Learning Metrics Between Tree Structured Data: Application to Image Recognition*

Laurent Boyer[2], Amaury Habrard[1], and Marc Sebban[2]

[1] Université de Provence, LIF, France
[2] Université de Saint-Etienne, Laboratoire Hubert Curien, France
{laurent.boyer,marc.sebban}@univ-st-etienne.fr,
amaury.habrard@lif.univ-mrs.fr

Abstract. The problem of learning metrics between structured data (strings, trees or graphs) has been the subject of various recent papers. With regard to the specific case of trees, some approaches focused on the learning of edit probabilities required to compute a so-called stochastic tree edit distance. However, to reduce the algorithmic and learning constraints, the deletion and insertion operations are achieved on entire subtrees rather than on single nodes. We aim in this article at filling the gap with the learning of a more general stochastic tree edit distance where node deletions and insertions are allowed. Our approach is based on an adaptation of the EM optimization algorithm to learn parameters of a tree model. We propose an original experimental approach aiming at representing images by a tree-structured representation and then at using our learned metric in an image recognition task. Comparisons with a non learned tree edit distance confirm the effectiveness of our approach.

1 Introduction

In many machine learning or pattern recognition tasks, the choice of metrics plays an essential role for computing similarity between objects. Some classification, clustering or learning techniques are even intrinsically based on a metric, that is the case for the nearest-neighbors-based algorithms or some kernel-based methods. So, the choice or the parametrization of a similarity measure can drastically influence the result of an algorithm. One way to improve the influence of a metric is to integrate domain knowledge about the objects. While calling on an expert seems to be reasonable for small amount of data in domains where the background knowledge does exist, it becomes clearly intractable with huge data sets, where the expertise is low. In this context, a solution is to automatically infer the metric while capturing domain knowledge from a learning sample.

The general problem of learning metrics received an increasing interest since the beginning of 2000. With regards to numerical data, Bilenko et al [1] proposed an EM-based algorithm that integrates constraints and metric learning in the domain of semi-supervised clustering. Schultz et al [2] use some SVM techniques

* This work is funded by the MARMOTA project and the PASCAL Network of Excellence.

J.N. Kok et al. (Eds.): ECML 2007, LNAI 4701, pp. 54–66, 2007.

to learn a measure given a set of relative comparisons of the form *"x is closer to y than to z"*. Kummamuru et al in [3] improved these techniques providing the concept of Context-sensitive Learnable Asymmetric Dissimilarity (CLAD) measures. Bayoudh et al [4] proposed an approach for learning a measure by analogy in the form *"x is to y as z is to t"*. Concerning structured data, recent works have tried to tackle this learning problem with data represented by strings or trees. In the majority of the cases, they dealt with the edit distance (ED) [5] that handles three primitive edit operations (deletion, insertion, substitution) for changing an input instance into an output one. The resulting learned metrics lead to significant improvements on real world applications. For instance, Oncina et al. [6] introduced a string ED learning algorithm via the inference of a discriminative stochastic transducer. They showed a dramatic improvement on a handwritten digit recognition task, using Freeman codes for converting scanned digits to strings. In [7], Ristad and Yianilos provided a generative model of string ED, and illustrated its high utility on the difficult problem of learning the pronunciation of words in conversational speech. Recently, the Pascal network of excellence funded a pump priming project on the learning of a stochastic tree ED for musical recognition. A first learning algorithm, where deletions and insertions only concern entire subtrees, has been proposed in [8]. Although this type of tree ED is costless from an algorithmic standpoint (quadratic complexity [9] rather than a polynomial complexity of order 4 for a more general case [10]), it is not the most used in the literature because of a clear loss of generality. In this paper, we propose to overcome this restriction by allowing insertions and deletions of single nodes. However, this requires to define a new probabilistic learning framework. This is the main aim of this paper. Then, we propose to apply our learned metric on an image recognition task, whose novelty comes from the use of a structured representation of images. If much work has been done on images having high levels of definition, the question of recognizing small images for which the definition is too low to allow the application of numerical techniques (such as segmentation into regions) is still an open problem. Moreover, numerical vectors are, in general, not suited for expressing notions such as sequentiality or relationships between features. In this context, we think that a symbolic structural representation can provide a richer modeling of the object. Among the first approaches using a symbolic representation for image recognition, Jolion et al. [11] have proposed a method for encoding some relevant information of images in strings. The idea consists in extracting some characteristic points with a high level of contrast and to sort them in the form of a string. Despite of its interest, this representation does not include spatial knowledge, that implies a strong loss of information. In order to add this spatial information, one needs a two dimensional representation. In this paper, we propose an original representation of images in the form of trees, and we use our learned tree ED in an image recognition task.

