Accurate comparison of tree sets using HMM-based descriptor vectors

Sylvain Iloga[1,2,3]

[1]Dept. of computer science, ENS
University of Maroua
P.O.box 55 Maroua
Cameroon

[2]IRD, UMMISCO F-93143
Sorbonne University
F-93143, Bondy
France

[3]ENSEA, CNRS, ETIS UMR 8051
CY Cergy Paris University
F-95000, Cergy
France

ABSTRACT. Trees are among the most studied data structures, and have been used in many applications where several techniques have been developed for comparing two trees belonging to the same category. Until the end of year 2020, there was a serious lack of metrics for comparing two weighted trees and the problem of comparing two tree sets was not specifically addressed. These limitations have been overcome in a paper published in 2021 where a customizable metric based on hidden Markov models have been proposed for comparing two tree sets, each containing a mixture of trees belonging to various categories. Unfortunately, that metric does not allow the use of non metric-dependent classifiers which take descriptor vectors as inputs. This paper addresses this drawback by deriving a descriptor vector for each tree set from the behavior of its associated models after a sufficiently long time. The comparison between two tree sets is then realized by comparing their associated descriptor vectors. Classification experiments involving many classifiers carried out on the databases *FirstLast-L* and *FirstLast-LW* showed an accuracy of 99.75%, which is close to the optimal value 100% exhibited by the original metric. Additional clustering experiments exhibited 54.25% and 98.75% of correctly clustered instances respectively for *FirstLast-L* and *FirstLast-LW*.

RÉSUMÉ. Les arbres sont parmi les structures de données les mieux étudiées dans de nombreux domaines et plusieurs techniques ont été développées pour comparer deux arbres appartenant à la même catégorie. Jusqu'en fin 2020, il y avait un sérieux manque de métriques pour comparer deux arbres pondérés et le problème de la comparaison de deux ensembles d'arbres n'était pas spécifiquement abordé. Ces limites ont été surmontées dans un article publié en 2021 dans lequel une métrique adaptative basée sur des modèles de Markov cachés a été proposée pour comparer deux ensembles d'arbres, chacun contenant un mélange d'arbres appartenant à diverses catégories. Malheureusement, cette métrique ne permet pas l'utilisation de classificateurs ne dépendant pas d'une métrique et prenant des vecteurs de descripteurs en entrée. Le présent article s'attaque à cette limite en dérivant un vecteur descripteurs pour chaque ensemble d'arbres à partir du comportement de ses modèles associés sur le long terme. La comparaison entre deux ensembles d'arbres est alors réalisée en comparant leurs vecteurs de descripteurs associés respectifs. Des expériences de classification impliquant de nombreux classificateurs effectuées sur les bases de données *FirstLast-L* et *FirstLast-LW* ont permi d'obtenir une meilleure précision de 99.75%, ce qui est presqu'identique aux 100% obtenus par la métrique d'origine. Des expériences de clustering supplémentaires ont exhibé 54.25% et 98.75% d'instances correctement groupées respectivement pour *FirstLast-L* et *FirstLast-LW*.

KEYWORDS : Trees, Comparison of tree sets, Distance between trees, Hidden Markov Models

MOTS-CLÉS : Arbres, Comparaison d'ensembles d'arbres, Distance entre arbres, Modèles de Markov cachés

1. Introduction

Trees are among the most common and well-studied data structures in a great variety of applications, including compiler design, graph transformation, automatic theorem proving, information retrieval, structured text database, pattern recognition, and signal processing [1, 2]. Consequently, several techniques have been developed for comparing two labeled or unlabeled trees belonging to the same category (i.e: rooted or unrooted, ordered or unordered). Until the end of year 2020, these existing techniques were based on one of the 5 following concepts: *tree Edit* [3–23], *tree Alignment* [24, 25], *tree Inclusion* [26–30], *tree pattern matching* [31–36] and *Subtrees/supertrees similarity* [37–42]. These existing techniques unfortunately embedded the three following main drawbacks:

1) There was a serious lack of metrics for comparing two weighted trees.

2) The problem of comparing two tree sets was not specifically addressed.

3) Existing techniques did not enable to explicitly specify the targeted nodes properties on which the tree comparison must be performed.

