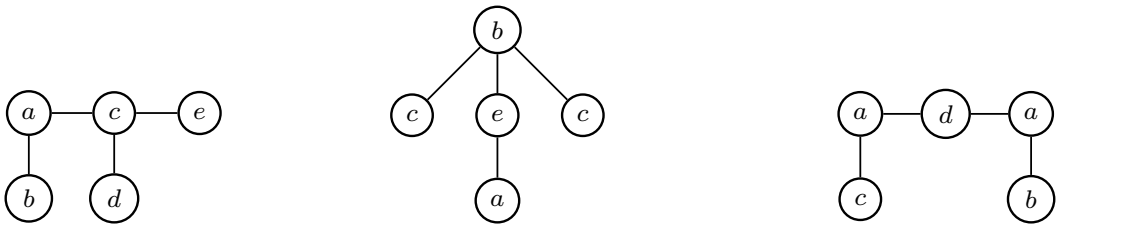
$$\varphi_P(T,T') = 100 - \overline{\varphi}_P(T,T') \quad (in\%). \tag{13}$$

4.5 Example of tree sets comparison

Consider the three singletons $T_1 = \{t_1\}$, $T_2 = \{t_2\}$ and $T_3 = \{t_3\}$ where t_1, t_2, t_3 are the node-labeled and arc-weighted trees respectively presented in Fig. 8a–c. We have applied the proposed measures to compare the trees in the pairs (T_1,T_2), (T_1,T_3) and (T_2,T_3). The selected set of targeted nodes properties is $P = \{p_1,p_2,\ldots,p_6\}$ where for each node y:

Table 2 Comparison of the pairs (T_1, T_2), (T_1, T_3) and (T_2, T_3)

T	T'	$\Omega_P(T,T')$						*Euclidean*		*Mahattan*	
		p_1	p_2	p_3	p_4	p_5	p_6	$d_P(T,T')$	$\varphi_P(T,T')$ in %	$d_P(T,T')$	$\varphi_P(T,T')$ in %
T_1	T_2	58.33	75.78	8.75	66.67	94.90	83.33	109.84	55.16	212.23	64.63
T_1	T_3	65.0	58.33	13.46	66.67	93.42	75.0	110.59	54.85	228.12	61.98
T_2	T_3	75.0	73.58	40.63	92.85	86.21	66.97	78.62	67.90	164.77	72.54



(a) t_1: Unrooted ordered **(b)** t_2: Rooted unordered **(c)** t_3: Unrooted unordered

Fig. 9 Examples of trees unrooted and unordered trees

1. $p_1(y)$ is the degree of y.
2. $p_2(y)$ is the label of y, the set of possible labels being $\{1, 2, 3, 4, 5, 7, 8, 12, 15, 24\}$
3. $p_3(y)$ is the weight of the arc linking y to its father.
4. $p_4(y) = 1$ if y is not a leaf and the label of y is the sum of the labels of its children, else $p_4(y) = 0$.
5. $p_5(y) = 1$ if the label of y is odd, else $p_5(y) = 0$.
6. $p_6(y) = 1$ if y is not a leaf and the label of y is strictly greater than the labels of its children, else $p_6(y) = 0$.

All the experiments realized in this paper have been executed on a personal computer with the following properties: (1) Processors: *Intel(R) Core(TM) i7-8665U CPU @1.9GHz 2.11GHz* (2) RAM: *16 GB*. After executing the proposed approach on the pairs (T_1, T_2), (T_1, T_3) and (T_2, T_3), we have obtained the results presented in Table 2. These results reveal that the highest similarity rate is always obtained for the pair (T_2, T_3), irrespective of the selected distance in $\mathbb{R}^6$. For each pair of tree sets, Table 2 gives details related to the similarity rates regarding each property. These results enable to make several remarks. One can for example remark that the pair (T_2, T_3) has the highest similarity rate regarding the topology (property p_1). This remark seems to be accurate when we observe the high similarity between the visual shapes of the trees in T_2 and T_3. Table 2 also enables to remark the very low similarity rates 8.75% and 13.46% respectively obtained for the pairs (T_1, T_2) and (T_1, T_3) regarding the arc weights (property p_3), these values are far from the 40.63% obtained for the pair (T_2, T_3). This other remark also seems to be accurate because all the arc weights in T_1 are low while those in T_2 and T_3 are mostly high. Other remarks regarding the remaining properties can be made analogically. The global processes of tree modeling and tree comparison respectively took around 187 milliseconds and 20 ms for the three pairs on the experimental computer.

4.6 Generalization to unrooted and unordered trees

The proposed measures can also be used to compare tree sets containing unrooted trees and unordered trees. This is possible by initially transforming every unrooted or unordered tree t into a set of rooted ordered trees, while preserving the nodes and the arcs existing in t. Proceeding this way, all the information embedded in t is transferred in the resulting tree set. The node-labeled trees t_1, t_2 and t_3 respectively presented in Figs. 9a–c will serve as examples in this section. Consider a finite tree set T. The following transformations must be applied to every tree $t \in T$ before to compare T to any other tree set:

1. If t is already a rooted ordered tree, no transformation is applied to t.
2. If t is ***unrooted ordered***, then t is transformed into the tree set $f(t)$ containing the $|t|$ possible rooted ordered trees that can be derived from t when each node is considered as the root, while preserving the left-to-right order existing between the nodes of t in the resulting trees. When this principle is applied to the unrooted ordered tree t_1 of Fig. 9a, the set $f(t_1) = \{t_{11}, \ldots, t_{15}\}$ whose content is depicted in Fig. 10a–e is obtained.

Table 3 Comparison of the pairs $(\{t_1\}, \{t_2\})$, $(\{t_1\}, \{t_3\})$ and $(\{t_2\}, \{t_3\})$ containing the unrooted and unordered trees of Fig. 9a–c

T	T'	$\Omega_P(T,T')$		*Euclidean*		*Mahattan*	
		p_1	p_2	$d_P(T,T')$	$\varphi_P(T,T')$ in %	$d_P(T,T')$	$\varphi_P(T,T')$ in %
$\{t_1\} \rightarrow f(t_1)$	$\{t_2\} \rightarrow g(t_2)$	68.06	50.59	58.84	58.40	81.35	59.32
$\{t_1\} \rightarrow f(t_1)$	$\{t_3\} \rightarrow h(t_3)$	63.96	74.89	43.93	68.94	61.15	69.42
$\{t_2\} \rightarrow g(t_2)$	$\{t_3\} \rightarrow h(t_3)$	78.03	69.24	37.80	73.27	52.73	73.64

Each sigleton is transformed into a tree set of rooted ordered trees before the comparison