The paper is organized as follows. We introduce in Section 2 some definitions and notations. Then, we recall the classic tree ED in Section 3. Section 4 deals

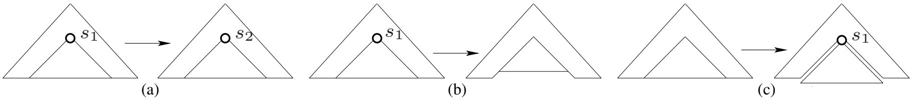


Fig. 1. (a) Substitution of $s_1 \in \mathcal{L}$ into s_2 (b) Deletion of s_1 (c) Insertion of s_1

with our stochastic model for learning tree edit probabilities. We finally present our application in image recognition in Section 5.

2 Definitions and Notations

We assume we handle ordered labeled trees of arbitrary arity. There is a left-to-right order among siblings of a tree and trees are labeled with elements of a set $\mathcal{L}$ of labels. We denote $\mathcal{T}(\mathcal{L})$ the set of all labeled trees buildable from $\mathcal{L}$.

Definition 1. *Let $\mathcal{V}$ be a set of nodes. We inductively define trees as follows: a node is a tree, and given T trees $a_1, .., a_T$ and a node $r \in \mathcal{V}$, $r(a_1, .., a_T)$ is a tree. r is the root of $r(a_1, .., a_T)$, and $a_1, .., a_T$ are subtrees.*

Definition 2. *Let $\mathcal{L}$ be a set of labels, and let $\lambda \notin \mathcal{L}$ be the empty label. Let $l : \mathcal{V} \rightarrow \mathcal{L}$ be a labeling function. $r(a_1, .., a_T)$ is a labeled tree if its nodes are labeled according to l.*

We assume that trees can be concerned by three edit operations (see Fig.1): The *substitution* operation which consists in changing the label $l(r)$ of a node r by another label of $\mathcal{L}$; the *deletion* operation which removes a node r of parent r', the children of r becoming a subsequence of those of r' according to a left-to-right order; the *insertion* operation adds a node r as a child of r' making r the parent of a subsequence of children of r' according to a left-to-right order.

Definition 3. *Assuming that a cost function assigns a cost to each edit operation, an edit script between two trees $r(a_1, .., a_T)$ and $r'(b_1, .., b_V)$ is a set of edit operations changing $r(a_1, .., a_T)$ into $r'(b_1, .., b_V)$. The cost of an edit script is the sum of the costs of its edit operations.*

Definition 4. *The edit distance (ED) between two trees $r(a_1, .., a_T)$ and $r'(b_1, .., b_V)$ is the cost of the minimum cost edit script.*

We are interested in the learning of a probabilistic tree ED. Roughly speaking, we aim at learning the probability of each edit operation used during a transformation process of an input tree into an output one. These probabilities are the parameters of a generative model describing a joint distribution over (input,output) pairs of trees. In [8], we proposed a first solution to this problem, in a restrictive case of tree edit distance, when a deletion (resp. an insertion) implies the removal (resp. the add) of an **entire subtree**. The objective of this paper is to fill the gap with a more general approach of the tree ED allowing the insertion/deletion of nodes. This case is more general because the deletion (or insertion) of an entire subtree can also be achieved by iteratively using the deletion (or insertion) operation on a single node. However, it implies to set a new probabilistic framework, intrinsically more difficult due to a larger size of the search space. To do that, we recall the principle of the algorithms computing such a tree ED. The interested reader can find more details in [10,12,13].

3 Tree ED Algorithm

To allow a larger spectrum of applications, the majority of the tree ED algorithms usually handled forests, a tree being a particular case of a forest.

Definition 5. *A forest $F = \{a_1, .., a_T\}$ is a set of trees. F is an ordered forest if there is a left-to-right order among the trees $a_1, .., a_T$ and if each tree is ordered.*

Definition 6. *Let F be a forest, and $\rho(a)$ be the root node of a tree $a \in F$. $F - \rho(a)$ is the forest obtained from F by the deletion of $\rho(a)$. Children of $\rho(a)$ becomes a sequence of trees of the forest $F - \rho(a)$. $f(\rho(a))$ is the forest composed of the children of $\rho(a)$. $F - a$ is the forest obtained by removing the tree a of F.*

