A Discriminative Model of Stochastic Edit Distance in the form of a Conditional Transducer*

Marc Bernard, Jean-Christophe Janodet, Marc Sebban

EURISE, Université Jean Monnet de Saint-Etienne,
23, rue Paul Michelon, 42023 Saint-Etienne, France.
{marc.bernard,janodet,marc.sebban}@univ-st-etienne.fr

Abstract. Many real-world applications such as spell-checking or DNA analysis use the Levenshtein edit-distance to compute similarities between strings. In practice, the costs of the primitive edit operations (insertion, deletion and substitution of symbols) are generally hand-tuned. In this paper, we propose an algorithm to learn these costs. The underlying model is a probabilitic transducer, computed by using grammatical inference techniques, that allows us to learn both the structure and the probabilities of the model. Beyond the fact that the learned transducers are neither deterministic nor stochastic in the standard terminology, they are conditional, thus independant from the distributions of the input strings. Finally, we show through experiments that our method allows us to design cost functions that depend on the string context where the edit operations are used. In other words, we get kinds of *context-sensitive* edit distances.

Keywords. Edit Distance, Stochastic Transducers, Discriminative Models, Grammatical Inference.

1 Introduction

Real world applications such as spell checking, speech recognition, DNA analysis or plagiarism detection often use the Levenshtein distance, the so-called Edit Distance (ED) [12], to compute similarities of string pairs. The common feature of ED-based methods is that they are static, in the sense of using *a priori* fixed costs for the primitive edit operations (insertion, deletion, substitution), that leaves little room for adaptation to the string context. Nevertheless, in many real domains, the level of an edit cost should be able to depend not only on the pair of symbols handled but also on the context where the operation occurs. For instance, in computational biology, a given edit operation involving the same two symbols can highly depend on its location in the DNA sequence.

* This work was supported in part by the IST Programme of the European Community, under the Pascal Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

One solution would consist in manually assigning costs to edit operations that reflect the likelihood of the corresponding transformations. But the setting up of this strategy is difficult and seems to be not realistic overall for applications with a low level of expertise. Some recent work tried to overcome the previously mentioned drawbacks by automatically learning the primitive edit costs, rather than hand-tuning them for each domain. Several probabilistic models have been proposed to learn a *stochastic* ED in the form of stochastic transducers [9, 1, 8], conditional random fields (CRF) [7], or pair-Hidden Markov Models (pair-HMM) [5]. These models provide a probability distribution over the edit operations and thus over the string pairs. The stochastic ED between two sequences can then be computed from the negative logarithm of the probability of the string pair.

Although these methods have provided some significant improvements on pattern recognition tasks in comparison with the classic non-learned ED, they share at least one of the following two drawbacks (sometimes both). The first one is a *statistical bias* of the inferred model. Actually, the majority of these approaches aim at learning a *generative* model rather than a *discriminative* classifier [2]. In other words, they learn a joint probability distribution $p(x, y)$ over the string pairs (x, y), so the resulting conditional density $p(y|x)$, required in classification tasks, is a biased classifier that depends on the input distribution $p(x)$. A solution, as proposed in [7, 8], consists in directly learning a conditional distribution, called a *discriminative* classifier.

The second drawback is a *limitation on the expressive power* of the model. Actually, the structure of the learned model (*i.e.* the number of states in the transducer or in the CRF or in the pair-HMM) is always *a priori* fixed in the proposed approaches. The goal is to learn the parameters (the edit costs) assuming that the fixed structure is able to capture the most important configurations which can arise from the alignment of two sequences. Since determining such a structure depends on the domain, this often constitutes a tricky task that can result in a bad adaptation of the model to the string context.

In this paper, we propose to take into account both these problems, by learning not only the structure but also the parameters of a so-called *conditional edit transducer*. The motivations that justify the learning of such a transducer are the following. First, we think that an efficient way to model a stochastic ED actually consists in viewing it as a stochastic transduction between the input X and output Y alphabets [8, 9]. In other words, it means that the relation constituted by a set of (*input,output*) strings can be compiled in the form of a 2-tape automaton, called a *stochastic finite-state transducer*. The interpretation of the ED as a stochastic transduction naturally leads to two possible string distances [9]: the first one describes the most likely transduction between the two strings, while the second is defined by aggregating all transductions between them. In this paper, we focus on the first stochastic distance, a so-called *Viterbi Edit Distance* [9]. We motivate this choice by the fact that we will use an adaptation of the well-known Viterbi algorithm for learning the structure **and** the parameters of the *conditional edit transducer*.

Actually, stochastic transducers suffer from the lack of training algorithms [6] which generally only learn the parameters of an imposed structure, using the Expectation Maximization algorithm (EM) [4]. We claim in this paper that this drawback can be efficiently overcome using grammatical inference algorithms, that constitutes the second motivation of our work. Basically, a transduction between two strings $x \in X^*$ and $y \in Y^*$, in the specific domain of the ED, can be rewritten using an adapted Viterbi algorithm in the form of an optimal sequence of edit operations $z = z_1...z_n, z_i \in (X \cup \{\lambda\}) \times (Y \cup \{\lambda\}) \setminus \{(\lambda, \lambda)\}$ (where λ is the empty string). Thus, we can exploit grammatical inference algorithms for learning over this new alphabet the structure of the model (and its parameters) in the form of a probabilistic finite state automaton.