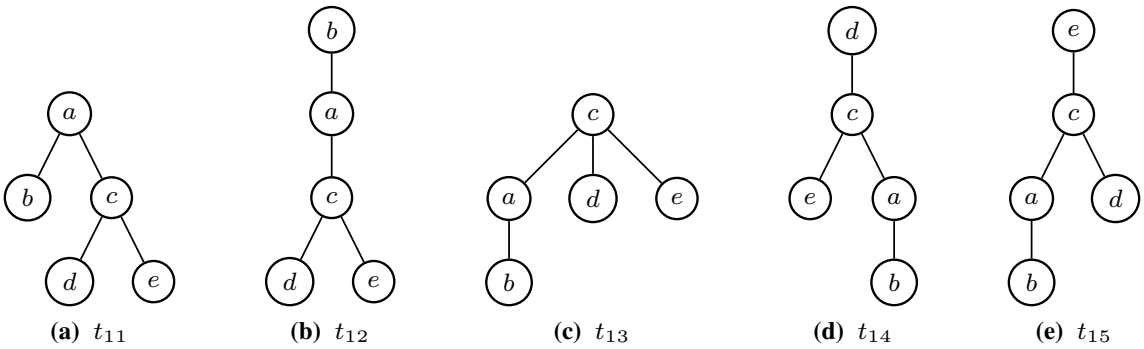
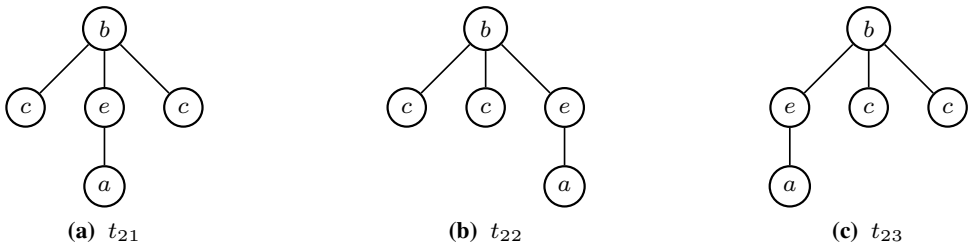


Fig. 10 The set $f(t_1) = \{t_{11}, \ldots, t_{15}\}$ derived from the unrooted ordered tree t_1 of Fig. 9a

Fig. 11 The set $g(t_2) = \{t_{21}, t_{22}, t_{23}\}$ derived from the rooted unordered tree t_2 of Fig. 9b



3. If t is ***rooted unordered***, then t is transformed into the tree set $g(t)$ containing all the rooted ordered trees that can be derived from t by realizing all the possible permutations in t. Therefore, the size of $g(t)$ is bounded by the highest possible number of permutations in t which is $(|t| - 1)!$ When this principle is applied to the rooted unordered tree t_2 of Fig. 9b, the set $g(t_2) = \{t_{21}, t_{22}, t_{23}\}$ whose content is shown in Figs. 11a–c is obtained.
4. If t is ***unrooted unordered***, then t is first transformed into a temporary tree set $tmp(t)$ containing the $|t|$ possible rooted unordered trees that can be derived from t when each node is considered as the root. Each tree $\bar{t} \in tmp(t)$ is then transformed into the tree set $g(\bar{t})$ containing all the rooted ordered trees that can be derived from $\bar{t}$ by realizing all the possible permutations in $\bar{t}$. The final set $h(t) = \bigcup_{\bar{t}} g(\bar{t})$ of rooted ordered trees derived from t is obtained by gathering all the tree sets obtained for every $\bar{t}$. The size of $h(t)$ is consequently bounded by $|t|.(|t| - 1)! = |t|!$ When this principle is applied to the unrooted unordered tree t_3 of Fig. 9c, the set $h(t_3) = \{t_{31}, \ldots, t_{38}\}$ whose content is presented in Fig. 12a–h is obtained.

The fact that the size of the tree set obtained after transforming an unordered tree is exponential is not surprising because the NP-completeness and the NP-hardness of the problem of comparing unordered trees has been thoroughly demonstrated [6, 8, 26, 30, 49]. This exponential size highly increases the HMMs training time cost. Nevertheless, the duration of the model training phase can be considerably reduced in that case if a parallel version of the *Baum-Welch* algorithm is used to train the models.

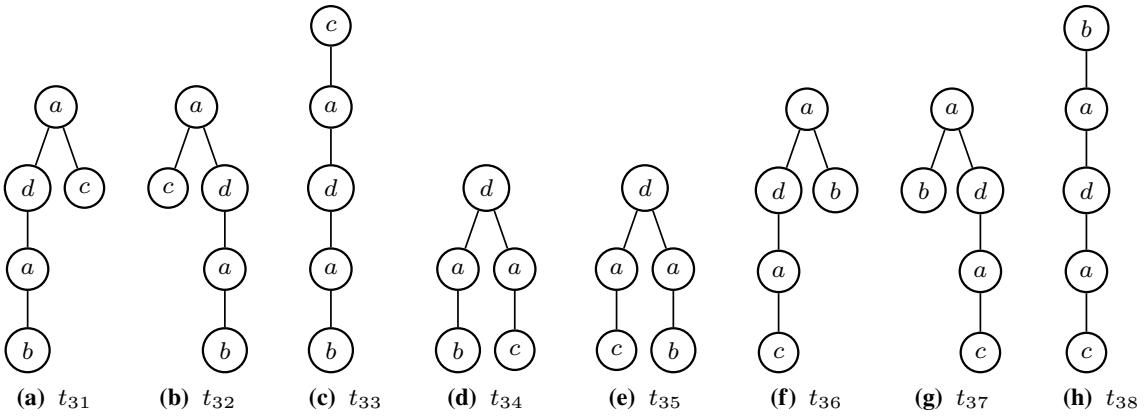


Fig. 12 The set $h(t_3) = \{t_{31}, \ldots, t_{38}\}$ derived from the unrooted unordered tree t_3 of Fig. 9c

Details related to this issue are provided at the end of this paper.

We have used the aforementioned principle to compare the trees in the pairs $(\{t_1\}, \{t_2\})$, $(\{t_1\}, \{t_3\})$ and $(\{t_2\}, \{t_3\})$ containing the unrooted and unordered trees of Fig. 9a–c. The set of targeted nodes properties is limited here to $P = \{p_1, p_2\}$ where for each node y:

1. $p_1(y)$ is the degree of y.
2. $p_2(y)$ is the label of y, the set of labels being $\{a, b, c, d, e\}$

The results of the comparison presented in Table 3 reveal that t_2 and t_3 are the most similar trees, irrespective of the selected distance in $\mathbb{R}^2$. The overall tree modeling process for these three pairs took around 1.5 seconds and the computation of the similarities between these same pairs took around 17 milliseconds.

The former example demonstrates that the proposed measures are highly flexible compared to existing measures. Indeed, unlike existing measures which can only compare two trees of the same category (i.e: two rooted ordered trees, two rooted unordered trees, etc.), the measures proposed in this paper can compare two tree sets, each containing a mixture of trees belonging to various categories. Furthermore, the proposed measures can handle trees where each node and each arc has many attributes (labels, weights). In that case, the user must only define the suitable properties targeting the various attributes to perform the comparison.

