Learning Stochastic Tree Edit Distance

Marc Bernard, Amaury Habrard, Marc Sebban

Learning Stochastic Tree Edit Distance*

Marc Bernard[1], Amaury Habrard[2], Marc Sebban[1]

[1] EURISE – Université Jean Monnet de Saint-Etienne
23, rue Paul Michelon – 42023 Saint-Etienne cedex 2 – France
{marc.bernard,marc.sebban}@univ-st-etienne.fr
[2] LIF – Université de Provence
39, rue Frédéric Joliot Curie – 13453 Marseille cedex 13 – France
{amaury.habrard}@lif.univ-mrs.fr

Abstract. Trees provide a suited structural representation to deal with complex tasks such as web information extraction, RNA secondary structure prediction, or conversion of tree structured documents. In this context, many applications require the calculation of similarities between tree pairs. The most studied distance is likely the tree edit distance (ED) for which improvements in terms of complexity have been achieved during the last decade. However, this classic ED usually uses *a priori* fixed edit costs which are often difficult to tune, that leaves little room for tackling complex problems. In this paper, we focus on the learning of a stochastic tree ED. We use an adaptation of the Expectation-Maximization algorithm for learning the primitive edit costs. We carried out series of experiments that confirm the interest to learn a tree ED rather than a priori imposing edit costs.
Keywords. Stochastic tree edit distance, EM algorithm, generative models, discriminative models.

1 Introduction

Nowadays, there is a growing interest for tree-structured data due to the potential applications in information extraction from the web, computational biology or phylogeny. Indeed, the hierarchical structure of trees is more suited for modeling web pages (XML, HTML), the RNA secondary structure of a molecule or phylogenetic trees than a flat representation such as strings. In applications, one often needs similarity measures to compare two different instances. This is, for example, useful for defining conversion models for dealing with heterogeneous XML data. In this context, many approaches have extended the well known string edit distance (ED) to trees [1].

The tree ED is usually defined as the less costly set of basic operations to change one tree to another. These primitive operations are constituted of the *substitution*, the *deletion* and the *insertion* of a node. The tree ED-based methods use, in general, *a priori* fixed costs for these so-called primitive edit operations. However, in many domains, an edit cost can highly depend on the nature of the

* This work is part of the ongoing ARA Marmota research project

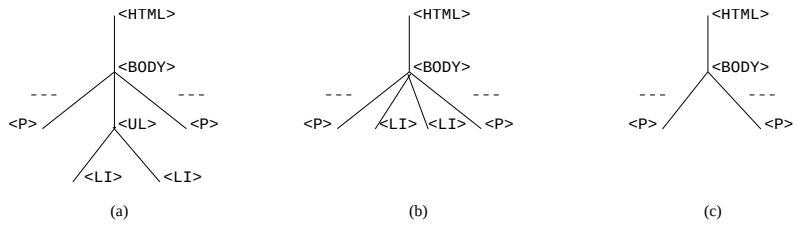


Fig. 1. Strategies to delete of a node within a tree.

symbols handled in a given operation. For example, the probability of changing a given symbol in a RNA structure depends on the probability that a genetic mutation occurs on this symbol. Thus, the similarity of two trees can strongly vary according to the specific domain in consideration. A solution could consist in assigning costs according to an expert valuation. However, this strategy may not be efficiently done in domains where the expertise is low. Moreover, even if the expertise level is sufficient, assigning a relevant cost to each edit operation can become a tricky task. Another way to overcome this drawback is to learn the edit costs from a sample of tree pairs. This can be achieved by modeling an ED as a stochastic process and using probabilistic methods to learn the model.

Note that in the context of strings, several approaches have been proposed during the last decade to learn a stochastic ED in the form of stochastic transducers [2, 3], conditional random fields [4], or pair-Hidden-Markov-Models (pair-HMM) [5]. A parametric approach has been presented in [6] in the context of graph ED, where each edit operation is modeled by a Gaussian Mixture Density. Nevertheless, as far as we know, no method was proposed to directly learn edit costs for a stochastic tree edit distance. The aim of this paper is to fill this gap by a stochastic method specifically adapted to trees.