The rest of this paper is organized as follows. After some notations and definitions in Section 2 and 3, we propose an adaptive approach for learning a conditional edit transducer. This learning requires to find an optimal alignment of string pairs in the form of a set of edit operations (Section 4). From this new set of sequences built on the alphabet of edit operations, we infer a probabilistic automaton with ALERGIA [3]. To learn a discriminative model, the automaton is corrected to satisfy constraints of conditional distribution (Section 5). The conditional edit transducer is then deduced from the automaton by splitting each transition according to the input and output alphabets. In Section 6, we carry out several series of experiments showing the behavior of our learned ED in a comparative study.

2 On Edit Distances

An *alphabet* X is a finite nonempty set of symbols called *letters*. A *string* x over X is a finite sequence $x = a_1 a_2 \dots a_n$ of letters. Let $|x|$ denote the length of x, λ the empty string and X^* the set of all strings. In the sequel, we will use two (non necessarily) distinct alphabets X and Y whose respective strings will generally be indicated by $x = a_1 a_2 \dots a_n$ and $y = b_1 b_2 \dots b_m$, for sake of simplicity.

Let us recall that the *edit distance* is the smallest number of substitutions, insertions and deletions required to transform a string x into another y. More formally, let $E_s = X \times Y$ be the set of *substitutions*, $E_i = \{\lambda\} \times Y$ the set of *insertions*, $E_d = X \times \{\lambda\}$ the set of *deletions* and $Z = E_s \cup E_i \cup E_d$ the set of all edit operations: $Z = (X \cup \{\lambda\}) \times (Y \cup \{\lambda\}) \setminus \{(\lambda, \lambda)\}$. An element $(a, b) \in Z$ will be denoted $(a : b)$. Let $c : Z \to \mathbb{R}^+$ be a fixed *primitive cost function* that assigns a non negative weight to each edit operation. The *edit distance* $d(x, y)$ between two strings $x \in X^*$ and $y \in Y^*$ is recursively defined as follows:

$$d(x,y) = \min \begin{cases} 0 & \text{if } x = \lambda \text{ and } y = \lambda \\ c(a:b) + d(x',y') & \text{if } x = ax' \text{ and } y = by' \\ c(a:\lambda) + d(x',y) & \text{if } x = ax' \\ c(\lambda:b) + d(x,y') & \text{if } y = by'. \end{cases}$$

The edit distance can also be defined through the notion of alignment. Given two strings $x \in X^*$ and $y \in Y^*$, an *alignment* between x and y is a sequence

of edit operations, thus a string $z = (u_1 : v_1) \dots (u_p : v_p) \in Z^*$, such that (1) $u_i \in X \cup \{\lambda\}$ and $v_i \in Y \cup \{\lambda\}$ and $(u_i : v_i) \neq (\lambda : \lambda)$ for all $i \in 1..p$ and (2) $x = u_1 u_2 \dots u_p$ and $y = v_1 v_2 \dots v_p$. Let $L(x, y)$ denote the set of all alignments between x and y. Computing $d(x, y)$ consists in exhibiting an alignment of minimum cost between x and y: $d(x, y) = \min_{\{z \in L(x,y) : z = z_1 \dots z_p\}} c(z_1) + \dots + c(z_p)$. For instance, assuming that $X = Y = \{a, b\}$ and $c(a_i : b_j) = 1$ if $a_i \neq b_j$ and 0 otherwise, we get $d(aba, bab) = 2$ with $(a : \lambda)(b : b)(a : a)(\lambda : b)$ or $(\lambda : b)(a : a)(b : b)(a : \lambda)$ as minimal alignments.

Notice that both $d(x, y)$ and the alignments of minimum cost between x and y can be computed in $\mathcal{O}(|x| \cdot |y|)$ time using dynamic programming techniques [12]. However, faced with practical situations, the problem is generally not to efficiently compute the edit distance itself but rather to find a relevant primitive cost function able to capture the most important configurations which can arise from the alignments of two sequences. As we said in introduction, imposing a unique cost for a substitution of two letters, whatever they are, and not taking into account the location where this edit operation occurs in the alignment (that we call the string context), is not relevant for dealing with complex problems. We propose in the following to learn a suited probabilistic model, called a *conditional edit-transducer*, to take into account this string context.

3 On Conditional Edit-Transducers

Roughly speaking, a standard transducer is a finite state machine that takes strings from an input alphabet X and rewrites them into strings of an output alphabet Y. In the context of edit distance, every alignment between $x \in X^*$ and $y \in Y^*$ can be viewed as a rewrite derivation of x into y using the edit operations. So a finite state machine that would achieve this transduction is a special transducer, called an *edit transducer*, whose transitions are labelled from the unique alphabet Z of edit operations.