4.7 The case of the empty set

φ_P and d_P are applicable to finite tree sets. Therefore, their behavior when they are applied to the empty set must be described. It is in this perspective that the two following conventional properties are adopted here:

Property 1 *Given that* $\emptyset$ *is completely similar to itself, we assume that for every set P of nodes properties we have:* $\varphi_P(\emptyset, \emptyset) = 100\%$ *which implies that* $d_P(\emptyset, \emptyset) = 0$.

Property 2 *For every tree set* $T \neq \emptyset$ *and for every set P of nodes properties, given that* $\emptyset$ *is completely different from T, we assume that :* $\varphi_P(T, \emptyset) = \varphi_P(\emptyset, T) = 0\%$ *which implies that* $d_P(T, \emptyset) = d_P(\emptyset, T) = \tilde{d}_P$.

5 Experimental results

5.1 Comparison of sets of text documents

A text document is a sequence of symbols. Several tables enable to match a symbol with a positive numeric code. The most popular tables used for the English language are the *ASCII* and *EBCDIC* tables [2]. The *ASCII* table is preferred in this paper. Text documents comparison has been an interesting axis of research since many years and an overview of some relevant existing methods to perform this task is available in [61]. Consider two text documents d_1 and d_2. The most widespread principle used in existing approaches consists firstly in removing the 'stop words' appearing in d_1 and d_2, these are words which are presumed to be not informative as to the meaning of the documents [62]. The result of the deletion of *stop words* is the list of all the remaining words appearing in both documents. A vector $\overrightarrow{d_1}$ (resp. $\overrightarrow{d_2}$) is then used to save the number of times each word of the list appears in d_1 (resp. d_2). The final documents comparison is performed by comparing $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ using existing similarity measures between vectors. The two following limitations of the techniques based on this principle have been stated in

[2] http://www.simotime.com/asc2ebc1.htm.

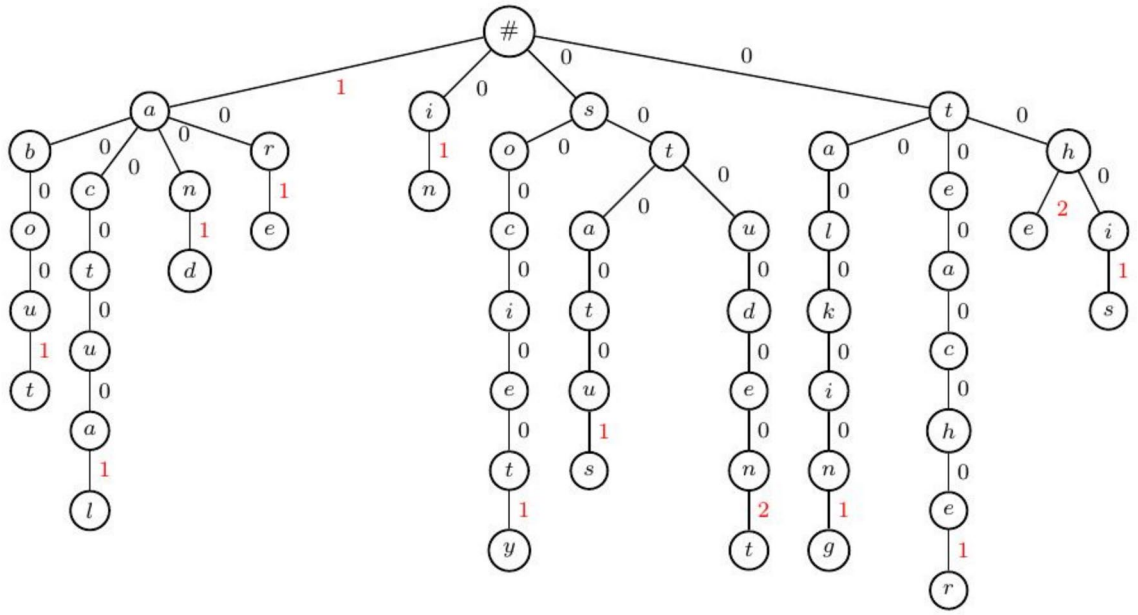


Fig. 13 The node-labeled and arc-weighted prefix tree t_d associated with the text document d whose content is: "this student and a teacher are talking about the student status in the actual society"

[59] (1) the words appearing in d_1 and d_2 are both required for the construction of each vector, (2) only two single text documents can be compared. The authors of [59] solved these problems by proposing a histogram-based approach for comparing two sets of text documents using HMMs. The main drawback of the technique proposed in [59] is that each word is considered as an isolated word which has no link with any other word in the document. However, many words in a text document can be linked in various ways, they can for example share the same prefix.

The goal of this section is to overcome this limitation of [59] by performing the comparison while considering common prefixes. More precisely, the proposed measures are adopted here to compare two sets D and D' of text documents. Each document in D and D' is initially transformed into a prefix tree (described in the next section), after what the comparison is performed.

5.1.1 Transformation of a document into a prefix tree

Consider a text document d composed of many words, each word being a sequence of characters. We propose to save all the words appearing in d as a particular node-labeled and arc-weighted tree t_d. The node labels in t_d are characters and the root node is assigned the special label '#'. The children of each node in t_d are ordered from left to right in increasing order of the ASCII codes associated with their labels. Given a node y, when we concatenate the node labels found in the path starting from $root(t_d)$ to y, we obtain the word $w(y)$ associated with y, the symbol '#' being ignored in $w(y)$. The tree t_d is constructed in such a way that for every child z of a node y, the word $w(z)$ is obtained by concatenating $w(y)$ and the label of z, i.e: $w(y)$ is a prefix of $w(z)$. The weight of the arc linking a node y to its father is the number of occurrences of $w(y)$ in d. Consider for example the English document d having the following content: "this student and a teacher are talking about the student status in the actual society". When the above principle is applied on d, the tree t_d depicted in Fig. 13 is obtain.

5.1.2 Distance and similarity computation

Consider two finite sets $D = \{d_1, \ldots, d_n\}$ and $D' = \{d'_1, \ldots, d'_k\}$ of text documents that we want to compare according to the set $P = \{p_1, \ldots, p_m\}$ of nodes properties. Each document in D and D' is first transformed into a prefix tree as described in Sect. 5.1.1. When this is done, the sets of prefix trees $T_D = \{t_{d_1}, \ldots, t_{d_n}\}$ and $T_{D'} = \{t_{d'_1}, \ldots, t_{d'_k}\}$ are obtained. Finally, Eqs. 14 and 15 are respectively used to compute the distance and the similarity between D and D'.

$$\beta_P(D, D') = d_P(T_D, T_{D'}), \tag{14}$$

$$\chi_P(D, D') = \varphi_P(T_D, T_{D'}) \quad (in\%), \tag{15}$$

Table 4 Results of the comparison of the text documents d and d'

T	T'	$\Omega_P(T,T')$			*Euclidean*		*Mahattan*	
		p_1	p_2	p_3	$\beta_P(T,T')$	$\chi_P(T,T')$ in %	$\beta_P(T,T')$	$\chi_P(T,T')$ in %
$\{d\}$	$\{d'\}$	53.42	58.94	56.36	75.89	56.18	131.28	56.24

5.1.3 Example of text documents comparison

Consider the two following text documents d and d' that were also compared in [59]:

1. ***d***: "yesterday morning, paul went to school after eating because his school is away from home. at the end of the day, paul left school and went home"
2. ***d'***: "paul went to school yesterday after eating in the morning, this is because his school is away from his house. he went back to his house after classes."