In 2021, these drawbacks have been overcome in [43] where a customizable metric based on Hidden Markov Models (HMMs) was proposed for comparing two tree sets, each containing a mixture of trees belonging to various categories. The metric proposed in that paper handles labeled and unlabeled trees, as well as weighted and unweighted trees where each node/arc can have several attributes (labels, weights). Unlike previous techniques, it allows the user to explicitly specify the targeted nodes properties on which the comparison should be performed and there is no boundary on the possible choices of these properties. The author of that paper relied on the *Depth-First Search* (*DFS*) traversal of a tree [44] to design the HMMs. The comparison between two tree sets was finally performed by comparing their associated HMMs. That metric showed perfect accuracies of 100% during flat classification experiments carried out on two online available synthetic databases [1] named *FirstLast-L* and *FirstLast-LW* using the *Nearest Neighbor* (*NN*) classifier. This performance was 41% higher than the one exhibited when the *tree Edit* distance was selected for *FirstLast-L*.

An important limitation of the metric proposed in that paper is related to the fact that it does not allow the use of non metric-dependent classifiers (i.e: *Support Vector Machines* [45], *Multilayer perceptron* [46], *Decision trees* [47], etc.) which absolutely require descriptor vectors as inputs. The current paper addresses this limitation by deriving a descriptor vector for each tree set from its associated models. Consider the set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties. Given a targeted nodes property $p_i \in P$ and the model $\lambda_i(T)$ associated with a tree set T according to p_i, we capture the overall proportion of time spent by $\lambda_i(T)$ observing each symbol after a sufficiently long time, irrespective of the state from which this symbol is observed as a characteristic of T. We apply this for all the properties in P and the resulting values are sequentially gathered as the components of a descriptor vector $\overrightarrow{T}$. Finally, the comparison between two tree sets is realized by comparing their associated descriptor vectors. The accuracy of the proposed descriptor vectors is evaluated through classification and unsupervised clustering experiments carried out on the databases *FirstLast-L* and *FirstLast-LW*.

The rest of this paper is organized as follows: the state of the art is presented in Section 2, followed by a summarized presentation of HMMs in Section 3. The description

1. http://perso-etis.ensea.fr/sylvain.iloga/FirstLast/index.html

of the proposed approach is realized in Section 4, while experimental results are exhibited in Section 5. The last section is dedicated to the conclusion.

2. State of the art

2.1. Related work

A tree can be defined as a connected acyclic graph. A detailed overview on graph theory and applications is available in [48]. Many distances have yet been proposed for comparing trees and they are based on one of the following concepts: *tree edit*, *tree alignment*, *tree inclusion*, *tree pattern matching*, *subtrees/supertrees similarity* and *HMMs*.

The *tree edit* distance [3–23] is based on the analysis of the number of edit operations required to transform a tree t_1 into another tree t_2. The three following edit operations are considered for a given node: insertion, deletion and substitution. *Tree alignment* [24, 25] is a particular case of *tree edit* where insertions are always performed before deletions. 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 [26–30] is the sum of the costs of the delete operations found in $S(t_1, t_2)$. *Tree pattern matching* [31–36] consists in finding the instances of a given pattern tree in a specific target tree. *Subtrees/supertrees similarity* [37–42] are generally realized by finding the maximum agreement subtree, the largest common subtree or the smallest common supertree.

Recently in 2021, a *HMM*-based technique have been proposed for comparing two finite tree sets [43]. The author of that paper remarked that *DFS* sequentially transits from one node depth to another and at each step, the algorithm observes the properties of the visited node. Consider the set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties. For every property $p_i{\in}P$, the former observation enabled to transform any tree t into a Markov chain $\delta_i(t)$ where the hidden states were the nodes depths, while the symbols were the values of property p_i for every node of t. When this principle was applied on a tree set $T = \{t_1, \ldots, t_n\}$, the set $\Delta_i(T) = \{\delta_i(t_1), \ldots, \delta_i(t_n)\}$ of Markov chains was obtained. The content of $\Delta_i(T)$ was later used to initialize then to train the HMM $\lambda_i(T)$ associated with T according to property p_i. Finally, the comparison between two tree sets was performed through the comparison of their associated HMMs for every property $p_i{\in}P$. The main assets of that technique are listed below:

1) It is designed for comparing finite tree sets.
2) It handles rooted/unrooted as well as ordered/unordered trees.
3) It compares weighted/unweighted as well as labeled/unlabeled trees.
4) It allows each node/arc to have many labels/weights.
5) It requires the specification of the targeted nodes properties.
6) It outperforms the *tree edit* distance with 41% of accuracy gain.

2.2. Problem statement

Until the end of year 2020, existing techniques for comparing two trees [3–42] embedded many drawbacks which were overcome in 2021 by the customizable HMM-based metric proposed in [43] for comparing two tree sets.

An important limitation of the technique proposed in [43] is related to the fact that it does not allow the use of non metric-dependent classifiers (i.e: *Support Vector Machines*, *Multilayer perceptron*, *Decision trees*, etc.) which absolutely require descriptor vectors as inputs. The current paper tackles this limitation by deriving a descriptor vector for each tree set from its associated models. The comparison between two tree sets is then realized by comparing their respective associated descriptor vectors.