As we said before, the primitive edit operations for the standard tree ED are the substitution, insertion and deletion of a node. The most efficient procedures proposed notably by Shasha *et al.* [7] and Klein [8] have a polynomial complexity of order 4. In these approaches, when a node r is deleted within a tree, all its children are then connected to the father of r. This may be not relevant in some cases, for example in an HTML document: considering a set of items in an unordered list (see Fig. 1.a), it seems clearly irrelevant to delete the `<UL>` node without deleting the `<LI>` items (Fig. 1.b). Thus, to overcome this drawback and to also reduce the algorithmic complexity, we decided to use the less costly (with a quadratic complexity) tree ED, initially proposed by Selkow [9], as a base of our stochastic approach[3]. In this case, only a deletion of an entire (sub)tree can occur, and its removal implies the deletion of all its nodes from the leaves (Fig. 1.c). Note that the insertion of a (sub)tree follows the same principle, *i.e.* requires the iterative insertion of its nodes.

[3] Note that our learning method can be adapted to any other tree ED.

We propose in this paper two approaches for learning, from a sample of (*input,output*) pairs of trees, the costs used for computing a stochastic tree ED. First, we learn a *generative* model in the form of a joint distribution over tree pairs inspired by [2] in the case of strings. The advantage of such generative models is to provide an estimate of the unknown joint density with a small variance. However, it has an important drawback: the estimate is biased because it depends on the distribution of the *input* trees. In other words, this generative model will work if the distribution over the learning input trees follows the unknown underlying density of the input trees. This constraint justifies our second approach based on the learning of a *discriminative* model in the form of a conditional distribution. This type of models is known [10] to provide an unbiased estimate (despite a higher variance). We will show that such a strategy will work whatever the input distribution we use.

The rest of the paper is organized as follows: After some notations and definitions about the classic tree ED in Section 2, our two learning methods are presented in Section 3. They are based on an adaptation of the well-known *Expectation-Maximization* algorithm (EM) [11]. In Section 4, we carry out several series of experiments before concluding.

2 Tree ED

After some notations and definitions about trees, we present the main edit operations allowing us to change a tree into another one. Then, we describe a usual breadth-first-scanning-based approach for computing the ED.

2.1 Notations and definitions

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 V be a set of nodes. We inductively define trees as follows: a node is a tree, and given T trees $a_1, \ldots, a_T$ and a node $v \in V$, $v(a_1, \ldots, a_T)$ is a tree. v is the root of $v(a_1, \ldots, a_T)$, and $a_1, \ldots, a_T$ are subtrees.*

Definition 2. *Let $\mathcal{L}$ be a set of labels, and let $\lambda \notin \mathcal{L}$ be the empty label. Let $\phi : V \to \mathcal{L}$ be a labeling function. $v(a_1, \ldots, a_T)$ is a labeled tree if its nodes are labeled according to ϕ. Assuming that $\phi(v)$ is equal to a given label $l \in \mathcal{L}$, for convenience, we will also denote the labeled tree $v(a_1, \ldots, a_T)$ by $l(a_1, \ldots, a_T)$.*

2.2 Edit operations and edit cost functions

We are only concerned by three possible edit operations on trees: deletion of a subtree a_i (denoted (a_i, λ)), insertion of a subtree a_j (denoted (λ, a_j)), and substitution of the label l of a tree root by l' (denoted (l, l')) (see Fig. 2). Let

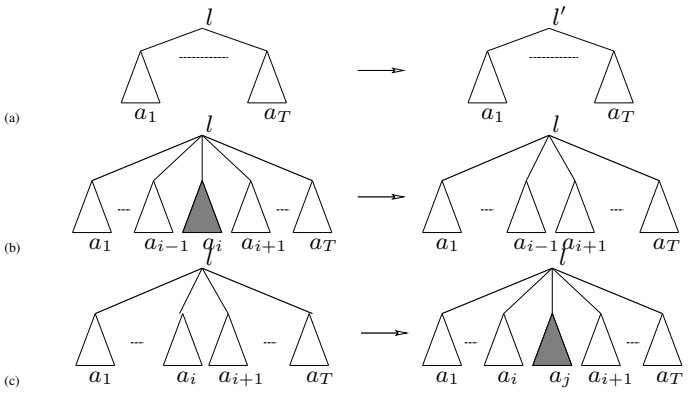


Fig. 2. (a) Substitution of l by l' (b) Deletion of a_i (c) Insertion of a_j

us define a cost function δ_t over these previous edit operations. Since a deletion or an insertion of a tree are respectively achieved by iteratively removing or inserting a set of nodes, δ_t can be directly defined from a cost function δ of edit operations on labels of the nodes. More formally, δ is a function defined from $(\mathcal{L} \cup \{\lambda\}) \times (\mathcal{L} \cup \{\lambda\}) \backslash \{(\lambda, \lambda)\}$ to $[0, 1]$.