In this example, *stop words* have not been deleted following the experience realized in [59]. The comparison is performed according to the set $P = \{p_1, p_2, p_3\}$ of targeted nodes properties, where for each node y:

1. $p_1(y)$ is the degree of y.
2. $p_2(y)$ is the label of y, with $p_2(y) \in \{\#, a, b, \ldots, z\}$.
3. $p_3(y)$ is the weight of the arc linking y to its father.

The comparison results presented in Table 4 reveal that the proposed measures perform a finer analysis of text documents than the technique proposed in [59]. Indeed, when these two documents were compared in [59], the low similarity rate 39.63% was obtained because the common prefixes were not considered. However, many words in d and d' share common prefixes. When these common prefixes are now considered, the similarity rate between d and d' increases to values greater than 56%. The HMMs training took 700 milliseconds and the computation of the similarity took 13 milliseconds.

5.2 Nearest Neighbor classification

The goal of this section is to demonstrate the accuracy of the proposed measures when the suitable set of targeted nodes properties is selected. This can be achieved by evaluating the performances of the proposed tree distance when it is used by a metric-dependent classifier like the *NN*. Such an evaluation has already been done for the tree distances based on the edit operations [63].

5.2.1 Experimental databases

We have constructed two online available synthetic tree databases respectively named *FirstLast-L* and *FirstLast-LW*. [3] Each database contains four classes composed of 100 rooted ordered trees. *FirstLast-L* contains node-labeled trees and the set of node labels is $\{1, 2, \ldots, 26\}$. *FirstLast-LW* contains the same trees found in *FirstLast-L*, with weighted arcs. The peculiarity of these databases is that they contain trees which are neither characterized by the exact values of the degrees/labels of the nodes, nor by the exact values of the weights of the arcs. Rather, they are characterized by specific and non-trivial properties verified by the degrees/labels of the nodes, as well as the weights of the arcs. These properties are presented in Table 5. Beside the properties specified in that table, every tree t in these two databases verifies $(0 < depth(t) \leq 4)$. Figure 14a–fig14d depict examples of node-labeled and arc-weighted trees verifying the properties presented in Table 5 for *FirstLast-LW*. The corresponding trees verifying the properties presented in Table 5 for *FirstLast-L* are obtained by ignoring the arc weights.

5.2.2 Targeted nodes properties

We realized two experiments. During the first one, the specific properties verified by each node in *FirstLast-L* and *FirstLast-LW* are intentionally ignored as if they were unknown. Therefore, the default sets of targeted nodes properties $\overline{P}_L = \{\overline{p}_1, \overline{p}_2\}$ and $\overline{P}_{LW} = \{\overline{p}_1, \overline{p}_2, \overline{p}_3\}$ have been selected respectively for *FirstLast-L* and for *FirstLast-LW* where for each node y:

1. $\overline{p}_1(y)$ is the degree of y.
2. $\overline{p}_2(y)$ is the label of y, the set of labels being $\{1, \ldots, 26\}$.
3. $\overline{p}_3(y)$ is the weight of the arc linking y to its father.

Given that the trees in *FirstLast-L* and *FirstLast-LW* are characterized by non-trivial properties, one can obviously conjecture that the first experience will show poor classification results.

During the second experience, the specific properties verified by each node in *FirstLast-L* and *FirstLast-LW* are considered. The content of Table 5 has therefore been used for deriving the sets $P_L = \{p_1, \ldots, p_8\}$ and $P_{LW} = \{p_1, \ldots, p_{12}\}$ of targeted nodes properties respectively for *FirstLast-L* and

[3] http://perso-etis.ensea.fr/sylvain.iloga/FirstLast/index.html.

Table 5 Specific properties verified by each tree in *FirstLast-L* and *FirstLast-LW*

Class name	Topology in *FirstLast-L* and *FirstLast-LW*	Labels in *FirstLast-L* and *FirstLast-LW*	Weights in *FirstLast-LW*
First	Every non-leaf node has at least three children among which only the first child is not a leaf	The parity of the label of each non-leaf node is identical to the parity of the label of its first child, but different from the parity of the labels of its other children	The weight of the arc linking each node y to its father is $\left\lceil \frac{1}{2} \times (l(f(y)) + l(y)) \right\rceil$
Last	Every non-leaf node has at least three children among which only the last child is not a leaf	The parity of the label of each non-leaf node is identical to the parity of the label of its last child, but different from the parity of the labels of its other children	The weight of the arc linking each node y to its father is $\left\lceil \sqrt{l(f(y)) \times l(y)} \right\rceil$
Both	Every non-leaf node has at least three children among which only the first and the last children are not leaves	The parity of the label of each non-leaf node is identical to the parity of the labels of its first and last children, but different from the parity of the labels of its other children	The weight of the arc linking each node y to its father is $\lvert l(f(y)) - l(y) \rvert + 1$
None	Every non-leaf node has at least three children among which only the first and the last children are leaves	The parity of the label of each non-leaf node is different from the parity of the labels of its first and last children, but identical to the parity of the labels of its other children	The weight of the arc linking each node y to its father is $\left\lceil \frac{\max(l(f(y)), l(y))}{\min(l(f(y)), l(y))} \right\rceil$

The expressions $l(y)$ and $f(y)$ respectively refer to the label and to the father of a node y

FirstLast-LW. For each node y, if $l(y)$ and $f(y)$ respectively refer to the label of y and to the the label of its father, then:

1. $p_1(y) = 1$ if y has at least three children among which only the first child is not a leaf, else $p_1(y) = 0$.
2. $p_2(y) = 1$ if y has at least three children among which only the last child is not a leaf, else $p_2(y) = 0$.
3. $p_3(y) = 1$ if y has at least three children among which only the first and the last children are not leaves, else $p_3(y) = 0$
4. $p_4(y) = 1$ if y has at least three children among which only the first and the last children are leaves, else $p_4(y) = 0$
5. $p_5(y) = 1$ if y has at least three children and the parity of $l(y)$ is identical to the parity of the label of its first child, but different from the parity of the labels of its other children, else $p_5(y) = 0$.
6. $p_6(y) = 1$ if y has at least three children and the parity of $l(y)$ is identical to the parity of the label of its last child, but different from the parity of the labels of its other children, else $p_6(y) = 0$.
7. $p_7(y) = 1$ if y has at least three children and the parity of $l(y)$ is identical to the parity of the labels of its first and last children, but different from the parity of the labels of its other children, else $p_7(y) = 0$
8. $p_8(y) = 1$ if y has at least three children and the parity of $l(y)$ is different from the parity of the labels of its first and last children, but identical to the parity of the labels of its other children, else $p_8(y) = 0$
9. $p_9(y) = 1$ if the weight of the arc linking y to its father is $\left\lceil \frac{1}{2} \times (l(f(y)) + l(y)) \right\rceil$, else $p_9(y) = 0$.
10. $p_{10}(y) = 1$ if the weight of the arc linking y to its father is $\left\lceil \sqrt{l(f(y)) \times l(y)} \right\rceil$, else $p_{10}(y) = 0$.
11. $p_{11}(y) = 1$ if the weight of the arc linking y to its father is $|l(f(y)) - l(y)| + 1$, else $p_{11}(y) = 0$.
12. $p_{12}(y) = 1$ if the weight of the arc linking y to its father is $\left\lceil \frac{\max(l(f(y)), l(y))}{\min(l(f(y)), l(y))} \right\rceil$, else $p_{12}(y) = 0$.