3. Hidden Markov models

3.1. HMM definition

A HMM $\lambda = (A, B, \pi)$ is fully characterized by [49]:

1) The number N of states of the model. The set of states is $S = \{s_1, s_2, \dots, s_N\}$. 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 $\vartheta = \{v_1, v_2, \dots, v_M\}$. The symbol observed at time x is generally noted $o_x \in \vartheta$.

3) The state transition probability distribution $A = \{A[s_i, s_j]\}$ where $A[s_i, s_j] = Prob(q_{x+1} = s_j | q_x = s_i)$ with $1 \leq i, j \leq N$.

4) The symbols probabilities distributions $B = \{B[s_i, v_k]\}$ where $B[s_i, v_k] = Prob(v_k$ 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 $\pi = \{\pi[s_i]\}$ where $\pi[s_i] = Prob(q_1 = s_i)$ with $1 \leq i \leq N$.

3.2. Manipulation of a HMMs

Consider a sequence of symbols $O = o_1 o_2 \dots o_X$ and a HMM $\lambda = (A, B, \pi)$. The probability $Prob(O|\lambda)$ to observe O given λ is efficiently calculated by the *Forward-Backward* algorithm [49] which runs in $\theta(X.N^2)$. Given a sequence of symbols $O = o_1 o_2 \dots o_X$, it is possible to iteratively re-estimate the parameters of a HMM $\lambda = (A, B, \pi)$ in order to maximize the value of $Prob(O|\overline{\lambda})$, where $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi})$ is the re-estimated model. The *Baum-Welch* algorithm [49] is generally used to perform this re-estimation. This algorithm runs in $\theta(\beta.X.N^2)$ where β is the user-defined maximum number of iterations. The *Baum-Welch* algorithm can also train a HMM for multiple sequences. The algorithm maximizes the value of $Prob(O|\overline{\lambda}) = \sum_{k=1}^{K} Prob(O^{(k)}|\overline{\lambda})$ where $O = \{O^{(1)}, \dots, O^{(K)}\}$ is a set of K sequences of symbols and $O^{(k)} = o_1^{(k)} \dots o_{X_k}^{(k)}$ is the k^{th} sequence of symbols of O. In the case of multiple sequences, this algorithm runs in $\theta\left(\beta.(\sum_{k=1}^{K} X_k).N^2\right)$. In this paper, the value $\beta = 100$ is selected following [43].

3.3. Stationary distribution of a HMM

A vector $\varphi = (\varphi[s_1], \dots, \varphi[s_N])$ is a stationary distribution of a HMM $\lambda = (A, B, \pi)$ if:

1) $\forall j, (\varphi[s_j] \geq 0)$ and $\left(\sum_j \varphi[s_j] = 1\right)$

2) $\varphi = \varphi.A \Leftrightarrow (\varphi[s_j] = \sum_i \varphi[s_i] \times A[s_i, s_j], \forall j)$

$\varphi[s_j]$ estimates the overall proportion of time spent by λ in state s_j after a sufficiently long time. φ can be extracted from any line of the matrix $A^w = A{\times}A{\times}\ldots{\times}A$ (w times) when $w{\rightarrow}{+}\infty$. Therefore, the computation of φ requires $\theta(w.N^3)$ arithmetic operations.

4. The proposed approach

4.1. Main idea

Consider the set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties and let us assume that the *DFS* traversal of a tree t is executed by a robot r. According to the observation made by the author of [43], for each property $p_i{\in}P$, the robot r sequentially transits from one node depth to another and at each step, r observes the value $p_i(y)$ of the currently visited node $y{\in}t$. The MC $\delta_i(t)$ resulting from this traversal of t executed by the robot r according to property p_i embeds relevant information related to the overall behavior (movements, observations) of r. Given a tree set $T = \{t_1, \ldots, t_n\}$ and its corresponding set $\Delta_i(T) = \{\delta_i(t_1), \ldots, \delta_i(t_n)\}$ of MCs, the HMM $\lambda_i(T)$ was trained in [43] with the *Baum-Welch* algorithm to learn and to simulate the overall behavior of r during the traversal of the trees in T.

The main idea of the current paper is that, one can capture the behavior of r from $\lambda_i(T)$ by evaluating the overall proportion of time spent by r observing each possible value of p_i after a sufficiently long time, irrespective of the node depth from which this value is observed. More precisely, we evaluate the probability $\gamma(o|\lambda_i(T))$ of observing each symbol o after a sufficiently long time, given the model $\lambda_i(T)$, irrespective of the state from which o is observed. When this principle is applied for every property $p_i{\in}P$ and for every possible symbol, the resulting probabilities are sequentially saved as the components of the descriptor vector $\overrightarrow{T}$ associated with the tree set T.