Definition 1. *A finite-state edit-transducer (*FSET*) is a 5-tuple* $\mathcal{A} = \langle Q, Z, i, F, T \rangle$ *such that* Q *is a finite set of states,* $Z = (X \cup \{\lambda\}) \times (Y \cup \{\lambda\}) \setminus \{(\lambda, \lambda)\}$ *the alphabet of edit operations,* $i \in Q$ *the initial state and* $F : Q \to [0, 1]$ *(resp.,* $T : Q \times Z \times Q \to [0, 1]$*) a function that assigns a weight to every state (resp., transition). We also assume the following* Determinism Condition*:* $\forall p \in Q, \forall (a : b) \in Z,\ Card(\{q : T(p, (a : b), q) > 0\}) \leq 1$.

A state is *final* iff $F(p) > 0$. Moreover, we will never consider the transitions $(p, (a : b), q)$ whose weights $T(p, (a : b), q)$ are null since they are not useful from a computational point of view. Indeed, computing the weight of an alignment $z = z_1 \dots z_n \in Z^*$ *w.r.t.* a FSET $\mathcal{A} = \langle Q, Z, i, F, T \rangle$, denoted $P(z|\mathcal{A})$, consists in finding a sequence of transitions $(i, z_1, p_1)(p_1, z_2, p_2) \dots (p_{n-1}, z_n, p_n)$ that starts from the initial state and is labeled with the letters of z. Due to the Determinism Condition, at most one such a path exists in $\mathcal{A}$ and then, the weight is: $P(z|\mathcal{A}) = T(i, z_1, p_1) \times T(p_1, z_2, p_2) \times \dots \times T(p_{n-1}, z_n, p_n) \times F(p_n)$.

Since we aim here at learning a discriminative model, let us now define a *conditional* FSET:

Definition 2. *A conditional finite-state edit-transducer (*CFSET*) is a* FSET $\mathcal{C} = \langle Q, Z, i, F, T\rangle$*, whose transitions are written* $(b|a)$ *rather than* $(a:b) \in Z$*, such that,* $\forall p \in Q, \forall a \in X$*,*

$$F(p) + \sum_{b\in Y, q\in Q} T(p, (b|\lambda), q) = 1, \quad (1)$$

$$\sum_{b\in Y, q\in Q} T(p, (b|a), q) + \sum_{q\in Q} T(p, (\lambda|a), q) + \sum_{b\in Y, q\in Q} T(p, (b|\lambda), q) = 1. \quad (2)$$

An example of CFSET is given in Fig.1(a). Basically, a CFSET $\mathcal{C}$ *is not* a DPFA [11] over Z^* since $F(p) + \sum_{(b|a)\in Z, q\in Q} T(p, (b|a), q) \neq 1$ in general. However, by using Constraints (1) and (2), we can show that for every fixed input string $x \in X^*$, $P(y|\mathcal{C}, x) = \sum_{z\in L(x,y)} P(z|\mathcal{C})$ defines a distribution over Y^*: $\sum_{y\in Y^*} P(y|\mathcal{C}, x) = 1$, that is the reason why we speak of *conditional* FSETs. A formal proof of this property, in the case of a CFSET *with only one state*, can be found in [8]. Below, we just give a hint of the general case on an example.

Let us fix $x = aa$ and consider the CFSET $\mathcal{C}$ of Fig.1(a). $\mathcal{C}$ can be used to produce strings of Y^* incrementally by following its transitions while consuming the letters of x. For instance, starting from the initial state 1, there are 3 cases: Either, one can produce a b (with a probability of 0.3) from nothing (insertion) by following the transition $(1, (b|\lambda), 2)$, and must then produce a string from state 2, by remembering that no letter of x was consumed. Or, one can make a substitution of an a by a b (with a probability of 0.5), by following the transition $(1, (b|a), 1)$, and must then produce a string from state 1, by remembering that one a of x was consumed, so that only one a remains in the input. Or, one can delete an a (with a probability of 0.2) in the input without producing anything (deletion), by following the transition $(1, (\lambda|a), 1)$, and must then produce a string from state 1 and only one a in input. When all the letters of the input string are consumed, no more substitution or deletion can be done, whatever the state. So one can only make last insertions before stopping. For instance, directed by $x = aa$ and $\mathcal{C}$, the string *bbbb* can be produced by the following path: $(1, (b|a), 1)(1, (b|\lambda), 2)(2, (b|a), 1)(1, (b|\lambda), 2)$.