Let F_1 and F_2 be two forests and a and b the rightmost trees of F_1 and F_2 respectively. Let δ be a cost function on pairs of labels, representing the edit operations. The ED $d(F_1, F_2)$ for the general case of forests is given by:

$$\begin{array}{ll}
d(\lambda,\lambda) & = 0 \\
d(F_1,\lambda) & = d(F_1-\rho(a),\lambda) + \delta(l(\rho(a)),\lambda) \\
d(\lambda,F_2) & = d(\lambda,F_2-\rho(b)) + \delta(\lambda,l(\rho(b))) \\
d(F_1,F_2) = min & \left\{\begin{array}{ll}
d(F_1-\rho(a),F_2) + \delta(l(\rho(a)),\lambda) & \backslash * \, deletion \\
d(F_1,F_2-\rho(b)) + \delta(\lambda,l(\rho(b))) & \backslash * \, insertion \\
d(F_1-a,F_2-b) + d(f(\rho(a)),f(\rho(b))) & \\
\ +\delta(l(\rho(a)),l(\rho(b))) & \backslash * \, substitution
\end{array}\right.
\end{array}$$

where $l(\rho(x))$ is the label of the root of tree x.

This pseudo-code suggests a dynamic programming approach to compute the tree ED. In fact, we can note that $d(F_1, F_2)$ depends on a constant number of relevant subproblems of smaller size. Zhang and Shasha [10] defined these subproblems from the notion of *keyroots* of a given tree a:

$$keyroots(a) = \{\rho(a)\} \cup \{r \in \mathcal{V}(a) | r\ has\ a\ left\ sibling\}.$$

From this set of keyroots (see Fig 2.a), one can deduce the set of *special subforests* of a (see Fig 2.b), defined by the forests $f(u)$, where $u \in keyroots(a)$. Zhang and Shasha also defined the set of relevant subproblems that allows us to design a dynamic programming algorithm to compute the tree ED. These relevant subproblems are all the forests corresponding to the prefixes of the special subforests (see Fig 2.(b+c)). Then, to compute the tree ED $d(F_1, F_2)$, one can show that is it sufficient to compute $d(S_1, S_2)$ for all relevant subproblems S_1 and S_2 (for more details see [10]). So far, we assumed that we had a function δ

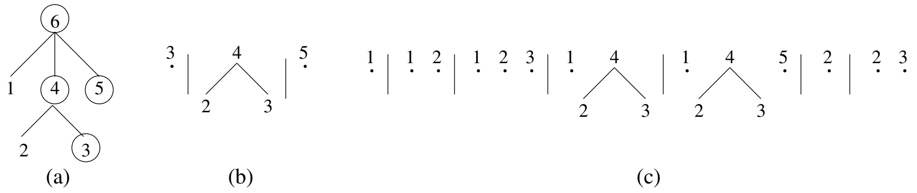


Fig. 2. (a) Example of keyroots represented by nodes with a circle, (a)+(b) the special subforests, and (a)+(b)+(c) the relevant subproblems

which returns the cost induced by an edit operation. In real world applications, these costs are often tuned by hand. We claim that machine learning techniques can efficiently be used to improve this task which can become tricky when the size of the alphabet is large. In the next section, we show how to automatically learn edit probabilities (rather than edit costs) from a learning set of tree pairs.

4 Learning Tree Edit Probabilities

4.1 Stochastic Tree ED

A *stochastic tree ED* supposes that edit operations occur according to a random process. We aim at learning the underlying distribution $\delta(s,s')$, $(s,s') \in (\{\mathcal{L} \cup \{\lambda\})^2$, in order to infer a generative model of all possible edit scripts. We will use a special symbol # to denote the end of an edit script. For sake of convenience, we will also denote the termination cost of a script $\delta(\#)$ by $\delta(\lambda,\lambda)$. To be a statistical distribution, the function δ must fulfill the following conditions:

$$\sum_{(s,s') \in (\mathcal{L} \cup \{\lambda\})^2} \delta(s,s') = 1 \;\; and \;\; \delta(s,s') \geq 0 \tag{1}$$

Let $e = e_1 \cdots e_n$ be an edit script with n edit operations ($e_i = (s,s') \neq (\lambda,\lambda)$), the probability of e is evaluated by: $p(e) = \prod_{i=1}^{n} \delta(e_i) \times \delta(\#)$. To model the distance between two trees, we propose to compute the probability of all ways to change a tree a into another one b (as described in [7] for the case of strings).