4.2. Methodology

Consider a tree set $T = \{t_1, \ldots, t_n\}$ and the set $P = \{p_1, \ldots, p_m\}$ composed of m user-defined targeted nodes properties. The methodology applied in the current paper for deriving the descriptor vector $\overrightarrow{T}$ associated with T can be summarized in the three following steps:

1) **Tree modeling:** For each property $p_i{\in}P$, the tree modeling principle proposed in [43] [2] is preserved here to obtain the model $\lambda_i(T)$. Therefore, this step is not described in the following sections.

2) **Probability computing:** Let ϑ_i be the set of symbols of $\lambda_i(T)$. For every property p_i and for every symbol $o{\in}\vartheta_i$, we compute the probability $\gamma(o|\lambda_i(T))$ of observing o after a sufficiently long time, given the model $\lambda_i(T)$, irrespective of the state from which o is observed.

3) **Construction of the descriptor vector:** All the probabilities computed at step 2 are sequentially saved as the components of the descriptor vector $\overrightarrow{T}$ associated with T.

Figure 1 summarizes the main steps of the proposed methodology for deriving the descriptor vector $\overrightarrow{T}$ associated with a tree set T.

2. See Section 4.3 and Figure 4 of [43]

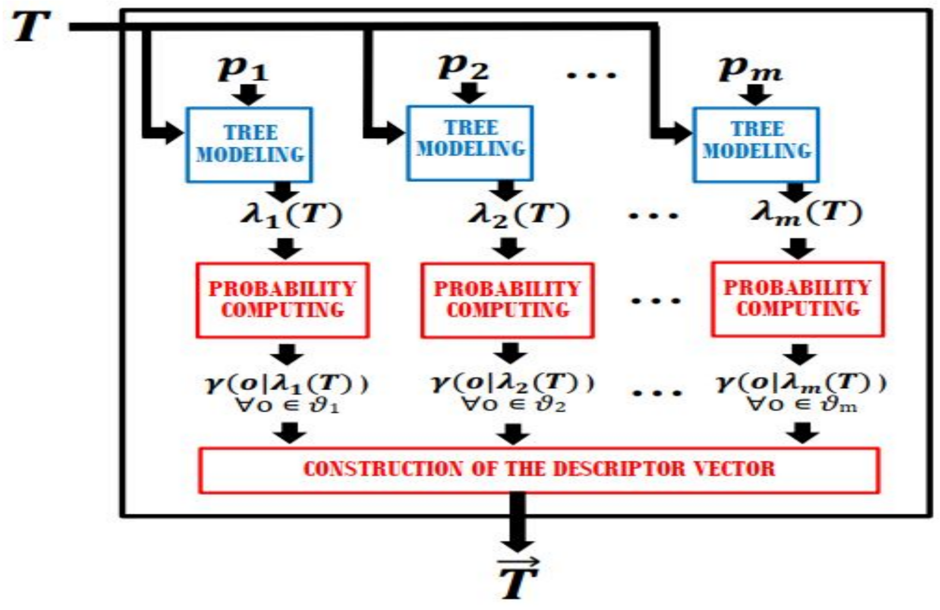
T
p_1
p_2
p_m
TREE MODELING
TREE MODELING
TREE MODELING
$\lambda_1(T)$
$\lambda_2(T)$
$\lambda_m(T)$
PROBABILITY COMPUTING
PROBABILITY COMPUTING
PROBABILITY COMPUTING
$\gamma(o|\lambda_1(T))$ $\forall o \in \vartheta_1$
$\gamma(o|\lambda_2(T))$ $\forall o \in \vartheta_2$
$\gamma(o|\lambda_m(T))$ $\forall o \in \vartheta_m$
CONSTRUCTION OF THE DESCRIPTOR VECTOR
$\overrightarrow{T}$

Figure 1 – Methodology for deriving the descriptor vector $\overrightarrow{T}$ associated with a tree set T.

4.3. Probability computing

Consider a tree set T and its associated model $\lambda_i(T) = (A_i^T, B_i^T, \pi_i^T)$ according to property $p_i \in P$, where $P = \{p_1, \ldots, p_m\}$ is the user-defined set of targeted nodes properties. In order to compute $\gamma(o|\lambda_i(T))$, one must first evaluate the overall proportion of time spent by $\lambda_i(T)$ observing o in state s_j after a sufficiently long time as follows:

1) Evaluate the overall proportion of time spent by $\lambda_i(T)$ in state s_j after a sufficiently long time. This proportion is given by the j^{th} component $\varphi_i^T[s_j]$ of the stationary distribution of $\lambda_i(T)$.