The cost of the deletion of a tree can then be recursively computed as follows: $\delta_t(l(a_1, \ldots, a_T), \lambda) = \delta(l, \lambda) + \sum_{i=1}^{T} \delta_t(a_i, \lambda)$. As we said in introduction, the cost matrix δ is usually *a priori* fixed. For example, consider the cost matrix δ of Fig. 3 and a given tree $b(c, d)$, then $\delta_t(b(c, d), \lambda) = \delta(b, \lambda) + \delta_t(c, \lambda) + \delta_t(d, \lambda) = \delta(b, \lambda) + \delta(c, \lambda) + \delta(d, \lambda) = 1.5$. Based on the same principle, the insertion of a tree requires successive insertions of its nodes: $\delta_t(\lambda, l'(b_1, \ldots, b_V)) = \delta(\lambda, l') + \sum_{j=1}^{V} \delta_t(\lambda, b_j)$. Finally, the substitution of two labels is defined as follows: $\delta_t(l, l') = \delta(l, l')$.

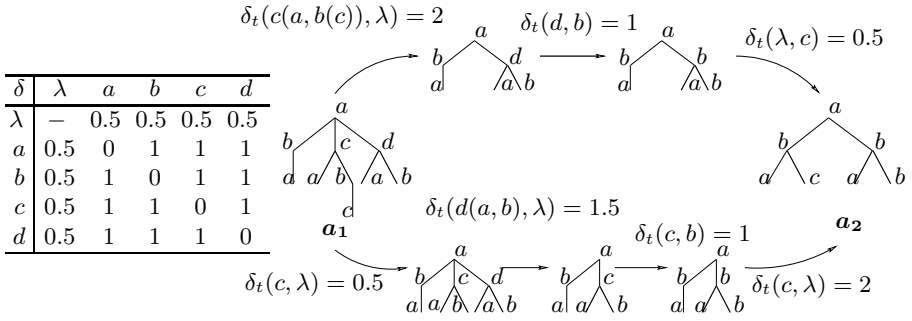
2.3 Classic tree ED algorithms

Once the cost function δ_t is established, it is possible to define a tree ED based on the following notion of edit script.

Definition 3. *Let a_1 and a_2 be two trees, an edit script on a_1 and a_2 is a sequence of edit operations changing a_1 into a_2. The cost of an edit script is the sum of the costs of its edit operations.*

Note that several scripts can exist (as shown in Fig. 3).

Definition 4. *The tree ED between two trees is the cost of the minimum cost edit script.*



δ	λ	a	b	c	d
λ	$-$	0.5	0.5	0.5	0.5
a	0.5	0	1	1	1
b	0.5	1	0	1	1
c	0.5	1	1	0	1
d	0.5	1	1	1	0

Fig. 3. A matrix δ and two possible edit scripts on two given trees a_1 and a_2.

The tree ED $d(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))$ between two trees $l(a_1,\ldots,a_T)$ and $l'(b_1,\ldots,b_V)$ as described in [9] can be recursively computed as follows:

$$\begin{aligned}
d(\lambda,\lambda) &= 0\\
d(l(a_1,\ldots,a_T),\lambda) &= \delta_t(l(a_1,\ldots,a_T),\lambda)\\
d(\lambda,l'(b_1,\ldots,b_V)) &= \delta_t(\lambda,l'(b_1,\ldots,b_V))\\
d(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V)) &= \delta(l,l')+d'(a_1,\ldots,a_T:b_1,\ldots,b_V)
\end{aligned}$$

where d' is defined as follows:

$$\begin{aligned}
d'(\lambda:\lambda) &= 0\\
d'(a_1,\ldots,a_T:\lambda) &= d'(a_1,\ldots,a_{T-1}:\lambda)+\delta_t(a_T,\lambda)\\
d'(\lambda:b_1,\ldots,b_V) &= d'(\lambda:b_1,\ldots,b_{V-1})+\delta_t(\lambda,b_V)\\
d'(a_1,\ldots,a_T:b_1,\ldots,b_V) &= \min\begin{cases} d'(a_1,\ldots,a_{T-1}:b_1,\ldots,b_V)+\delta_t(a_T,\lambda)\\ d'(a_1,\ldots,a_T:b_1,\ldots,b_{V-1})+\delta_t(\lambda,b_V)\\ d'(a_1,\ldots,a_{T-1}:b_1,\ldots,b_{V-1})+d(a_T,b_V)\end{cases}
\end{aligned}$$

This distance can be efficiently computed using dynamic programming. In the next section, we show how it is possible to automatically learn the matrix δ from a corpus of tree pairs. Our stochastic approach is based on an adaptation of the well known EM algorithm [11]. EM aims at estimating the hidden parameters of a probabilistic model from a learning sample. In our case, these parameters are the costs of the matrix δ. In the following, the densities (joint or conditional) will be denoted with a subscript δ when they will be estimated from δ.