Definition 7. *Let two trees a and b, we denote by $E(a,b)$ the set of all possible edit scripts for transforming a in b. The stochastic tree ED between a and b is defined by: $d_s(a,b) = -\log \sum_{e \in E(a,b)} p(e)$.*

To learn the matrix δ, we propose to adapt the Expectation-Maximization (EM) algorithm [14] to this specific context of tree ED. Let us remind that EM estimates the hidden parameters of a probabilistic model by maximizing the likelihood of a learning sample. In our case, the parameters will correspond to the matrix δ of edit probabilities, and the learning sample will be composed of (input,output) tree pairs. In a pattern recognition task, these pairs can be either randomly generated from instances of the same class, or built by hand by an

Algorithm 1. $\boldsymbol{\alpha}(\{a_1,..,a_T\},\{b_1,..,b_V\})$

```
Input : Two forests $\{a_1,..,a_T\},\{b_1,..,b_V\}$
Let $\alpha$ be a matrix of dimension $(T+1)\times(V+1)$; $\alpha[\{\},\{\}]\leftarrow 1$
for $t$ de 0 à $T$ do
    for $v$ de 0 à $V$ do
        if $(t>0)$ or $(v>0)$ then
            $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}]=0$
        if $t>0$ then
            $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}]+=\delta(l(\rho(a_t)),\lambda).\alpha[\{a_1,..,f(\rho(a_t))\},\{b_1,..,b_v\}]$
        if $v>0$ then
            $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}]+=\delta(\lambda,l(\rho(b_v))).\alpha[\{a_1,..,a_t\},\{b_1,..,f(\rho(b_v))\}]$
        if $(t>0)$ and $(v>0)$ then
            $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}]+=\boldsymbol{\alpha}(f(\rho(a_t)),f(\rho(b_v))).\delta(l(\rho(a_t)),l(\rho(b_v)))$
                                              $.\alpha[\{a_1,..,a_{t-1}\},\{b_1,..,b_{v-1}\}]$
return $\alpha[\{a_1,..,a_T\},\{b_1,..,b_V\}]$
```

expert who judged them as being similar. EM achieves an expectation step followed by a maximization stage. During the first step, EM accumulates, from the training corpus, the expectation of each hidden event (edit operation) for transforming an input tree into an output one. In the maximization step, EM sets the parameter values (edit probabilities) in order to maximize the likelihood.

4.2 Forward and Backward Functions

To learn the matrix δ, EM uses two auxiliary functions, so-called *forward* (α) and *backward* (β), that are respectively described in Algorithms 1 and 2. The bold font is used for a recursive call of these algorithms ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$), while the normal font (α and β) describes intermediate values stored in a local matrix. We can note that both functions α and β return the quantity $\sum_{e\in E(a,b)} p(e)$, *i.e* the sum of probabilities of all paths (described by a script e) changing an input forest into an output one. Beyond the fact that they allow to compute the tree ED (cf Def. 7), they are overall combined to achieve the expectation step in order to estimate the expectation of each edit operation (see Fig. 3 and details in the next section). What is important to note is that functions α and β are nothing else but an extension to the stochastic case of the original tree ED algorithm. Actually, they contain the three main instructions corresponding to the three edit operations. For instance, considering the substitution operation, $\boldsymbol{\alpha}(f(\rho(a_t)),f(\rho(b_v)))$ and $\alpha[\{a_1,..,a_{t-1}\},\{b_1,..,b_{v-1}\}]$ are the stochastic version of $d(f(\rho(a)),f(\rho(b)))$ and $d(F_1-a,F_2-b)$ respectively. The main difference is that in our probabilistic framework, we use all the paths transforming a forest into another one, while the classic ED only keeps the costless path.

4.3 Expectation

During the expectation step, we estimate the expectation of the hidden events, *i.e* the edit operations used to transform an input tree into an output one. These expectations are stored in an auxiliary matrix γ $(|\mathcal{L}|+1)\times(|\mathcal{L}|+1)$. This process

Algorithm 2. $\boldsymbol{\beta}(\{a_1,..,a_T\},\{b_1,..,b_V\})$

```
Input       : Two forests$\{a_1,..,a_T\},\{b_1,..,b_V\}$
Let $\beta$ be a matrix of dimension $(T+1)\times(V+1)$; $\beta[\{\},\{\}] \leftarrow 1$
for $t$ de $T$ à $0$ do
    for $v$ de $V$ à $0$ do
        if $(t < T)$ or $(v < V)$ then
            $\beta[\{a_t,..,a_T\},\{b_v,..,b_V\}] = 0$
        if $t < T$ then
            $\beta[\{a_t,..,a_T\},\{b_v,..,b_V\}] += \delta(l(\rho(a_T)),\lambda).\boldsymbol{\beta}[\{a_t,..,f(\rho(a_T))\},\{b_v,..,b_V\}]$
        if $v < V$ then
            $\beta[\{a_t,..,a_T\},\{b_v,..,b_V\}] += \delta(\lambda,l(\rho(b_V))).\boldsymbol{\beta}[\{a_t,..,a_T\},\{b_v,..,f(\rho(b_V))\}]$
        if $(t < T)$ and $(v < V)$ then
            $\beta[\{a_t,..,a_T\},\{b_v,..,b_V\}] += \delta(l(\rho(a_T)),l(\rho(b_V))).\boldsymbol{\beta}(f(\rho(a_T)),f(\rho(b_V)))$
                                                    $.\boldsymbol{\beta}(\{a_t,..,a_{T-1}\},\{b_v,..,b_{V-1}\})$
return $\beta[\{a_1,..,a_T\},\{b_1,..,b_V\}]$
```