More generally, we can build the automaton $\mathcal{P}$ that generates all the strings of Y^* following the transitions of $\mathcal{C}$ while consuming the letters of x (see Fig.1(b)). The states are pairs of the form $\langle k, \alpha\rangle$ where $k \in \{1, 2\}$ is a state of $\mathcal{C}$ and α is a prefix of $x = aa$ corresponding to the beginning of x that is already consumed, *i.e.*, $\alpha \in \{\lambda, a, aa\}$. The initial state is $\langle 1, \lambda\rangle$ since 1 is the initial state of $\mathcal{C}$ and no letter of x is initially read. A state $\langle k, \alpha\rangle$ is final iff (1) k is a final state in $\mathcal{C}$ and (2) all the letters of x have been consumed: $\alpha = x = aa$. The transitions are of the form $(\langle k, \alpha\rangle, (b_j|a_i), \langle l, \beta\rangle)$ and they appear iff (1) $(k, (b_j|a_i), l)$ is a transition in $\mathcal{C}$ and (2) $\beta = \alpha.a_i$, that is to say, $\beta = \alpha$ in the case of a insertion $(b_j|a_i) = (b|\lambda)$ and $\beta = \alpha.a$ in the case of a deletion $(b_j|a_i) = (\lambda|a)$ or a substitution $(b_j|a_i) = (b|a)$. Finally, the transitions and the final states come with the probabilities that are assigned by $\mathcal{C}$.

It is now clear that if we want $\mathcal{P}$ to generate a distribution over Y^*, then $\mathcal{P}$ must be a PFA [11], *i.e.*, for every state, the probability of the outgoing transitions

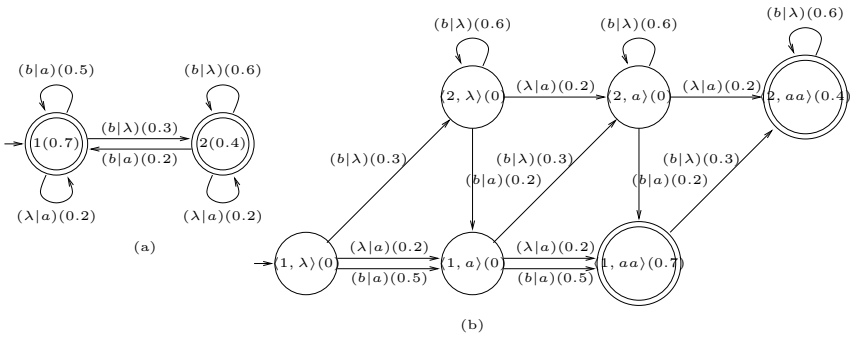

Fig. 1. (a) The CFSET $\mathcal{C}$. (b) The probabilistic automaton $\mathcal{P}$ modeling the distribution over Y^* conditionally to $\mathcal{C}$ and the input string $x = aa$.

plus the probability of this state to be final must be 1. This is exactly the statements of Constraints (1) and (2). Indeed, Constraint (1) concerns the case where all the letters in the input string x are consumed, thus tackles the final states of $\mathcal{P}$; the generation of the output string can only be done by using insertions, before stopping. Constraint (2) concerns the case where not all the letters in the input string x are consumed yet, thus tackles the non final states of $\mathcal{P}$; all the edit operations can be used to generate the output string, but this generation is forbidden to stop (since some letters of x remains to be consumed). Hence, Constraints (1) and (2) insure that for every fixed input string x, the construction of $\mathcal{P}$ yields a PFA, that brings: $\sum_{y \in Y^*} P(y|\mathcal{C}, x) = 1$.

At last but not least, by using the same terminology as that of [9], two edit distances can be defined from every CFSET $\mathcal{C}$, the *stochastic edit distance*: $d^s_{\mathcal{C}}(y|x) = -\log P(y|\mathcal{C}, x) = -\log\left(\sum_{z \in L(x,y)} P(z|\mathcal{C})\right)$ and the *Viterbi edit distance*: $d^v_{\mathcal{C}}(y|x) = -\log\left(\max_{z \in L(x,y)} P(z|\mathcal{C})\right)$. We will only consider the latter in the rest of the paper. Indeed, on the one hand, we propose in Section 4 an efficient algorithm that allows us to compute the Viterbi edit distance. On the other hand, we will need this algorithm to develop, in Section 5, a method to learn the structure and the parameters of a CFSET that maximizes the likelihood of a learning sample. So studying only the Viterbi edit distance allows us to kill two birds with one stone.

4 Computing the Viterbi Edit Distance from a CFSET

Given two strings $x \in X^*$ and $y \in Y^*$, a CFSET provides several possible alignments between x and y. For instance, if we consider that of Fig.1(a), then the strings $x = aa$ and $y = bbb$ may be aligned by $z_1 = (b|a)(b|a)(b|\lambda)$ or $z_2 = (b|\lambda)(b|a)(b|\lambda)(\lambda|a)$. Nevertheless, as $P(z_1|\mathcal{C}) = 0.5 \times 0.5 \times 0.3 \times 0.4 = 0.03$ and $P(z_2|\mathcal{C}) = 0.3 \times 0.2 \times 0.3 \times 0.2 \times 0.4 = 0.00144$, we deduce that z_1 may be the optimal alignment between x and y, unless there exists another alignment of higher probability, that is not the case of z_2.