5.2.3 Classification results

A ten-folds cross-validation has been realized during the classification phase where each tree t of the experimental databases was considered as the singleton $\{t\}$. For each database, the training and testing sets of each fold were obtained after randomly shuffling the trees of each class. These training and testing sets have also been made available online [3]. For each of the two experiences, the *NN* classifier has been executed using the proposed tree distance as metric.

In the conditions of the first experience where the specific nodes properties have been ignored, the *Manhattan* distance exhibited the best accuracies of 44.25% and 36% respectively

Table 6 Confusion matrix of the *Nearest Neighbor* classifier in *FirstLast-L* when the proposed tree distance is used as tree metric and when the specific nodes properties are ignored

(a) *Euclidean*					(b) *Manhattan*				
	First	*Last*	*Both*	*None*		*First*	*Last*	*Both*	*None*
First	24	7	66	3	*First*	24	4	69	3
Last	28	35	14	23	*Last*	25	39	17	19
Both	30	0	41	29	*Both*	22	0	62	16
None	24	1	34	41	*None*	21	0	24	55

The accuracies are 35.25% and 44.25% when the *Euclidean* and the *Manhattan* distances are respectively selected in $\mathbb{R}^3$

Table 7 Confusion matrix of the *Nearest Neighbor* classifier in *FirstLast-LW* when the proposed tree distance is used as tree metric and when the specific nodes properties are ignored

(a) *Euclidean*					(b) *Manhattan*				
	First	*Last*	*Both*	*None*		*First*	*Last*	*Both*	*None*
First	9	9	81	1	*First*	8	11	80	1
Last	20	41	33	6	*Last*	17	41	34	8
Both	29	2	59	10	*Both*	23	2	66	9
None	6	7	57	30	*None*	8	6	57	29

The accuracies are 34.75% and 36% when the *Euclidean* and the *Manhattan* distances are respectively selected in $\mathbb{R}^3$

for *FirstLast-L* and *FirstLast-LW*. Tables 6 and 7 respectively present the confusion matrices for *FirstLast-L* and *FirstLast-LW*. These poor performances confirm the former conjecture.

In the conditions of the second experience where the specific nodes properties have been considered, the *Euclidean* and the *Manhattan* distances both exhibited a perfect accuracy of 100% for the two experimental databases. These perfect performances demonstrate the power of the proposed tree distance when the suitable set of targeted nodes properties is selected.

5.2.4 Comparison with the tree Edit distance

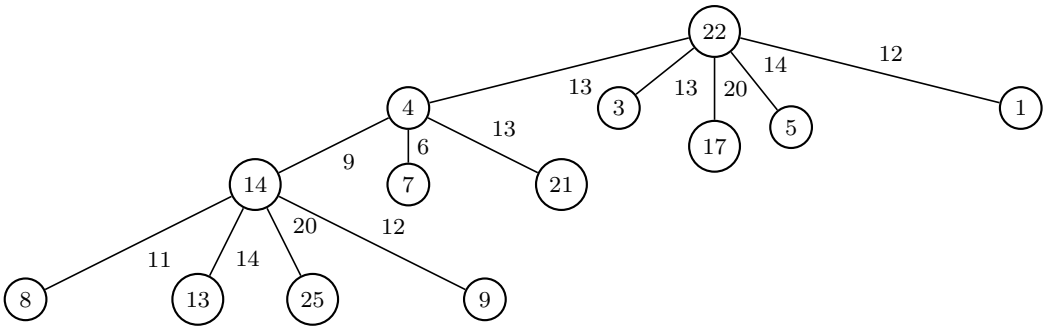
We have compared the performance of the proposed tree distance with the performance of the widespread *tree Edit* distance when used as metrics for the *NN* classifier on *FirstLast-L*. No comparison was realized for *FirstLast-LW* because existing tree distances (including the *tree Edit* distance) are not applicable to arc-weighted trees. To achieve our goal, we have first downloaded a Java executable program available online [4] which implements the *All Path tree Edit Distance* (AP-TED) proposed in [14]. This choice is motivated by the fact AP-TED is a recent robust and memory efficient version of the *tree Edit* distance. The Java executable program we have downloaded takes as input two node-labeled trees and returns their *tree Edit* distance. After executing the *NN* classifier on *FirstLast-L* using AP-TED as tree metric, a low accuracy of 59% was obtained. The corresponding confusion matrix is presented in Table 8.

Although the 59% of obtained by AP-TED are better than the best accuracy (44.25%) exhibited by the proposed distance when the specific tree properties are ignored, these two performances remain very unsatisfactory. But, the accuracy obtained by AP-TED is 41% lower than the perfect accuracy obtained by the proposed tree distance on this same database when the suitable set of targeted nodes properties is selected. This big gap is due to the fact that unlike the proposed tree distance, existing tree distances (including the *tree Edit* distance) cannot consider the specific nodes properties, even when these properties are clearly defined.

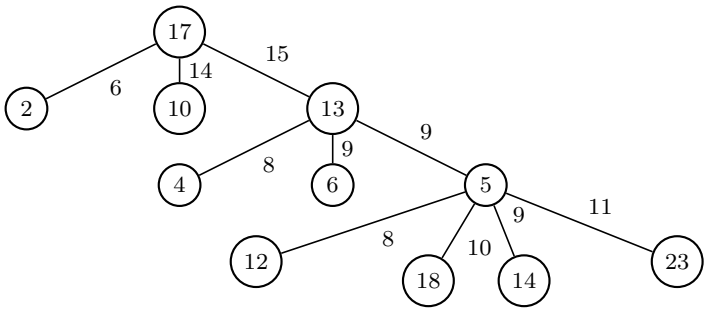
5.2.5 The new challenge

As it has been demonstrated throughout this section, the proposed tree distance can lead to a perfect *NN* classifier if the suitable set of targeted nodes properties is chosen. This choice was obvious for the databases *FirstLast-L* and *FirstLast-LW* because the specific properties verified by the content of each class were clearly specified. But this will generally not be case for other tree sets. Thus, the new challenge resides now in the manual or automatic discovery of the optimal set of targeted nodes properties that the best describes the trees in each tree set. Once this is done, the proposed tree distance may induce excellent classification results.