3 Learning tree ED

We propose in the following two ways of learning a *stochastic edit distance* between two trees $l(a_1,\ldots,a_T)$ and $l'(b_1,\ldots,b_V)$. The first one concerns a *generative* model based on the estimation $p_\delta(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))$ of the unknown joint probability $p(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))$. The second proposition, a so-called *discriminative* approach, aims at learning a stochastic ED from the estimated conditional distribution $p_\delta(l'(b_1,\ldots,b_V)|l(a_1,\ldots,a_T))$. The main difference between the two approaches occurs during the maximization step of EM.

Input: Two trees $l(a_1, \ldots, a_i)$ and $l'(b_1, \ldots, b_j)$, $1 \leq i \leq T$ and $1 \leq j \leq V$
Output: Probability of pair $(l(a_1, \ldots, a_i), l'(b_1, \ldots, b_j))$
$\alpha[0..T, 0..V]$ a $(T+1) \times (V+1)$ matrix; $\alpha[0,0] \leftarrow \delta(l, l')$
for $t = 0$ *to* i **do**
  for $v = 0$ *to* j **do**
    if $(t > 0)$ *or* $(v > 0)$ **then** $\alpha[t, v] \leftarrow 0$
    if $(t > 0)$ **then** $\alpha[t, v] \leftarrow \alpha[t, v] + \boldsymbol{\alpha}(a_t, \lambda) \times \alpha[t-1, v]$
    if $(v > 0)$ **then** $\alpha[t, v] \leftarrow \alpha[t, v] + \boldsymbol{\alpha}(\lambda, b_v) \times \alpha[t, v-1]$
    if $(t > 0)$ *and* $(v > 0)$ **then** $\alpha[t, v] \leftarrow \alpha[t, v] + \boldsymbol{\alpha}(a_t, b_v) \times \alpha[t-1, v-1]$
return $\alpha[i][j]$

Algorithm 1: $\boldsymbol{\alpha}(l(a_1, \ldots, a_i), l'(b_1, \ldots, b_j))$

3.1 Joint tree ED

A *stochastic ED* supposes that edit operations occur according to an unknown random process. We aim at learning the underlying probability distribution $\delta(l, l')$ of these edit operations in order to estimate a joint probability distribution $p_\delta(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$ over tree pairs. We can show that this joint density will be valid if the following condition is fulfilled over the edit costs:

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

The joint probability represents the contribution of all ways to generate the two trees. $p_\delta(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$ is sufficient to model the *stochastic ED* defined as $d_s(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V)) = -\log p_\delta(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$.

To learn the matrix δ and then compute this joint probability $p_\delta(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$, we use an adaptation of the EM algorithm. Let us recall that EM achieves an expectation step followed by a maximization stage. During the first step, EM accumulates the expectation of each hidden event (edit operation) on the training corpus. In the maximization step, EM sets the parameter values (edit costs) to their relative expectations on the learning sample. To compute the joint probability, EM uses two auxiliary functions, so-called *forward* ($\boldsymbol{\alpha}$) and *backward* ($\boldsymbol{\beta}$).

To learn a stochastic tree ED, we adapted EM in the context of trees. The new forward function $\boldsymbol{\alpha}$ is described in Algorithm 1, whereas the new backward function $\boldsymbol{\beta}$ is presented in Algorithm 2.

These two functions are composed of two recursions: a breadth-first recursion on the children of the considered node and a depth-first one on the subtrees of this node. These two functions are symmetric. Although they process differently, they provide the same estimate $p_\delta(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$. Actually, the *forward* function visits the roots first and then scans the children from left to right, while the backward function processes from right to left and finally visits the roots of the tree pair. Fig. 4 illustrates these two algorithms.