Algorithm 1: Probability of an optimal alignment between x and y

Input: Two strings $x = a_1 \dots a_n \in X^*$ and $y = b_1 \dots b_m \in Y^*$ and a CFSET $\mathcal{C} = \langle Q, Z, 1, F, T \rangle$ whose states are $Q = \{1, \dots, |Q|\}$ and 1 is initial.

Output: Maximum probability of every alignment between x and y *w.r.t.* $\mathcal{C}$.

$M[0][0][1] \leftarrow 1$;
for $s = 2$ to $|Q|$ **do**
  $M[0][0][s] \leftarrow 0$;
for $i = 1$ to n **do**
  for $s = 1$ to $|Q|$ **do**
    $M[i][0][s] \leftarrow \max_{t \in Q} M[i-1][0][t] \times T(t, (\lambda|a_i), s)$;
for $j = 1$ to m **do**
  for $s = 1$ to $|Q|$ **do**
    $M[0][j][s] \leftarrow \max_{t \in Q} M[0][j-1][t] \times T(t, (b_j|\lambda), s)$;
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    for $s = 1$ to $|Q|$ **do**
      $m_{deletion} \leftarrow \max_{t \in Q} M[i-1][j][t] \times T(t, (\lambda|a_i), s)$;
      $m_{insertion} \leftarrow \max_{t \in Q} M[i][j-1][t] \times T(t, (b_j|\lambda), s)$;
      $m_{substitution} \leftarrow \max_{t \in Q} M[i-1][j-1][t] \times T(t, (b_j|a_i), s)$;
      $M[i][j][s] \leftarrow \max(m_{deletion}, m_{insertion}, m_{substitution})$;
return $\max_{s \in Q} M[n][m][s] \times F(s)$

Algo.1 allows us to compute the probability of an optimal alignment between two strings $x = a_1 \dots a_n$ and $y = b_1 \dots b_m$ *w.r.t.* a CFSET $\mathcal{C} = \langle Q, Z, 1, F, T \rangle$ by dynamic programming. It uses a 3-dimension matrix M to store the probabilities according to (1) input and output strings (that is similar to the standard edit distance) and (2) the state in progress (that is new). More precisely, M's dimensions are $(0..|x|) \times (0..|y|) \times (1..|Q|)$ and $M[i][j][s]$ contains the probability of an optimal alignment, that reaches the state s, between the prefix $a_1 \dots a_i$ of x and the prefix $b_1 \dots b_j$ of y.

In Fig.2, we show an example of execution when $x = aa$ and $y = bbb$ and the CFSET $\mathcal{C}$ is that of Fig.1(a). Our algorithm fills up M, cell after cell. Let us focus on the computation of $M[2][1][1]$ (whose result, 0.1, is underlined in Fig.2). This cell must contain the maximum probability, according to $\mathcal{C}$, to reach state 1 after having consumed aa in input and produced b in output. Several possibilities can lead to this situation. (1) One can reach state 1 by making a deletion of an a in input, starting from a cell, in state 1 or 2, where a had been consumed in input and b produced in output, thus from the cells $M[1][1][1]$ or $M[1][1][2]$. By Algo.1, we get:

$$m_{deletion} = \max_{t \in Q} M[1][1][t] \times T(t, (\lambda|a), 1) = 0.1$$

(2) One can reach state 1 by making an insertion of a b in output, starting from a cell where aa had been consumed in input and no b produced in output, thus

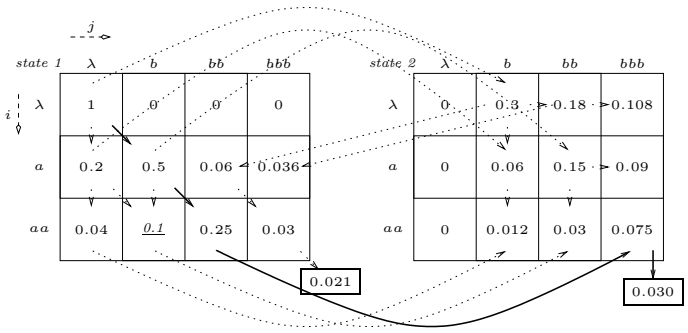

Fig. 2. Computation of $d_{\mathcal{C}}^{v}(y|x)$ when $x = aa$, $y = bbb$ *w.r.t.* the CFSET $\mathcal{C}$ of Fig.1(a). Each cell $M[i][j][s]$ contains the probability of an optimal alignment, that reaches the state s, between the prefix $a_1 \dots a_i$ of x and the prefix $b_1 \dots b_j$ of y. The arrows are useful to re-construct all the possible alignments between x and y. Both cells out of the arrays contain the probabilities of such alignments. As $0.030 > 0.021$, we deduce that $d_{\mathcal{C}}^{v}(bbb|aa) = -\log 0.03 \simeq 1.52$ thanks to the optimal alignment $(b|a)(b|a)(b|\lambda)$.

from the cells $M[2][0][1]$ or $M[2][0][2]$. By Algo.1, we get:

$$m_{insertion} = \max_{t \in Q} M[2][0][t] \times T(t, (b|\lambda), 1) = 0.$$

(3) One can reach state 1 by making the substitution of a b by an a, starting from a cell where a had been consumed in input and λ produced in output, thus from the cells $M[1][0][1]$ and $M[1][0][2]$. By Algo.1, we get:

$$m_{substitution} = \max_{t \in Q} M[1][0][t] \times T(t, (b|a), 1) = 0.1$$

Therefore, $M[2][1][1] = 0.1$. Moreover, this score is achieved after either a deletion $(\lambda|a)$ from $M[1][1][1]$ or a substitution $(b|a)$ from $M[1][0][0]$, what we indicate in Fig.2 with the dotted arrows that point to the cell $M[2][1][1]$.