takes a training tree pair (x, y) in input. Then, for all the subtree pairs (a_t, b_v), where a_t is a subtree of x and b_v a subtree of y, it accumulates the expectations of the three edit operations consisting either in deleting $\rho(a_t)$, or inserting $\rho(b_v)$ or substituting $l(\rho(a_t))$ by $l(\rho(b_v))$. The pseudo-code of the expectation step is described in Algorithm 3, which requires the following definitions.

Definition 8. *A postorder traversal of a labeled tree $x = r(a_1, .., a_T)$ is obtained by first recursively visiting the subtrees $a_t, t = 1..T$ and then the root r. The postorder numbering assigns a number to each node of x according to a postorder traversal. Let $\phi_\alpha : \mathcal{V}(x) \rightarrow \mathcal{T}(\mathcal{L})^*$ be the function that takes a node r' and returns the ordered forest composed of the subtrees with root a node having a number strictly smaller than that of r' according to a postorder numbering.*

Definition 9. *Let $x = r(a_1, .., a_T)$ be an ordered tree. Let $\phi_\beta : \mathcal{V}(x) \rightarrow \mathcal{T}(\mathcal{L})^*$ be the function that takes a node r' and returns the ordered tree with root r and with the children of r having a number strictly smaller than that of r' according to a reverse postorder numbering.*

Fig. 3 shows an example of postorder (in arabic font) and reverse postorder numbering (in roman font). Considering the node labeled by `4|III` of the left tree, ϕ_α returns the forest composed of 3 subtrees with root the nodes labeled respectively by `1|VI`, `2|V` and `3|IV`, ϕ_β returning the subtree with root the node labeled by `6|I` and with the child `5|II`. Let us recall that this algorithm calculates the expectation of the number of times each edit operation is used for changing a tree x into another one y. To do this, for each edit operation (whose probability is given by δ), we consider not only all the ways leading to this operation (given by $\boldsymbol{\alpha}$) but also those allowing us to finish the transformation (given by $\boldsymbol{\beta}$) after the edit operation. While the deletion and insertion operations are quite understandable, the substitution one deserves some explanations. Fig. 3 graphically describes the substitution of the input node `4|III` into the output one `4|IV`. This requires to calculate the forward function $\boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) - f(\rho(a_t)), \phi_\alpha(\rho(b_v)) - f(\rho(b_v)))$, *i.e.* the probability of

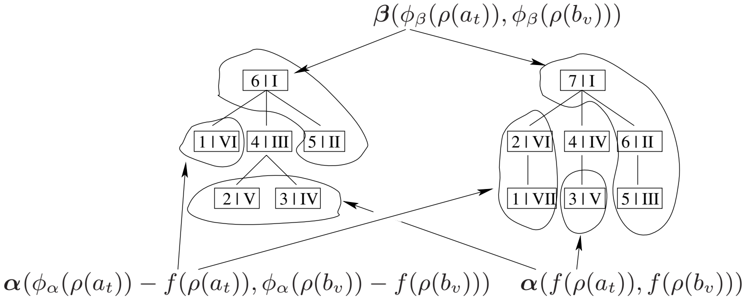

$\boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))$
6 | I
1 | VI
4 | III
5 | II
2 | V
3 | IV
7 | I
2 | VI
4 | IV
6 | II
1 | VII
3 | V
5 | III
$\boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) - f(\rho(a_t)), \phi_\alpha(\rho(b_v)) - f(\rho(b_v)))$
$\boldsymbol{\alpha}(f(\rho(a_t)), f(\rho(b_v)))$