2) Multiply the result obtained at step 1 by the probability of observing o in state s_j which is $B_i^T[s_j, o]$.

The value of $\gamma(o|\lambda_i(T))$ is finally obtained by repeating this process for every state s_j and summing the resulting proportions as shown in Equation 1.

$$\gamma(o|\lambda_i(T)) = \sum_{j=1}^{N} \varphi_i^T[s_j] \times B_i^T[s_j, o] \tag{1}$$

4.4. Construction of the descriptor vector

The descriptor vector $\overrightarrow{T}$ associated with T is constructed by sequentially saving the values of $\gamma(o|\lambda_i(T))$ for every property $p_i \in P$ and for every symbol $o \in \vartheta_i$ as described in Algorithm 1. The first line of Algorithm 1 initializes the index of the components of the descriptor vector. Line 2 browses the properties, while line 3 browses the symbols for each property. The value of $\gamma(o|\lambda_i(T))$ is computed at line 4 according to Equation 1 and saved as the current component of the descriptor vector at line 5. Line 6 moves to the index of the next component of the descriptor vector. The descriptor vector is finally returned at line 9.

Algorithm 1 $Vector(T, P, \{\lambda_i(T)\}, \{\vartheta_i\}, \{\varphi_i^T\})$

```
1: $k \leftarrow 1$
2: for each $(p_i \in P)$ do
3:     for each $(o \in \vartheta_i)$ do
4:         $\gamma(o|\lambda_i(T)) \leftarrow \sum_{j=1}^{N} \varphi_i^T[s_j] \times B_i^T[s_j, o]$
5:         $\overrightarrow{T}[k] \leftarrow \gamma(o|\lambda_i(T))$
6:         $k \leftarrow k + 1$
7:     end for
8: end for
9: return $\overrightarrow{T}$
```

A consequence of Algorithm 1 is that the dimension of the descriptor vector $\overrightarrow{T}$ denoted in this paper as α is the sum of the numbers of symbols of all its $|P|$ associated models as precised in Equation 2. It can be easily demonstrated that Algorithm 1 requires $\theta(\alpha.N)$ arithmetic operations.

$$\alpha = \sum_{i=1}^{|P|} |\vartheta_i| \tag{2}$$

The following conventional property is adopted here to associate a descriptor vector with the empty set: '*The descriptor vector associated with $\emptyset$ according to the set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties is ($\overrightarrow{\emptyset} = \overrightarrow{0} \in \mathbb{R}^\alpha$).*'

4.5. Comparison of tree sets

The comparison between two tree sets T_1 and T_2 according to the set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties is performed here by calculating the value of any existing distance or similarity measure between their associated descriptor vectors $\overrightarrow{T_1}$ and $\overrightarrow{T_2}$. Equation 3 shows how this can be done with the *Euclidean* distance and the *Manhattan* distance. Several other possible distance and similarity measures are online available [3].

$$\begin{aligned} d_P(T_1, T_2) &= \sqrt{\sum_{k=1}^{\alpha} (\overrightarrow{T_1}[k] - \overrightarrow{T_2}[k])^2} \qquad \textit{(Euclidean)} \\ d_P(T_1, T_2) &= \sum_{k=1}^{\alpha} |\overrightarrow{T_1}[k] - \overrightarrow{T_2}[k]| \qquad \textit{(Manhattan)} \end{aligned} \tag{3}$$

5. Experimental results

5.1. Experimental tasks

The flat classification and clustering experiments realized in the current work are both realized with the *WEKA* soft [50]. Unlike [43] where only the *Nearest Neighbor* classifier was experimented, three additional non metric-dependent classifiers have also been experimented here. Thus, the four following classifiers have been experimented, their

3. https://numerics.mathdotnet.com/Distance.html

corresponding names in WEKA are in brackets: *Nearest Neighbor* (IBk), *Support Vector Machines* with polynomial kernels (SMO), *Multilayer perceptron* (MLP) and *Decision Trees* (J48). These classifiers have been used in WEKA with their default settings and a 10 folds cross-validation ($90\% - 10\%$) has been applied for each experiment.

The $Kmeans$ [51] and the *Expectation-Maximization* (EM) [52] clustering algorithms have been selected here to evaluate how much the proposed descriptor vectors can enable to accurately organize the experimental data (trees) into their corresponding classes (clusters) in an unsupervised process. The selected clustering algorithms have been used in *WEKA* with their default settings except the desired number of clusters which was initially fixed to the value 4 (the number of classes in each experimental database).