At the end of the loops, all the letters of x are consumed and all those of y are produced. So Algo.1 returns:

$$\max_{s \in Q} M[2][3][s] \times F(s) = \max(M[2][3][1] \times F(1), M[2][3][2] \times F(2)) = 0.03.$$

So $d_{\mathcal{C}}^{v}(bbb|aa) = -\log 0.03 \simeq 1.52$. Moreover, since we have stored the best edit operations in M, we deduce that only one alignment is optimal: $(b|a)(b|a)(b|\lambda)$.

5 Learning an Optimal CFEST

The second task we have to tackle concerns the learning of the CFSET. As we said in introduction, stochastic transducers suffer from the lack of learning algorithm. The main reason comes from the fact that stochastic transducers are not deterministic with respect to the input strings. We are going to show in this section that the specific context of learning stochastic edit distance leaves

room for learning not only the parameters but also the structure of a CFSET. The strategy of our iterative algorithm is based on the following remarks.

By Def.2, an edit transducer over $X^* \times Y^*$ is a kind of probabilistic automaton over Z^*. So our point is that learning a CFSET ultimately returns to the problem of learning a PFA over Z^*. Indeed, if we can replace each string pair (x, y) of the learning sample by a sequence z of edit operations corresponding to the most probable alignment between x and y, then we will be able to learn a DPFA modeling the Viterbi edit distance with usual grammatical inference algorithms (such as ALERGIA [3] or MDI [10]). Fortunately, Algo.1 can provide us with such optimal alignments. However, learning a probabilistic model in the form of a DPFA from the alignments will provide us with a *generative* model, that is to say, a joint distribution over $X^* \times Y^*$. In order to learn a *discriminative* model, we have to re-normalize the current distribution at each iteration.

Algorithm 2: Learning the optimal CFSET

Input: A sample $LS = \{(x_k, y_k) : x_k \in X^*, y_k \in Y^*, k \in 1..n\}$ of string pairs.
Output: A CFSET $\mathcal{C}$.
$\mathcal{C} \leftarrow$ a random CFSET;
repeat
  let z_k be the most probable alignment of x_k and y_k *w.r.t.* $\mathcal{C}$ for all $k \in 1..n$;
  $\mathcal{A} \leftarrow \text{ALERGIA}(\{z_1, \ldots, z_n\})$;
  $\mathcal{C} \leftarrow \text{NORMALIZATION}(\mathcal{A}, \{\gamma(p, z, q) : p, q \in Q, z \in Z \cup \{(\lambda : \lambda)\}\})$;
until $\left(\sum_{k=1}^{n} d_{\mathcal{C}}^{v}(y_k|x_k)\right)$ does not decrease anymore;
return $\mathcal{A}$;

The pseudo-code of our learning algorithm is presented in Algo.2. We initialize our model to a random CFSET. Then, we run the following iterative estimation procedure. We use Algo.1 and the current CFSET for assigning the most probable alignment z_k to each string pair $(x_k, y_k) \in LS$. Then, we run ALERGIA for learning a DPFA over Z^*. Once the learning is achieved, we re-normalize the joint distribution described by this generative model to fulfill Constraints (1) and (2) of Def.2 and get a CFSET. The algorithm loops until the Viterbi edit distances computed on the learning sample does not decrease (significantly) anymore. This stopping criterium is equivalent to maximize the likelihood $(\prod_{k=1}^{n} P(y_k|\mathcal{C}, x_k))$ over LS, due to the definition of the distance.

To achieve the normalization, we must know the number of times each transition (p, z, q) of the DPFA $\mathcal{A}$ has been used by the learning sequences. These values, that are denoted $\gamma(p, z, q)$, are either directly returned by the inference algorithm (that is the case of ALERGIA), or must be computed by parsing again the learning sample. By convention, $\gamma(p, (\lambda : \lambda), p)$ denotes the number of times the parsing of any learning string ends in the state p. Once the frequencies $\gamma(p, z, q)$ are known, we can re-normalize each probability $T(p, z, q)$ of $\mathcal{A}$ that is the aim of Algo.3.