Fig. 3. Illustration of the Expectation step for a substitution operation

Algorithm 3. expectation(x, y)

```
Input       : Two trees $x$ and $y$
Let $\mathcal{E}$ be the empty tree;
foreach $a_t$ s.t. $\rho(a_t) \in \mathcal{V}(x) \cup \mathcal{E}, b_v$ s.t. $\rho(b_v) \in \mathcal{V}(y) \cup \mathcal{E}$ do
    if $a_t \neq \mathcal{E}$ then
        $\gamma(l(\rho(a_t)), \lambda) += \frac{\boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)), \phi_\alpha(\rho(b_v)) \cup \{b_v\}).\delta(l(\rho(a_t)), \lambda).\boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))}{\boldsymbol{\alpha}(x, y)}$
    if $b_v \neq \mathcal{E}$ then
        $\gamma(\lambda, l(\rho(b_v))) += \frac{\boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) \cup \{a_t\}, \phi_\alpha(\rho(b_v))).\delta(\lambda, l(\rho(b_v))).\boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))}{\boldsymbol{\alpha}(x, y)}$
    if $(a_t \neq \mathcal{E})$ and $(b_v \neq \mathcal{E})$ then
        $\gamma(l(\rho(a_t)), l(\rho(b_v))) +=$
        $\frac{\boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) - f(\rho(a_t)), \phi_\alpha(\rho(b_v)) - f(\rho(b_v))).\boldsymbol{\alpha}(f(\rho(a_t)), f(\rho(b_v))).\delta(l(\rho(a_t)), l(\rho(b_v))).\boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))}{\boldsymbol{\alpha}(x, y)}$
```

the forest pair $[\{1|\mathtt{VI}\}, \{2|\mathtt{VI}(1|\mathtt{VII})\}]$. This forest pair is constituted of subtrees with root node having a numbering smaller than 4 according to a postorder numbering (given by function ϕ_α), minus subtrees that are the children of 4|III and 4|IV (given by function f). We estimate, as well, the backward function $(\boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v))))$ on the pair $[\{6|\mathtt{I}(5|\mathtt{II})\}, \{7|\mathtt{I}(6|\mathtt{II}(5|\mathtt{III}))\}]$, with nodes smaller than III for the input forest and IV for the output one, according to a reverse postorder numbering. We need also to compute the forward function $(\boldsymbol{\alpha}(f(\rho(a_t)), f(\rho(b_v))))$ on the pair $[\{2|\mathtt{V}, 3|\mathtt{IV}\}, \{3|\mathtt{V}\}]$ corresponding to the children of the nodes involved in the substitution operation.

4.4 Maximization

The final step of the EM algorithm is achieved by the maximization procedure presented in Algorithm 4. This step is crucial since it ensures a convergence of the process under constraints thanks to the normalization of the expectations. For learning a stochastic tree ED in the form of a generative model, we must fulfill constraints of Eq.1. This implies a simple normalization consisting in dividing each expectation $\gamma(s, s')$ by the total accumulator $TA = \sum_{(s,s') \in (\mathcal{L} \cup \{\lambda\})^2} \gamma(s, s')$. With Algorithms 1,2,3 and 4, we can now present in Algorithm 5 the global procedure for learning a stochastic tree ED.

Algorithm 4. ***maximization***

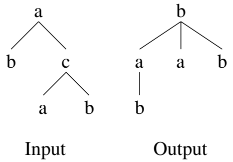
```
Input: A matrix of accumulators $\gamma$
Output: A matrix of probabilistic edit costs $\delta$
$TA \leftarrow 0$
foreach $(s, s') \in (\mathcal{L} \cup \{\lambda\})^2$ do $TA \leftarrow TA + \gamma(s, s')$
foreach $(s, s') \in (\mathcal{L} \cup \{\lambda\})^2$ do $\delta(s, s') \leftarrow \frac{\gamma(s,s')}{TA}$
```

Algorithm 5. ***tree edit distance − EM***

```
Input: $LS$ a learning set of tree pairs
repeat
    foreach $(s, s') \in (\mathcal{L} \cup \{\lambda\})^2$ do $\gamma(s, s') \leftarrow 0$
    foreach $(x, y) \in LS$ do expectation$(x, y)$
    maximization$(\gamma)$
until convergence
```

4.5 Example of Learning

We present here the running of our algorithm on a simple example, with an input alphabet $\mathcal{L}_1 = \{a, b, c\}$, an output alphabet $\mathcal{L}_2 = \{a, b\}$ and a training set composed of only one tree pair $[a(b, c(a, b)); b(c, a(b), a, b)]$ (see Fig. 4(a)). The algorithm converges towards an optimum after only 4 iterations. The learned matrix δ (initialized with random values) is described in Fig. 4(b). We can note that our algorithm has correctly learned the target. Actually, on this example, one optimal solution consists in: (i) inserting the symbol a that becomes the father of the symbol b, (ii) keeping unchanged the symbol b, (iii) deleting the symbol c and (iv) changing one out of twice the symbol a by b or by itself.