5.2. Experimental settings

The two synthetic tree databases selected during the experiments of the current work are *FirstLast-L* and *FirstLast-LW* [1] which have both been constructed and experimented in [43]. Each database contains 4 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 trees in these two databases are characterized by specific and non-trivial properties listed in [43] [4].

We realized two main 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. During the second main experience, these specific properties are considered. The sets $\overline{P}_L = \{\overline{p}_1, \overline{p}_2\}$, $\overline{P}_{LW} = \{\overline{p}_1, \overline{p}_2, \overline{p}_3\}$, $P_L = \{p_1, \ldots, p_8\}$ and $P_{LW} = \{p_1, \ldots, p_{12}\}$ of targeted nodes properties selected in [43] [5] for these experiences are also preserved in the current work.

In the context of each main experiment and considering each tree t of the experimental databases as the singleton $\{t\}$, we have executed Algorithm 1 to construct the descriptor vector $\overrightarrow{t}$ associated with t. The resulting descriptor vectors have then been saved into 4 online available '*arff*' files [1] which are taken as inputs by the soft *WEKA*.

5.3. Classification results

Classification results are presented in Tables 1a and 1b respectively for *FirstLast-L* and *FirstLast-LW*. These tables reveal that best accuracy obtained for these two databases is 99.75%. This is quite equal to the 100% obtained in [43]. Table 1a also reveals that the *NN* classifier performs better than the three non metric-dependent classifiers for *FirstLast-L*, irrespective of the set of targeted nodes properties. But the opposite situation is observed in Table 1b for *FirstLast-LW* where the three non metric-dependent classifiers show accuracies at least equal to 89% when the 3 default properties in $\overline{P}_{LW}$ are considered. This performance demonstrates the accuracy of the proposed descriptor vectors which enable to obtain good classification results, even with a default set of targeted nodes properties. Finally, when the 12 properties in P_{LW} are considered, all the classifiers exhibit comparable performances.

5.4. Clustering results

Clustering results are presented in Tables 2a and 2b respectively for *FirstLast-L* and *FirstLast-LW*. These tables reveal that best clustering performance obtained for *FirstLast-*

4. See Table 5 of [43]
5. See Section 5.2.2 of [43]

Table 1 – Classification results. Accuracies are in $(\%)$

(a)- *FirstLast-L*

	IBk		**SMO**	**MLP**	**J48**
	Euclid.	*Manhat.*			
$\overline{P}_L$	73.75	78	67.5	73.75	74.75
P_L	99.75	99.75	87.5	90.5	93.75

(b)- *FirstLast-LW*

	IBk		**SMO**	**MLP**	**J48**
	Euclid.	*Manhat.*			
$\overline{P}_{LW}$	84	87.25	90.75	89	89.75
P_{LW}	99.75	99.75	99.75	99.75	99.5

L is 54.25%, while a quasi perfect performance of 98.75% is obtained for *FirstLast-LW*. Given that the trees in *FirstLast-L* are only characterized by properties related to the topology and the node-labels, experimental results demonstrate that this database does not embed enough information to distinguish the 4 classes during an unsupervised clustering process. But when the arc-weights are additionally considered in *FirstLast-LW*, the 4 classes are accurately identified during the unsupervised clustering process with up to 98.75% correctly clustered instances when the 12 properties in P_{LW} are considered.

Table 2 – Clustering results. Correctly clustered instances are in $(\%)$

(a)- *FirstLast-L*

	Kmeans		**EM**
	Euclid.	*Manhat.*	
$\overline{P}_L$	47.5	47.25	54.25
P_L	45.5	44	42

(b)- *FirstLast-LW*

	Kmeans		**EM**
	Euclid.	*Manhat.*	
$\overline{P}_{LW}$	54.25	55.25	42.25
P_{LW}	86.5	92	98.75

5.5. Time cost

The time cost of the proposed technique for deriving the descriptor vector $\overrightarrow{T}$ of a tree set T according to the set P of targeted nodes properties can be step-by-step estimated as follows:

1) Design the $|P|$ HMMs associated with T: According to [43] [6], this step runs in $\theta\left(|P|.\beta.(\sum_{t\in T}|t|).N^2\right)$, where $|t|$ is the number of nodes in the tree t.

2) Compute the stationary distributions of the $|P|$ HMMs which runs in $\theta(|P|.w.N^3)$ with $w \rightarrow +\infty$. In the current work, the value $w = 100$ has been selected.

3) Construct the descriptor vector $\overrightarrow{T}$ associated with T using Algorithm 1 which runs in $\theta(\alpha.N)$.