Algorithm 3: Normalization fulfilling Constraints (1) and (2)

Input: A DPFA $\mathcal{A} = \langle Q, Z, i, F, T \rangle$ with the set of frequencies $\gamma(p, z, q)$.
Output: The corresponding CFSET after normalization.
for each $p \in Q$ **do**
  $N_p \leftarrow \sum_{q \in Q} \sum_{z \in Z \cup \{(\lambda:\lambda)\}} \gamma(p, z, q)$;
  $N_p(\lambda) \leftarrow \sum_{q \in Q} \sum_{b_j \in Y} \gamma(p, (\lambda : b_j), q)$;
  $N_p(a_i) \leftarrow \sum_{q \in Q} \sum_{b_j \in Y \cup \{\lambda\}} \gamma(p, (a_i : b_j), q), \forall a_i \in X$;
  $\delta_p \leftarrow 1 - (N_p(\lambda)/N_p)$;
  for all $a_i \in X, b_j \in Y, q \in Q$ **do**
    $T(p, (b_j|\lambda), q) \leftarrow \gamma(p, (\lambda : b_j), q)/N_p$;
    $T(p, (\lambda|a_i), q) \leftarrow \gamma(p, (a_i : \lambda), q) \times \delta_p/N_p(a_i)$;
    $T(p, (b_j|a_i), q) \leftarrow \gamma(p, (a_i : b_j), q) \times \delta_p/N_p(a_i)$;
  $F(p) \leftarrow \delta_p$;
return $\mathcal{A}$;

It is easy to check that this algorithm is sound: it suffices to verify that Constraints (1) and (2) are satisfied by all the states after normalization. This proof is presented in detail in [8] in the case of a CFSET with only state. It basically also applies to the case of several states since only local information to the states has to be considered, and no information concerning the relations between the states. Notice also that this normalization was proved to be optimal in the framework of an EM procedure [8].

6 Experiments

6.1 Application in Pattern Recognition on the NIST Database

To assess the performance of our algorithm on a pattern recognition task, we run it on the real world problem of handwritten digit classification. To achieve this task, we used a subset of the well-known NIST Database of the National Institute of Standards and Technology. The digits of this database are described in the form of 128 $\times$ 128 bitmap images written by 100 different writers. In our experimental setup, for simplifying our process, we reduced the size of the bitmaps to 16 $\times$ 16 images.

Then, we used a growing number of these digits as learning sample LS, and we kept 1,000 digits in a test sample TS. Since stochastic transducers handle strings, we encoded each digit in an octal form, according to a feature extraction strategy consisting in using Freeman codes for transforming the original vector in an octal string. Fig.3 describes the strategy from a sample of the class "9".

For learning our CFSET, we need a learning set of string pairs. We follow the strategy proposed in [9] consisting in building pairs of "similar" strings that describe the possible distortions between instances of each class $(0 \dots 9)$. It is possible to automatically build such pairs of (input,output) strings, where an input is a learning string of LS, and the output is a prototype of the input.

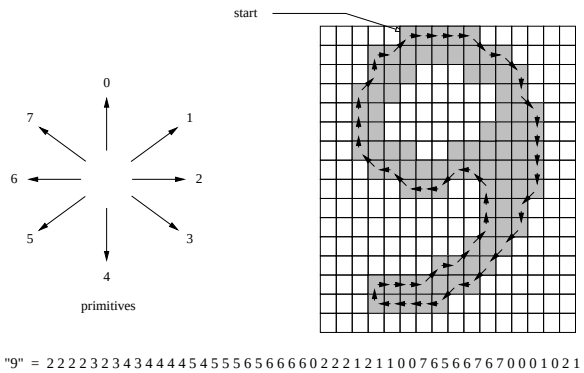

Fig. 3. Example of string coding character. Starting from the next found pixel (scanning the digit left-to-right from the top), the coding algorithm builds a string with absolute direction of the next pixel in the border.

To achieve this task, we used as prototype the corresponding 1-nearest-neighbor in LS (using the classic edit distance with the same edit cost for an insertion, deletion or a substitution) of each input.

Note that we could have used other ways to construct string pairs. A solution would be to generate all pairs in the same class. Beyond large complexity costs, this strategy would not be relevant in such a digit recognition task. Actually, the classes of digits are intrinsically multimodal. For example, a zero can be written either with an open loop or a closed one. In this case, the string that represents an "open" zero cannot be considered as a distortion of a "closed" zero, but rather as a different manner (a sort of sub-class) to design this digit. Therefore, a nearest-neighbor based strategy seems to be much more relevant.

We aim at showing with this series of experiments that learning the primitive edit costs of an edit distance in the form of a CFSET is more relevant than imposing these costs in advance. Thus, we will compare our approach with the classic edit distance. The experimental setup is the following: (1) Each set i of digits ($i = 0, .., 9$) is divided in 2 parts: a learning set LS_i and a test set TS_i. (2) From each LS_i, we build a set of string pairs PS_i in the form of $(x, NN(x)), \forall x \in LS_i$, where $NN(x) = \text{argmin}_{y \in LS_i - \{x\}} d(x, y)$ (d is the classic edit distance). (3) We learn a CFSET $\mathcal{C}$ from $\bigcup_i PS_i$ with our approach. (4) We classify each test digit $x' \in \bigcup_i TS_i$ by (a) the class i of the learning string $y \in \bigcup_i LS_i$ minimizing $d^v_{\mathcal{C}}(y|x')$ and (b) the class i of its nearest-neighbor $NN(x') \in \bigcup_i LS_i$.