Input: Two trees $l(a_i, \ldots, a_T)$ and $l'(b_j, \ldots, b_V)$, $1 \leq i \leq T$ and $1 \leq j \leq V$
Output: Probability of pair $(l(a_i, \ldots, a_T), l'(b_j, \ldots, b_V))$
$\beta[0..T, 0..V]$ a $(T+1) \times (V+1)$ matrix; $\beta[T, V] \leftarrow 1$
for $t = T$ *down to* $i - 1$ **do**
  for $v = V$ *down to* $j - 1$ **do**
    if $(t < T)$ *or* $(v < V)$ **then** $\beta[t, v] \leftarrow 0$
    if $(t < T)$ **then** $\beta[t, v] \leftarrow \beta[t, v] + \boldsymbol{\beta}(a_{t+1}, \lambda) \times \beta[t+1, v]$
    if $(v < V)$ **then** $\beta[t, v] \leftarrow \beta[t, v] + \boldsymbol{\beta}(\lambda, b_{v+1}) \times \beta[t, v+1]$
    if $(t < T)$ *and* $(v <)V$ **then** $\beta[t, v] \leftarrow \beta[t, v] + \boldsymbol{\beta}(a_{t+1}, b_{v+1}) \times \beta[t+1, v+1]$
if $i = 1$ *and* $j = 1$ **then return** $\beta[0][0] \times \delta(l, l')$ **else return** $\beta[i-1][j-1]$

Algorithm 2: $\boldsymbol{\beta}(l(a_1, \ldots, a_i), l'(b_1, \ldots, b_j))$

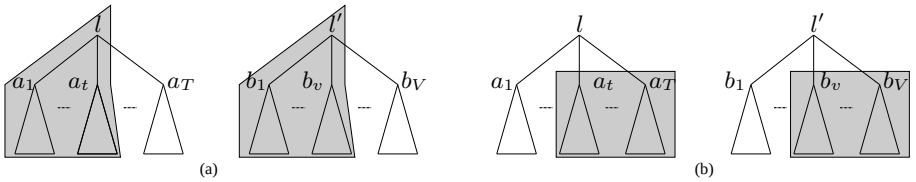


Fig. 4. (a) Evaluation of $\boldsymbol{\alpha}(l(a_1, \ldots, a_t), l'(b_1, \ldots, b_v))$ by the forward algorithm. (b) Evaluation of $\boldsymbol{\beta}(l(a_t, \ldots, a_T), l'(b_v, \ldots, b_V))$ by the backward algorithm.

Both functions can be computed with a quadratic complexity using dynamic programming techniques and allow us to define a probability distribution over pairs of trees:

$$\sum_{(a_i, b_j) \in (\mathcal{T}(\mathcal{L}))^2} p_\delta(a_i, b_j) = \sum_{(a_i, b_j) \in (\mathcal{T}(\mathcal{L}))^2} \boldsymbol{\alpha}(a_i, b_j) = \sum_{(a_i, b_j) \in (\mathcal{T}(\mathcal{L}))^2} \boldsymbol{\beta}(a_i, b_j) = 1$$

Let us present now the expectation and maximization steps for learning the edit costs. During the expectation step, we store in an auxiliary matrix γ $(|\mathcal{L}| + 1) \times (|\mathcal{L}| + 1)$ the expected number of times each edit operation was used to transform a tree in another one from a learning tree pairs LS. We apply for each tree pair $(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V)) \in LS$ the procedure *expectation*$(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$ described in Algorithm 3 (where $l_r(a_i)$ denotes the label of the root of a_i). Note that this function uses the previously mentioned *backward* and *forward* functions. Fig. 5 gives an illustration for evaluating a substitution.

The maximization step is crucial in the EM algorithm because it describes the normalization of the expectations ensuring a convergence of the process under constraints. The constraint to fulfill for learning a joint tree ED has been described in Eq.1. Thus, the normalization step is here very simple and only consists in dividing each expectation $\gamma(l, l')$ by the total accumulator $TA = \sum_{l \in \mathcal{L} \cup \{\lambda\}} \sum_{l' \in \mathcal{L} \cup \{\lambda\}} \gamma(l, l')$. The resulting maximization algorithm is described