6. Cf. Section 5.3 of [43]

It is according to these partial results that Equation 4 gives the expression of the theoretical time cost $Time(\overrightarrow{T})$ of the proposed approach.

$$\begin{aligned} Time(\overrightarrow{T}) &= \theta(a.N^3 + b.N^2 + \alpha.N) \text{ where} \\ N &= \max\{depth(t) \mid t{\in}T\} + 1 \\ a &= |P|.w \\ b &= |P|.\beta.\left(\sum_{t\in T} |t|\right) \end{aligned} \tag{4}$$

5.6. Comparison with [43]

The technique proposed in the current paper:

1) Is based on the HMMs designed in [43] for comparing two tree sets. It consequently inherits the main assets of that technique [7].

2) Unlike [43], it associates a descriptor vector $\overrightarrow{T}{\in}\mathbb{R}^\alpha$ whose components are interpretable with every tree set T such that all the operations that are applicable to vectors in $\mathbb{R}^\alpha$ are now also applicable to tree sets.

3) Performs a finer characterization of a tree set than [43]. Indeed, a tree set was characterized in [43] by its associated HMMs themselves. In the current paper, a tree set is rather characterized by meta-information related to the overall behavior of these HMMs after a sufficiently long time.

4) Exhibits good classification performances (78% for *FirstLast-L* and 90.75% for *FirstLast-LW*) with the default sets $\overline{P}_L$ and $\overline{P}_{LW}$ of targeted nodes properties, unlike [43] where lower performances were obtained (44.25% for *FirstLast-L* and 36% for *FirstLast-LW*) in identical conditions.

5) Shows an excellent classification accuracy of 99.75% for *FirstLast-L* and for *FirstLast-LW* when the suitable sets P_L and P_{LW} of targeted nodes properties are selected. This performance is quite identical to the 100% obtained in [43].

6) Is capable to correctly cluster 54.25% and 98.75% of the instances of *FirstLast-L* and *FirstLast-LW* respectively. No clustering was performed in [43].

6. Conclusion

The goal of this paper was to improve the HMM-based technique recently proposed in [43] for comparing two tree sets through the association of a descriptor vector with each tree set. Given a tree set T and a set $P = \{p_1, \ldots, p_m\}$ of targeted nodes properties, the author of [43] followed the principle of the *DFS* algorithm to associate $|P|$ HMMs with T, each HMM $\lambda_i(T)$ learned how much the nodes of the trees in T verify property p_i. The resulting models were finally compared to derive a distance and similarity between the two sets of trees. The technique proposed in [43] overcame the main limitations of the other existing techniques developed before its publication. Unfortunately, it did not allow the use of non metric-dependent classifiers which absolutely require descriptor vectors as inputs.

7. Cf. Section 5.4 of [43]

In order to derive the descriptor vector $\overrightarrow{T}$ associated with a tree set T in the current paper, we capture the behavior of its associated HMMs by evaluating the overall proportion of time spent by every HMM at observing each symbol o after a sufficiently long time, irrespective of the state from which o is observed. The resulting proportions are then sequentially saved as the components of the descriptor vector. The comparison between two tree sets is finally realized by comparing their associated descriptor vectors. Classification experiments involving four classifiers carried out on the databases *FirstLast-L* and *FirstLast-LW* showed a best accuracy of 99.75%. This is quite equal to the 100% obtained in [43]. Clustering experiments performed with the $Kmeans$ and the *Expectation-Maximization* algorithms exhibited 54.25% and 98.75% of correctly clustered instances respectively for *FirstLast-L* and *FirstLast-LW*. Future work can focus on the application of the proposed approach on real world datasets. Most of the perspectives stated at the end of [43] remain valid here and must be explored by future work.

7. References

[1] Gabriel Valiente. An efficient bottom-up distance between trees. In *spire*, pages 212–219, 2001.

[2] Philip Bille. Tree edit distance, alignment distance and inclusion. Technical report, Citeseer, 2003.

[3] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.

[4] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[5] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information processing letters*, 42(3):133–139, 1992.

[6] Kaizhong Zhang and Tao Jiang. Some max snp-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.

[7] Philip N Klein. Computing the edit-distance between unrooted ordered trees. In *European Symposium on Algorithms*, pages 91–102. Springer, 1998.

[8] Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.

[9] Hélène Touzet. Comparing similar ordered trees in linear-time. *Journal of Discrete Algorithms*, 5(4):696–705, 2007.

[10] Erik D Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms (TALG)*, 6(1):2, 2009.

[11] Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)*, 40(1):1–40, 2015.

[12] Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, 2016.

[13] Stefan Schwarz, Mateusz Pawlik, and Nikolaus Augsten. A new perspective on the tree edit distance. In *International Conference on Similarity Search and Applications*, pages 156–170. Springer, 2017.

[14] Kaizhong Zhang. Algorithms for the constrained editing distance between ordered labeled trees and related problems. *Pattern recognition*, 28(3):463–474, 1995.