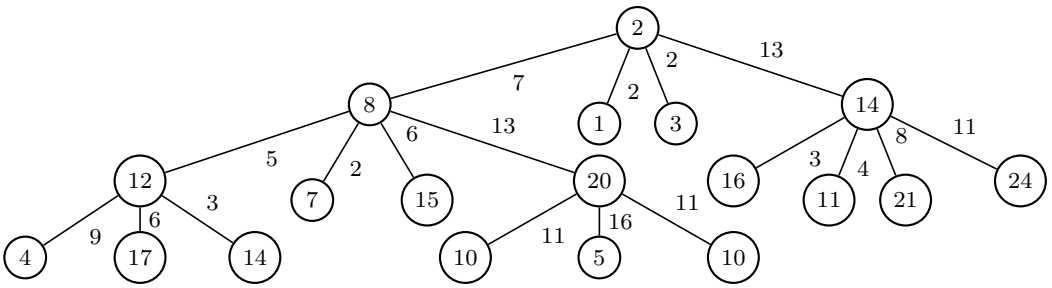
[4] http://tree-edit-distance.dbresearch.uni-salzburg.at/.



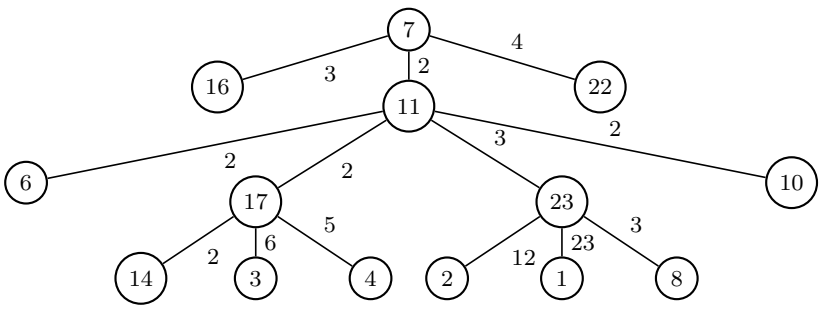
(a) Example of tree verifying the properties of class *First*



(b) Example of tree verifying the properties of class *Last*



(c) Example of tree verifying the properties of class *Both*



(d) Example of tree verifying the properties of class *None*

Fig. 14 Examples of node-labeled and arc-weighted trees verifying the properties presented in Table 5 for *FirstLast-LW*. The corresponding trees verifying the properties of *FirstLast-L* are obtained by ignoring the arc weights

Table 8 Confusion matrix of the *Nearest Neighbor* classifier in *FirstLast-L* when AP-TED is used as tree metric

	First	*Last*	*Both*	*None*
First	12	58	30	0
Last	0	98	2	0
Both	0	8	92	0
None	9	35	22	34

The accuracy is 59%

5.3 Time cost

We experimentally observed that the time cost of the proposed approach can be reduced to the time cost of the HMMs training phases. Consider a tree set $T = \{t_1, \dots, t_{|T|}\}$ and a set $P = \{p_1, \dots, p_m\}$ of targeted nodes properties. The *Baum-Welch* algorithm theoretically runs in $Time(T) = \theta\left(m.\gamma.(\sum_{k=1}^{|T|} |t_k|).N^2\right)$ to train the m HMMs associated with T. From this expression, one can deduce that the overall training time of the m HMMs associated with T is influenced by the following parameters:

1. The number m of targeted nodes properties.
2. The user-defined maximum number of iterations γ of the *Baum-Welch* algorithm.
3. The size $|T|$ of the tree set T.
4. The number $N = (depth(T) + 1)$ of states of the resulting HMMs.
5. The size $|t_k|$ of each tree, with $(1 \leq k \leq |T|)$.

When two tree sets T and T' are compared, the global training time is $(Time(T) + Time(T'))$. When the value of any of the aforementioned parameters is high, the overall training time increases. This can lead to a huge training time in some situations. The values of the parameters $|T|$, N and $|t_k|$ cannot be modified by the user since they are derived from the content of T. The two remaining parameters are user-dependent, they must therefore be carefully selected. In order to reduce the global training time cost, the following solutions can be applied:

1. Before comparing two tree sets, one should discover the optimal set of targeted nodes properties that the best describes the content of the two tree sets (following what is stated in Section 5.2.5). This will enable to avoid the inclusion of unnecessary properties while optimizing the comparison result.
2. The value of γ must not necessarily be very high to obtain accurate results. In this paper the value $\gamma = 100$ has been selected. Nevertheless, lower values can be experimented until a critical value under which the final result is seriously affected is reached.
3. Finally, the m HMMs can be trained in parallel.

5.4 Main assets of the proposed tree measures

The tree measures (distance and similarity) proposed in this paper:

1. Compare two tree sets, unlike existing measures which can only compare two single trees.
2. Compare two tree sets containing a mixture of trees belonging to various categories (rooted or unrooted, ordered or unordered), unlike existing measures which can only compare two trees of the same category.
3. Compare two tree sets containing weighted trees, unlike existing measures which can hardly accurately compare two weighted trees.
4. Handle the comparison of tree sets containing trees where each node and each arc can have several attributes (labels, weights), none of the existing measures can do this.
5. Are customizable because the targeted nodes properties on which the tree comparison is performed must be explicitly specified and there is no particular boundary on the possible choices of these properties. This is not possible with existing measures.
6. Are simple to understand and robust because they are based on *DFS* which is simple to understand and on HMMs which are handled by robust algorithms.
7. Are accurate because the proposed distance seriously outperformed the widespread *tree Edit* distance when they were both used for the flat *NN* classification of the database *FirstLast-L*.
8. Can naturally be implemented in parallel to reduce their computation time cost by concurrently training the HMMs associated with the various properties while using a parallel version of the *Baum-Welch* algorithm.

6 Conclusion

This paper addresses the problem of tree comparison. Many existing techniques have yet been proposed for this purpose, but they are all limited by several factors analyzed in Sect. 2.3. In order to overcome these limitations, a HMM-based technique for comparing two finite sets of rooted ordered trees which can be labeled or unlabeled, as well as weighted or unweighted is proposed in this work. The particularity of the proposed measures is that one must explicitly specify the targeted nodes properties (related to the topology, the labels and the weights) on which the tree comparison should be performed. Given a tree t and a set $P = \{p_1, \dots, p_m\}$ of targeted nodes properties, the proposed

approach follows the principle of the *DFS* algorithm to construct one Markov chain $\delta_i(t)$ for each property $p_i \in P$ $(1 \leq i \leq m)$. When this principle is applied for each property $p_i \in P$ on each tree of a tree set T, the set $\Delta_i(T)$ of Markov chains is obtained. This set is then used to initialize and to train a HMM $\lambda_i(T)$. Given two finite tree sets T and T', their m associated HMMs are compared by calculating the proposed distance $d_P(T, T')$ whose computation scheme is based on existing distances in $\mathbb{R}^m$ (i.e: *Euclidean*, *Manhattan*). The proposed similarity $\varphi_P(T, T')$ is finally derived from $d_P(T, T')$. These two measures have been generalized for the comparison of unrooted and unordered trees after transforming them into sets of rooted ordered trees.