Input: Two trees $l(a_1, \ldots, a_T)$ and $l'(b_1, \ldots, b_V)$
for t *from* 0 *to* T **do**
  for v *from* 0 *to* V **do**
    if $(t > 0)$ **then**
      $\gamma(l_r(a_t), \lambda) \leftarrow \gamma(l_r(a_t), \lambda) + \frac{\boldsymbol{\alpha}(l(a_1,\ldots,a_{t-1}),l'(b_1,\ldots,b_v))\boldsymbol{\alpha}(a_t,\lambda)\boldsymbol{\beta}(l(a_{t+1},\ldots,a_T),l'(b_{v+1},\ldots,b_V))}{\boldsymbol{\alpha}(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))}$
      $expectation(a_t, \lambda)$
    if $(v > 0)$ **then**
      $\gamma(\lambda, l_r(b_v)) \leftarrow \gamma(\lambda, l_r(b_v)) + \frac{\boldsymbol{\alpha}(l(a_1,\ldots,a_t),l'(b_1,\ldots,b_{v-1}))\boldsymbol{\alpha}(\lambda,b_v)\boldsymbol{\beta}(l(a_{t+1},\ldots,a_T),l'(b_{v+1},\ldots,b_V))}{\boldsymbol{\alpha}(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))}$
      $expectation(\lambda, b_v)$
    if $(t > 0)$ *and* $(v > 0)$ **then**
      $\gamma(l_r(a_t), l_r(b_v)) \leftarrow \gamma(l_r(a_t), l_r(b_v)) + \frac{\boldsymbol{\alpha}(l(a_1,\ldots,a_{t-1}),l'(b_1,\ldots,b_{v-1}))\boldsymbol{\alpha}(a_t,b_v)\boldsymbol{\beta}(l(a_{t+1},\ldots,a_T),l'(b_{v+1},b_V))}{\boldsymbol{\alpha}(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))}$
      $expectation\ (a_t, b_v)$

Algorithm 3: ***expectation***$(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$

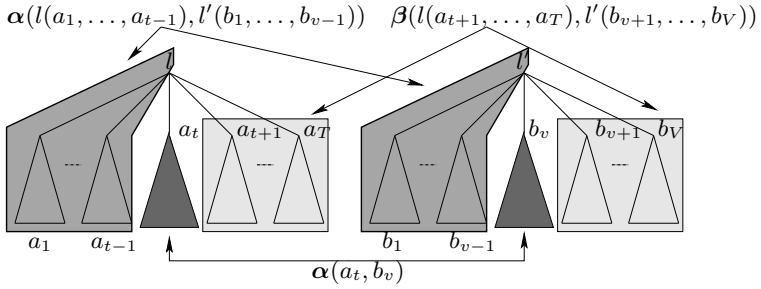


Fig. 5. Use of the forward and backward functions to evaluate a substitution cost.

in Algorithm 4. By combining Algorithms 1,2,3,4, we can now draw the general learning algorithm of a joint stochastic tree ED (see Algorithm 5). Note that the process is repeated until convergence. This is reached when the probability of each edit operation does not significantly change between two iterations.

Note that it is the normalization achieved in the maximization step that allows us to learn a joint distribution $p_\delta(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$. However, in order to use such a model in a classification task (for example for converting a structured document a_i into another one b_j), we would need a conditional distribution $p_\delta(b_j|a_i)$ rather than a joint one. Actually, in such a context, the input tree is known and we are looking for the optimal corresponding output. A simple solution would consist in computing $p_\delta(b_j|a_i)$ from the joint distribution such that $p_\delta(b_j|a_i) = \frac{p_\delta(a_i,b_j)}{p(a_i)}$. However, this implies a dependence on the input distribution $p(a_i)$, and thus can generate a bias.

Input: A matrix of accumulators γ
Output: A matrix of joint stochastic edit costs δ
$TA \leftarrow 0$
foreach $(l, l') \in (\mathcal{L} \cup \{\lambda\})^2$ **do** $TA \leftarrow TA + \gamma(l, l')$
foreach $(l, l') \in (\mathcal{L} \cup \{\lambda\})^2$ **do** $\delta(l, l') \leftarrow \frac{\gamma(l,l')}{TA}$

Algorithm 4: ***maximization*** (for joint distribution)

Input: LS a learning set of tree pairs
repeat
  foreach $(l, l') \in (\mathcal{L} \cup \{\lambda\})^2$ **do** $\gamma(l, l') \leftarrow 0$
  foreach $(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V)) \in LS$ **do**
    expectation$(l(a_1, \ldots, a_T), l'(b_1, \ldots, b_V))$
  maximization(γ)
until *convergence*

Algorithm 5: ***expectation − maximization***

One solution to overcome this drawback consists in directly learning a conditional distribution, usually called a discriminative model. The advantage of this approach is to remove the statistical bias of generative models. This is the goal of the next section. We propose a new maximization step aiming at normalizing the accumulators obtained after the expectation step such as to directly obtain a conditional distribution $p_\delta(b_j|a_i)$ at each stage of EM.