(a) Learning tree pair

δ	λ	a	b
λ	–	0.167	0
a	0	0.167	0.167
b	0	0	0.332
c	0.167	0	0

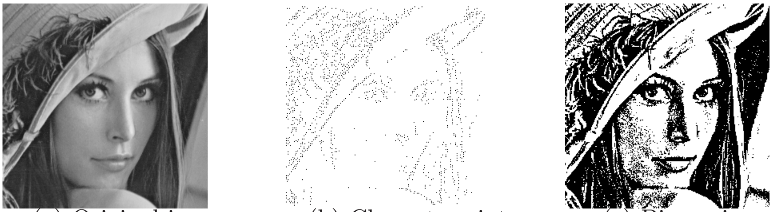
(b) Matrix δ after 4 EM iterations.

Fig. 4. Example of learning from a tree pair

5 Experiments in Image Recognition

5.1 From a Numerical to a Symbolic Representation of Images

In this section, we aim at verifying the interest of our learning algorithm on an image classification task. So far, the main trend in image recognition has mainly concerned numerical approaches based on color and texture [15,16,17]. However, many objects are poorly modeled with numerical values that can not express the relationships between attributes. Strings and trees are structured representations that allow us to take into account either the sequentiality or the



(a) Original image (b) Charact. points (c) Binary image

Fig. 5. Example of image represented by characteristic points

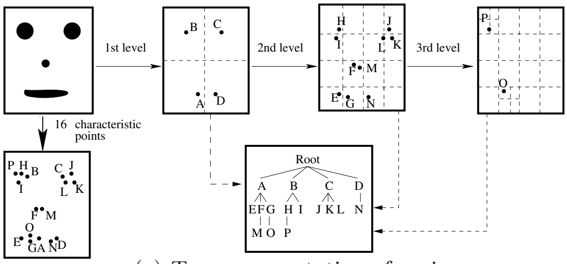
hierachization between attributes. A pioneer work has been recently achieved by Jolion et al. [11] with a string representation showing very interesting results on clustering and recognition tasks. The principle of this approach is first based on the extraction of characteristic points (see Fig. 5(b)) according to their contrast level in the original image (Fig. 5(a)). Then, to each of these points is assigned the symbol of the alphabet constituted of all the 512 binary masks 3 × 3, and that is applicable on that point in the binary version of the image (see Fig. 5(c)). Finally, these characteristic points are sorted according to their decreasing level of contrast, providing a sequence of masks labeled by a symbol $\in [0, 512]$.

5.2 Tree Representation of Images

This string representation outperformed numerical features in various classification and clustering tasks [11]. However, we can note that no spatial information is considered. One way to tackle this drawback is to consider a tree representation linking the depth of a tree with that information. To illustrate our approach, consider the example of Fig. 6(a). First, we propose to divide the image in four equal parts and we extract, for each of them, the characteristic point with the highest level of contrast. These four points constitute the first level of our tree. They are ordered from left-to-right according to their respective level of contrast. In a second step, we sub-divide each of the four original parts into four new sub-parts, and we extract again the characteristic points with the highest level of contrast in each sub-part. These points become the children of the node extracting during the first step. We recursively repeat this process until no more division can separate two points. To obtain a labeled tree, we assign to each node its corresponding mask applicable in the binary image. The main properties of this tree representation are the following: (i) We do not challenge the alphabet distribution observed in the sequence built with Jolion's approach; (ii) we keep the sequentiality between the characteristic points for each granularity level; and (iii) deep leaves represent a large local density of characteristic points.

5.3 Experimental Setup and Results

In order to assess the relevance of our model in a pattern recognition task, we applied it in handwritten digit classification. We used the NIST Special Database



(a) Tree-representation of an image.

SWL-SNL	TNL-SNL	TWL-SNL
+6	+11	+32

(b) The 1st column shows the interest of learning a string ED, the 2nd the contribution of our tree representation and the 3th the relevance of learning a tree ED.

Fig. 6. (a) Tree representation and (b) experimental results

3 of the National Institute of Standards and Technology, already used in several articles such as [18,6]. A part of this database consists in 128 × 128 bitmap images of handwritten digits written by 100 different writers. Each class of digit (from 0 to 9) has about 1,000 instances. We divided these data in a learning set LS (6,000 digits) and a test set TS (4,000 digits). Since our model handles trees, we coded each digit as previously explained but we reduced the alphabet from 2^9 to 15 by removing small frequent masks. Then, to construct a learning set of tree pairs, we used an uniform matrix δ of tree edit probabilities and we associated to each input tree $x \in LS$ the output tree $y \in LS, y \neq x$ s.t. $p(x, y)$ is maximal and s.t. x and y belong to the same class. We then learned matrix δ with our EM algorithm. We classified each digit $t \in TS$ by the class i of the tree $x \in LS$ maximizing $p(t, x)$ (result `TWL`).