The proposed measures have first been experimented for the comparison of two sets D and D' of text documents. Here, the common prefixes appearing in the various documents are exploited for constructing one node-labeled and arc-weighted prefix tree for each document. The resulting sets of prefix trees associated with D and D' are later compared using the proposed measures. This approach exhibited finer results than the recent technique proposed in [59].

We finally assessed the accuracy of the proposed tree distance by evaluating how it can empower the performances of the metric-based *NN* classifier. To achieve this goal, we constructed two synthetic tree databases named *FirstLast-L* and *FirstLast-LW*, each composed of four classes of 100 rooted ordered trees. *FirstLast-L* contains node-labeled trees and *FirstLast-LW* contains the same trees found in *FirstLast-L* with weighted arcs. The trees in these two experimental databases are characterized by specific and non-trivial properties presented in Table 5. The proposed distance has been experimented as tree metric for the *NN* classification of the trees in these two databases. When the specific nodes properties of these two databases were ignored, the low best accuracies of 44.25% and 36% were respectively obtained for *FirstLast-L* and *FirstLast-LW*. But when these specific properties were considered, a perfect accuracy of 100% was obtained for the two experimental databases. This last performance is 41% higher than performance obtained by the widespread *tree Edit* distance for *FirstLast-L*. The following interesting perspectives can be explored in future work:

1. Future work can conduct new experiments to evaluate how the proposed tree distance can empower the performances of other metric-dependent algorithms like clustering algorithms.
2. Consider a set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties. In this paper, we granted the same importance to every property $p_i \in P$ $(1 \leq i \leq m)$ during the computation of the distance and the similarity between two tree sets. However, depending on the selected application, it may be very interesting to grant different degrees of importance to these properties. Future work can implement modified versions of the proposed measures which consider the user-defined importance granted to each property.
3. In this paper, given a tree set T and a nodes property p_i, the number of states of the HMM $\lambda_i(T)$ associated with T for property p_i is $(depth(T) + 1)$. This setting becomes a problem when $depth(T)$ is high because an explosive number of states may lead to huge training time cost, as well as to not meaningful probabilities inside $\lambda_i(T)$. This limitation can be attenuated in future work by first organizing the depths' values as regular intervals, then by rather considering each state as an interval of depths (i.e: state $s_1 = [1, 3]$, state $s_2 = [4, 6]$, etc.).
4. Future work must focus on the design of manual and automatic techniques for discovering the optimal set of targeted nodes properties that the best describes the trees in each tree set. Frequent tree mining techniques stand as an excellent starting point to achieve this goal for the case of labeled trees.
5. Future work can explore the possibility of adapting the proposed approach for graph comparison. This can be done by transforming each graph G into a particular tree embedding all the information related to every vertex and every edge of G.
6. In this work, only the *Nearest Neighbor* classifier was experimented because it requires the use of a metric between two trees. However, there exist several other classifiers which are not metric-dependent and which cannot actually be experimented (i.e: *Support Vector Machines*, *Multilayer perceptron*, *Decision trees*, etc.). Indeed, for each singleton $\{t\}$, these classifiers require a descriptor vector $\vec{t}$ whose components belong to $\mathbb{R}$. Future work must find a way to use all the HMMs associated with $\{t\}$ for deriving a descriptor vector $\vec{t}$ associated with each singleton $\{t\}$ in order to enable the experimentation of classifiers which are not metric-dependent.
7. The time cost of the proposed approach can be considerably reduced if for each tree set, the HMMs training phase is realized in parallel by $|P|$ units of computations, where P is the set of targeted nodes properties. This parallel version theoretically divides the overall HMMs training time cost by $2 \times |P|$.
8. In addition to the former proposal, the HMMs training time cost can be further reduced in future work if each unit of computations is a cluster of processors. Here, a parallel version of the *Baum-Welch* algorithm must be executed by the members of the cluster. This can be

implemented analogically to the recent technique proposed in [64]. In that work, the authors used the *Message Passing Interface* (MPI) framework to implement in C language a parallel version of the HMM-based similarity measure between two finite sets of histograms initially proposed in [59]. That work exhibited an experimental average speed-up of 7.42 while using eight processors.

9. An alternative to the former MPI-based parallel implementation running on a cluster of computers is an implementation running on a *Field-Programmable Gate Array* (FPGA) chip. This can be realized following the principle of the FPGA-based version of the *Baum-Welch* algorithm proposed in [65].