3.2 Learning conditional tree ED

To achieve this task, we have to draw the new constraints corresponding to this conditional context $p_\delta(b_j|a_i)$. In fact, it is possible to model the output distribution conditionally to an input tree a_i in the form of a non deterministic probabilistic finite state automaton. Let us take a simple example to explain the principle. We assume that the input tree $a(b(b), a)$ is the one described in Fig. 6(a). Since we use a breadth-first scanning for computing the ED, $a(b(b), a)$ can be rewritten in the form of the string "abab". Thus, it is possible to model the output distribution conditionally to the input tree in the form of the probabilistic automaton of Fig. 6(b).

The cycles of each state correspond to the possible insertions before and after the reading of an input symbol. The state with a double circle is a final state and corresponds to the end of the reading of the input tree (which will be characterized by the termination symbol #). In order to learn a statistical distribution over the pairs of trees, it is easy to show that this automaton must satisfy the following two conditions:

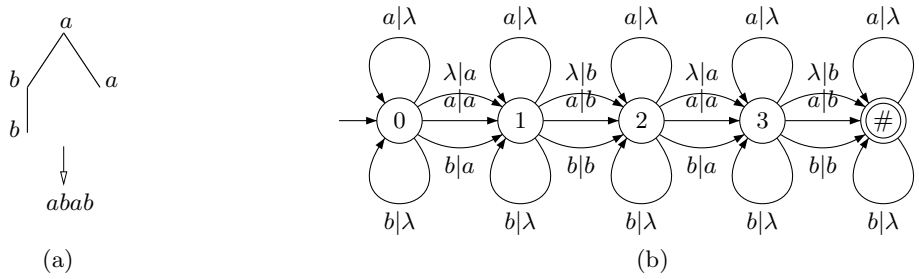


Fig. 6. Output distribution conditionally to an input tree

1. First, probabilities of the outgoing transitions of each state must sum to 1:

$$\forall l \in \mathcal{L}, \sum_{l' \in \mathcal{L} \cup \{\lambda\}} \delta(l'|l) + \sum_{l' \in \mathcal{L} \cup \{\lambda\}} \delta(l'|\lambda) = 1 \tag{2}$$

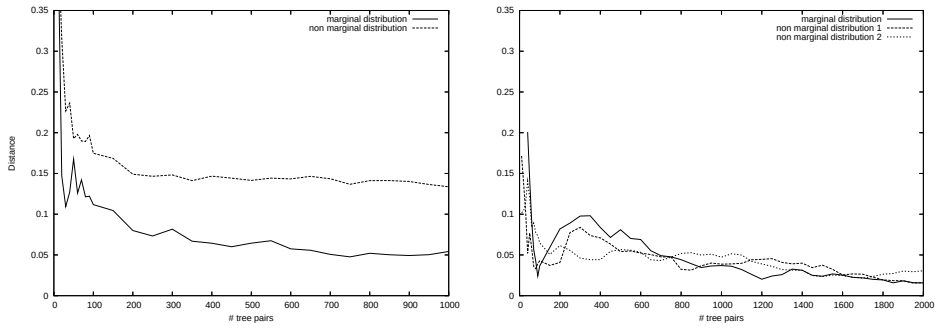
   where $\delta(l'|l)$ is now the probability to generate the output symbol l' conditionally to the input symbol l.
2. Second, probabilities from the final state must also describe a distribution:

$$\sum_{l' \in \mathcal{L}} \delta(l'|\lambda) + \delta(\#) = 1. \tag{3}$$

The optimal normalization under these new constraints is the solution of an optimization problem as that of presented in Dempster et al. [11]. In the following, we only provide in Algorithm 6 the normalization that fulfills these constraints 2 and 3. Due to the lack of space, we do not provide here the proof justifying this optimal solution, but the interested reader can find in [3] the principle of this proof in the case of *string pairs*.

Input: A matrix of accumulators γ
Output: A matrix of conditional stochastic edit costs δ
$N \leftarrow \sum_{l \in \mathcal{L} \cup \{\lambda\}} \sum_{l' \in \mathcal{L} \cup \{\lambda\}} \gamma(l, l')$; $N(\lambda) \leftarrow \sum_{l' \in \mathcal{L}} \gamma(\lambda, l')$
foreach $l \in \mathcal{L}$ **do** $N(l) \leftarrow \sum_{l \in \mathcal{L} \cup \{\lambda\}} \gamma(l, l')$
$\delta(\lambda|\lambda) \leftarrow \frac{N - N(\lambda)}{N}$
foreach $(l, l') \in (\mathcal{L} \cup \{\lambda\})^2$ **do** $\delta(l'|l) \leftarrow \frac{\gamma(l,l')}{N(l)} \frac{N - N(\lambda)}{N}$
foreach $l \in \mathcal{L}$ **do** $\delta(\lambda|l) \leftarrow \frac{\gamma(l,\lambda)}{N(l)} \frac{N - N(\lambda)}{N}$
foreach $l' \in \mathcal{L}$ **do** $\delta(l'|\lambda) \leftarrow \frac{\gamma(\lambda,l')}{N}$

Algorithm 6: ***maximization*** (for conditional distribution)



(a) (b)

Fig. 7. Results of our experiments.