We compared our learning approach with a standard tree ED with a priori fixed edit costs (result `TNL`). Moreover, to assess the relevance of our tree-based image representation, we used the same protocol with images coded in strings (see Section 5.1) using non learned and learned stochastic **string** EDs as presented in [6] (results `SNL` and `SWL`). To compare all the results, we present in Table 6(b) the relative accuracy gain on TS of each approach (`SWL`, `TNL`, `TWL`) in comparison with the standard string ED `SNL`. We can make the following interesting remarks: First, the results confirm the relevance of our tree-based image representation in comparison with strings (+11 percentage points); second, they definitely prove the interest of our approach for learning a tree similarity measure. Actually, not only a learned tree distance outperforms a standard string ED (+32 percentage points) but also it outperforms a non learned tree ED (+21 percentage points).

6 Conclusion

In this paper, we extended the tree ED, in its more general form, to a stochastic context. From this new point of view, the probabilities of the primitive edit operations are seen as hidden parameters that an adapted EM-based algorithm is able to learn from a set of tree pairs. We think that this work opens the door to significant improvements in classification and clustering, that is confirmed by our first experimental results in digit recognition. However, some problems deserve further investigations. First, we think that the constitution of the learning tree pairs can be highly improved and still constitutes an open problem; second, the tree representation issued from characteristic points must be further studied to tackle a larger spectrum of image recognition tasks; moreover, our algorithm has to be adapted in a form of a discriminative model (rather than the presented generative one) to handle small datasets; finally, in front of the emergence of huge datasets of XML documents, we plan to use our model on web applications.

References

1. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: 21th Int. Conf (ICML 2004), ACM Press, New York (2004)
2. Schultz, M., Joachims, T.: Learning a distance metric from relative comparisons. In: Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems], NIPS 2003, MIT Press, Cambridge (2003)
3. Kummamuru, K., Krishnapuram, R., Agrawal, R.: On learning asymmetric dissimilarity measures. In: Proc. of the 5th IEEE Int. Conf. on Data Mining (ICDM 2005), pp. 697–700. IEEE Computer Society Press, Los Alamitos (2005)
4. Bayoudh, S., Miclet, L., Delhay, A.: Learning by analogy: A classification rule for binary and nominal data. In: IJCAI, pp. 678–683 (2007)
5. Wagner, R., Fisher, M.: The string to string correction problem. Journal of the ACM (1974)
6. Oncina, J., Sebban, M.: Learning stochastic edit distance: application in handwritten character recognition. Journal of Pattern Recognition (2006)
7. Ristad, S., Yianilos, P.: Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(5), 522–532 (1998)
8. Bernard, M., Habrard, A., Sebban, M.: Learning stochastic tree edit distance. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 42–53. Springer, Heidelberg (2006)
9. Selkow, S.: The tree-to-tree editing problem. Information Processing Letters 6(6), 184–186 (1977)
10. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing, 1245–1262 (1989)
11. Jolion, J.: Some experiments on clustering a set of strings. In: Hancock, E.R., Vento, M. (eds.) GbRPR 2003. LNCS, vol. 2726, pp. 214–224. Springer, Heidelberg (2003)
12. Klein, P.: Computing the edit-distance between unrooted ordered trees. In: Proc. of the 6th European Symposium on Algorithms (ESA), pp. 91–102. Springer, Heidelberg (1998)
13. Bille, P.: A survey on tree edit distance and related problem. Theoretical Computer Science 337(1-3), 217–239 (2005)
14. Dempster, A., Laird, M., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc B(39), 1–38 (1977)
15. Pentland, A., Picard, R., Sclaroff, S.: Photobook: Tools for content-based manipulation of image databases. In: SPIE Storage and Retrieval of Image and Video Databases, vol. 2, pp. 18–32 (1995)
16. Wang, J., Li, J., Wiederhold, G.: Simplicity: Semantics-sensitive integrated matching for picture libraries. IEEE Trans. on Pat. Ana. Mach. Int. 23(9), 947–963 (2001)

17. Carson, C., Belongie, S., Greenspan, H., Malik, J.: Blobworld: Image segmentation using expectation-maximization and its application to image querying. IEEE Trans. on Pattern Analysis and Machine Intelligence 24(8), 1026–1038 (2002)
18. Gómez, E., Micó, L., Oncina, J.: Testing the linear approximating eliminating search algorithm in handwritten character recognition tasks. In: VI Symposium Nacional de reconocimiento de Formas y Análisis de Imágenes, pp. 212–217 (1995)