References

1. Valiente G (2001) An efficient bottom-up distance between trees. In: spire, pages 212–219
2. Bille P (2003) Tree edit distance, alignment distance and inclusion. Technical report, Citeseer
3. Liu T-L, Geiger D (1999) Approximate tree matching and shape similarity. In: Proceedings of the Seventh IEEE International Conference on Computer Vision, volume 1, pages 456–462. IEEE
4. Bhavsar VC, Boley H, Yang L (2004) A weighted-tree similarity algorithm for multi-agent systems in e-business environments. Comput Intell 20(4):584–602
5. Tai K-C (1979) The tree-to-tree correction problem. J ACM (JACM) 26(3):422–433
6. Zhang K, Shasha D (1989) Simple fast algorithms for the editing distance between trees and related problems. SIAM J Comput 18(6):1245–1262
7. Zhang K, Statman R, Shasha D (1992) On the editing distance between unordered labeled trees. Inf Process Lett 42(3):133–139
8. Zhang K, Jiang T (1994) Some max snp-hard results concerning unordered labeled trees. Inf Process Lett 49(5):249–254
9. Klein PN (1998) Computing the edit-distance between unrooted ordered trees. In: European Symposium on Algorithms, pages 91–102. Springer
10. Chen W (2001) New algorithm for ordered tree-to-tree correction problem. J Algorithms 40(2):135–158
11. Touzet H (2007) Comparing similar ordered trees in linear-time. J Discrete Algorithms 5(4):696–705
12. Demaine ED, Mozes S, Rossman B, Weimann O (2009) An optimal decomposition algorithm for tree edit distance. ACM Trans Algorithms (TALG) 6(1):2
13. Pawlik M, Augsten N (2015) Efficient computation of the tree edit distance. ACM Trans Database Syst (TODS) 40(1):1–40
14. Pawlik M, Augsten N (2016) Tree edit distance: robust and memory-efficient. Inf Syst 56:157–173
15. Schwarz S, Pawlik M, Augsten N (2017) A new perspective on the tree edit distance. In: International Conference on Similarity Search and Applications, pages 156–170. Springer
16. Zhang K (1995) Algorithms for the constrained editing distance between ordered labeled trees and related problems. Pattern Recogn 28(3):463–474
17. Zhang K (1996) A constrained edit distance between unordered labeled trees. Algorithmica 15(3):205–222
18. Richter T (1997) *A new measure of the distance between ordered trees and its applications*. Inst für Informatik
19. Lu CL, Su Z-Y, Tang CY (2001) A new measure of edit distance between labeled trees. In: International Computing and Combinatorics Conference, pages 338–348. Springer
20. Ouangraoua A, Ferraro P, Tichit L, Dulucq S (2007) Local similarity between quotiented ordered trees. J Discrete Algorithms 5(1):23–35
21. Selkow SM (1977) The tree-to-tree editing problem. Inf Process Lett 6(6):184–186
22. Shin-Yee L (1979) A tree-to-tree distance and its application to cluster analysis. IEEE Trans Pattern Anal Mach Intell 2:219–224
23. Tanaka E, Tanaka K (1988) The tree-to-tree editing problem. Int J Pattern Recognit Artif Intell 2(02):221–240
24. Shasha D, Zhang K (1990) Fast algorithms for the unit cost editing distance between trees. J Algorithms 11(4):581–621
25. Sridharamurthy R, Talha BM, Adhitya K, Vijay N (2018) Edit distance between merge trees. In: IEEE transactions on visualization and computer graphics, pages 1–14
26. Jiang T, Wang L, Zhang K (1995) Alignment of trees–an alternative to tree edit. Theoret Comput Sci 143(1):137–148
27. Jansson J, Lingas A (2001) A fast algorithm for optimal alignment between similar ordered trees. In: Annual Symposium on Combinatorial Pattern Matching, pages 232–240. Springer
28. Kilpeläinen P, et al (1992) Tree matching problems with applications to structured text databases
29. Alonso L, Schott R (1993) On the tree inclusion problem. In: International Symposium on Mathematical Foundations of Computer Science, pages 211–221. Springer
30. Kilpeläinen P, Mannila H (1995) Ordered and unordered tree inclusion. SIAM J Comput 24(2):340–356
31. Richter T (1997) A new algorithm for the ordered tree inclusion problem. In: Annual Symposium on Combinatorial Pattern Matching, pages 150–166. Springer
32. Chen W (1998) More efficient algorithm for ordered tree inclusion. J Algorithms 26(2):370–385
33. Hoffmann CM, O'Donnell MJ (1982) Pattern matching in trees. J ACM 29(1):68–95
34. Kosaraju SR (1989) Efficient tree pattern matching. In: 30th Annual Symposium on Foundations of Computer Science, pages 178–183. IEEE
35. Dubiner M, Galil Z, Magen E (1990) Faster tree pattern matching. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science, pages 145–150. IEEE
36. Ramesh RAMAKRISHNAN, Ramakrishnan IV (1992) Nonlinear pattern matching in trees. J ACM (JACM) 39(2):295–316
37. Zhang KZ, Shasha D, Wang JT-L (1994) Approximate tree matching in the presence of variable length don't cares. J Algorithms 16(1):33–66
38. Farach M, Thorup M (1995) Fast comparison of evolutionary trees. Inf Comput 123(1):29–37
39. Amir A, Keselman D (1997) Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms. SIAM J Comput 26(6):1656–1669
40. Khanna S, Motwani R, Yao FF (1995) Approximation algorithms for the largest common subtree problem. Citeseer
41. Akutsu T, Halldórsson MM (2000) On the approximation of largest common subtrees and largest common point sets. Theor Comput Sci 233(1–2):33–50
42. Gupta A, Nishimura N (1998) Finding largest subtrees and smallest supertrees. Algorithmica 21(2):183–210
43. Nishimura N, Ragde P, Thilikos DM (2000) Finding smallest supertrees under minor containment. Int J Found Comput Sci 11(03):445–465
44. Tan P-N, Steinbach M, Kumar V et al (2006) Cluster analysis: basic concepts and algorithms. Intro Data Min 8:487–568

45. Mucherino A, Papajorgji PJ, Pardalos PM (2009) Data Mining in Agriculture, volume 34, chapter k-Nearest Neighbor Classification. Springer, New York
46. Bondy JA, Uppaluri SRM, et al (1976) Graph theory with applications, volume 290. Macmillan London
47. Cheung T-Y (1983) Graph traversal techniques and the maximum flow problem in distributed computation. IEEE Trans Software Eng 4:504–512
48. Wagner RA, Fischer MJ (1974) The string-to-string correction problem. J ACM (JACM) 21(1):168–173
49. Matoušek J, Thomas R (1992) On the complexity of finding iso-and other morphisms for partial k-trees. Discrete Math 108(1–3):343–364
50. Torsello A, Hancock ER (2006) Learning shape-classes using a mixture of tree-unions. IEEE Trans Pattern Anal Mach Intell 28(6):954–967
51. Torsello A, Rossi L (2011) Supervised learning of graph structure. In: International Workshop on Similarity-Based Pattern Recognition, pages 117–132. Springer
52. Rabiner LR (1989) A tutorial on hidden markov models and selected applications in speech recognition. Proc IEEE 77(2):257–286
53. Iloga S, Romain O, Tchuenté M (2020) An efficient generic approach for automatic taxonomy generation using HMMs. Pattern Anal Appl 1–22
54. Falkhausen M, Reininger H, Wolf D (1995) Calculation of distance measures between hidden markov models. In: Fourth European Conference on Speech Communication and Technology
55. Do MN (2003) Fast approximation of kullback-leibler distance for dependence trees and hidden markov models. IEEE Signal Process Lett 10(4):115–118
56. Silva J, Narayanan S (2008) Upper bound kullback-leibler divergence for transient hidden markov models. IEEE Trans Signal Process 56(9):4176–4188
57. Lyngso RB, Pedersen CN, Nielsen H (1999) Metrics and similarity measures for hidden markov models. In: Proc Int Conf Intell Syst Mol Biol, pages 178–186
58. Zeng J, Duan J, Chengrong W (2010) A new distance measure for hidden markov models. Expert Syst Appl 37(2):1550–1555
59. Iloga S, Romain O, Tchuenté M (2018) An accurate hmm-based similarity measure between finite sets of histograms. Pattern Anal Appl 1–26
60. Sahraeian SME, Yoon B-J (2011) A novel low-complexity hmm similarity measure. IEEE Signal Process Lett 18(2):87–90
61. Huang A (2008) Similarity measures for text document clustering. In: Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand, pages 49–56
62. Nothman J, Qin H, Yurchak R (2018) Stop word lists in free open-source software packages. In: Proceedings of Workshop for NLP Open Source Software (NLP-OSS), pages 7–12
63. Rico-Juan JR, Micó L (2003) Some results about the use of tree/string edit distances in a˜ nearest neighbour classification task. In: Iberian Conference on Pattern Recognition and Image Analysis, pages 821–828. Springer
64. Noussi JBB, Tchendji MT, Iloga S (2019) Parallel hmm-based similarity between finite sets of histograms. http://cri-info.cm/?page_id=148
65. Espinosa-Manzo ALA, Arias-Estrada MO (2001) Implementing hidden markov models in a hardware architecture. In: Proceedings of the International Meeting of Computer Science (ENC'01), Aguascalientes, Mexico, volume II, pages 1007–1016