[15] Kaizhong Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3):205–222, 1996.

[16] Thorsten Richter. *A new measure of the distance between ordered trees and its applications*. Inst. für Informatik, 1997.

[17] Chin Lung Lu, Zheng-Yao Su, and Chuan Yi Tang. A new measure of edit distance between labeled trees. In *International Computing and Combinatorics Conference*, pages 338–348. Springer, 2001.

[18] Aïda Ouangraoua, Pascal Ferraro, Laurent Tichit, and Serge Dulucq. Local similarity between quotiented ordered trees. *Journal of Discrete Algorithms*, 5(1):23–35, 2007.

[19] Stanley M Selkow. The tree-to-tree editing problem. *Information processing letters*, 6(6):184–186, 1977.

[20] Shin-Yee Lu. A tree-to-tree distance and its application to cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):219–224, 1979.

[21] Eiichi Tanaka and Keiko Tanaka. The tree-to-tree editing problem. *International Journal of pattern recognition and artificial intelligence*, 2(02):221–240, 1988.

[22] Dennis Shasha and Kaizhong Zhang. Fast algorithms for the unit cost editing distance between trees. *Journal of algorithms*, 11(4):581–621, 1990.

[23] Raghavendra Sridharamurthy, Bin Masood Talha, Kamakshidasan Adhitya, and Natarajan Vijay. Edit distance between merge trees. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, pages 1–14, 2018.

[24] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of treesâan alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[25] Jesper Jansson and Andrzej Lingas. A fast algorithm for optimal alignment between similar ordered trees. In *Annual Symposium on Combinatorial Pattern Matching*, pages 232–240. Springer, 2001.

[26] Pekka Kilpeläinen et al. Tree matching problems with applications to structured text databases. 1992.

[27] Laurent Alonso and René Schott. On the tree inclusion problem. In *International Symposium on Mathematical Foundations of Computer Science*, pages 211–221. Springer, 1993.

[28] Pekka Kilpeläinen and Heikki Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.

[29] Thorsten Richter. A new algorithm for the ordered tree inclusion problem. In *Annual Symposium on Combinatorial Pattern Matching*, pages 150–166. Springer, 1997.

[30] Weimin Chen. More efficient algorithm for ordered tree inclusion. *Journal of Algorithms*, 26(2):370–385, 1998.

[31] Christoph M Hoffmann and Michael J O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[32] S Rao Kosaraju. Efficient tree pattern matching. In *30th Annual Symposium on Foundations of Computer Science*, pages 178–183. IEEE, 1989.

[33] Moshe Dubiner, Zvi Galil, and Edith Magen. Faster tree pattern matching. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 145–150. IEEE, 1990.

[34] RAMAKRISHNAN Ramesh and IV Ramakrishnan. Nonlinear pattern matching in trees. *Journal of the ACM (JACM)*, 39(2):295–316, 1992.

[35] KZ Zhang, Dennis Shasha, and Jason Tsong-Li Wang. Approximate tree matching in the presence of variable length don' t cares. *Journal of Algorithms*, 16(1):33–66, 1994.

[36] Tyng-Luh Liu and Davi Geiger. Approximate tree matching and shape similarity. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 456–462. IEEE, 1999.

[37] Martin Farach and Mikkel Thorup. Fast comparison of evolutionary trees. *Information and Computation*, 123(1):29–37, 1995.

[38] Amihood Amir and Dmitry Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM Journal on Computing*, 26(6):1656–1669, 1997.

[39] Sanjeev Khanna, Rajeev Motwani, and F Frances Yao. *Approximation algorithms for the largest common subtree problem*. Citeseer, 1995.

[40] Tatsuya Akutsu and Magnús M Halldórsson. On the approximation of largest common subtrees and largest common point sets. *Theoretical Computer Science*, 233(1-2):33–50, 2000.

[41] Arvind Gupta and Naomi Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210, 1998.

[42] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M Thilikos. Finding smallest supertrees under minor containment. *International Journal of Foundations of Computer Science*, 11(03):445–465, 2000.

[43] Sylvain Iloga. Customizable hmm-based measures to accurately compare tree sets. *Pattern Analysis and Applications*, pages 1–23, 2021.

[44] To-Yat Cheung. Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Transactions on Software Engineering*, (4):504–512, 1983.

[45] Mariette Awad and Rahul Khanna. Support vector machines for classification. In *Efficient Learning Machines*, pages 39–66. Springer, 2015.

[46] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[47] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[48] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[49] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[50] Witten Ian H. and Frank Eibe. Data mining: Practical machine learning tools and techniques. `http://weka.sourceforge.net/`, 2005.

[51] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[52] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.