4 Experiments

We carried out experiments to assess the relevance of our two models of stochastic ED to correctly estimate the parameters of a target model. If we are able to learn this target, this will mean that a learned tree ED will always outperform a classic tree ED with *a priori* hand-tuned costs. Actually, in the best case, the latters will be those of the learned matrix δ. In other words, this means that our learning algorithm will be efficient to deal with real-world applications.

The experimental setup is the following: First, we generate a target distribution defined by a theoretical matrix δ^* (describing either a joint or a conditional distribution). Then, we generate a sample of input trees according to a given input distribution. To build a learning set LS of tree pairs, we assign to each input instance an output tree. This one is generated using the input tree and the edit operations described by the target distribution δ^*. Note that in real world applications, such pairs would represent couples of similar instances (for example, pairs of (noisy, unnoisy) trees).

The aim is to learn δ^* from LS (constituted of a growing number of tree pairs) using both of our generative and discriminative models. To assess the effect of the input distribution on the learned model, we use different densities to generate the input trees. The performance criterion we use is the normalized distance between the target and the learned distributions.

In a first series of experiments, we focus on the generative model (*i.e.* a joint one). In this case, we build two sets of input trees. The first one is obtained using the marginal distribution of δ^* which is defined as follows: $\forall l \in \mathcal{L} \cup \{\lambda\}, \delta^*(l) = \sum_{l' \in \mathcal{L} \cup \{\lambda\}} \delta^*(l, l')$. The second one is generated using a random distribution. The chart of Fig.7(a) shows the results. As expected, the only one way to learn the target requires to use its marginal distribution to generate the input trees. The use of another (random) density leads to a bias, *i.e.* a large distance between the target and the learned model.

We use the same experimental setup during the second series of experiments aiming at learning a conditional target model. In this case, we tested three different input distributions (among them one is the marginal one). The chart of Fig.7(b) confirms that whatever the input distribution we use, our discriminative model is able to learn the target model.

5 Conclusion

In this paper, we proposed two original approaches for learning a *stochastic tree ED*. This is, as far as we know, the first attempt to learn such a distance specifically adapted to trees. In the first method, we modeled this distance as a joint distribution on tree pairs. This model has the advantage of having a small variance but is biased. Thus, it is suited for dealing with real applications where the instances are not numerous but describe well the underlying distribution. We also proposed to learn a stochastic edit distance from a conditional distribution that allows us to remove this bias. Such a way to proceed is interesting overall when the size of the learning set is sufficiently large, reducing then the variance of such a model. The experimental results confirm the interest of both approaches.

We plan to extend our work to stochastic models able to take into account edit costs varying according to the tree context. Actually, the cost of an edit operation can depend on the location where it occurs in the tree, that is not taken into account with our current structures. This implies to learn more complex models, such as stochastic tree transducers.

References

1. Bille, P.: A survey on tree edit distance and related problem. Theoretical Computer Science **337**(1-3) (2005) 217–239
2. Ristad, S., Yianilos, P.: Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(5) (1998) 522–532
3. Oncina, J., Sebban, M.: Learning stochastic edit distance: application in handwritten character recognition. Journal of Pattern Recognition (2006) to appear.
4. McCallum, A., Bellare, K., Pereira, P.: A conditional random field for discriminatively-trained finite-state sting edit distance. In: UAI2005. (2005)
5. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press (1998)
6. Neuhaus, M., Bunke, H.: A probabilistic approach to learning costs for graph edit distance. In: 17th Int. Conf. on Pattern Recognition, IEEE (2004) 389–393
7. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing (1989) 1245–1262
8. Klein, P.: Computing the edit-distance between unrooted ordered trees. In: Proc. of the 6th European Symposium on Algorithms (ESA), Springer (1998) 91–102
9. Selkow, S.: The tree-to-tree editing problem. Information Processing Letters **6**(6) (1977) 184–186
10. Bouchard, G., Triggs, B.: The trade-off between generative and discrminative classifiers. In: COMPSTAT'2004, Springer (2004)
11. Dempster, A., Laird, M., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc **B**(39) (1977) 1–38