Using the previous experimental setup, we can then compare the two approaches under exactly the same conditions. In order to assess each algorithm in different configurations, the number of learning strings varied from 50 (5 for each class of digits) to 500 (50 for each class), with a step of 50 strings per class. The test accuracy was computed with a test set containing always 1,000 strings (*i.e.* $|\bigcup_i TS_i| = 1,000$). The chart of Fig.4(a) shows the results of our experiments on the NIST database. As already shown in [8], learning an ED in the form of a *conditional* edit transducer is clearly relevant to achieve a pattern recognition task. Whatever the size of the learning set, the error rate obtained using a classic

edit distance is always higher than that obtained by using a CFSET. Note that even if theoretically, we expect the two methods to converge to the same rate when $|LS| \to \infty$, it means that our method needs less learning examples to reach the same error rate.

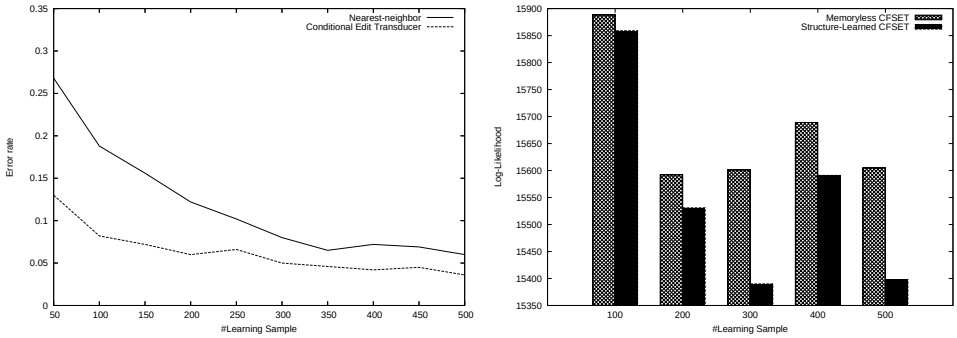

Fig. 4. 4(a)Results on the NIST database. 4(b) Comparison between memoryless CFSETs and CFSETs whose structure was learned.

6.2 Results on an Artificial Database

In this second series of experiments, we aim at showing the advantage of learning the parameters *and the structure* of a conditional tranducer with respect to other approaches that fix the structure. To this purpose, we focus on the algorithm by Oncina and Sebban that learns conditional memoryless edit-tranducers, that is to say, CFSETs with only one state. We use a random FSET with 5 states whose input and output alphabets are $X = Y = \{a, b, c, d\}$. Thus, every state has 24 outgoing transitions, labeled by the edit operations in $Z = (X \cup \{\lambda\}) \times (Y \cup \{\lambda\}) \setminus \{(\lambda, \lambda)\}$. Then we use this FSET to generate 1,000 strings over Z^*, that allow us to deduce 1000 string pairs over $X^* \times Y^*$. Half of them are used by both the algorithms to learn: the learning sample varies from 100 to 500 strings, with a step of 100 strings. The 500 other strings constitute the test set (TS). In order to measure the performance of both methods, we compute $\sum_{(x,y)\in TS} d^v_{\mathcal{C}}(y|x)$. The less this measure is, the best the CFSET is.

Fig.4(b) shows our results. From this histogram, we observe that a memoryless CFSET is systematically less powerfull than a CFSET whose structure was learned. Notice that the best results were obtained with 300 and 500 learning examples by transducers that had exactly 5 states, so that were relatively close to the target generating model. This result confirms that our method is able to capture the sensitivity of the edit costs to the string context, that is obviously not the case of a memoryless transducer with fixed costs.

7 Conclusion

In this paper, we propose a new algorithm to learn the cost function of a stochastic edit distance. Our method relies on conditional edit transducers whose parameters *and* structure are learned, thanks to grammatical inference techniques. Those transducers inherit all the advantages of conditional models described by Oncina and Sebban in [8]. Moreover, our experiments show that learning the structure allows us to overcome the memoryless transducers, since many-states transducers model complex edit cost functions that take into account the string-context where they are used.

References

1. M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. of the 9th Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, pages 39–48, 2003.
2. G. Bouchard and B. Triggs. The tradeoff between generative and discriminative classifiers. In J. Antoch, editor, *Proc. in Computational Statistics (COMPSTAT'04), 16th Symp. of IASC*, volume 16, Prague, 2004. Physica-Verlag.
3. R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proc. of 1st Int. Colloquium in Grammatical Inference (ICGI'94)*, pages 139–150. LNAI 862, 1994.
4. A. Dempster, M. Laird, and D. Rubin. Maximun likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, B(39):1–38, 1977.
5. R. Durbin, S.R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis.* Cambridge University Press, 1998.
6. J. Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, July 2002.
7. A. McCallum, K. Bellare, and P. Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In *Proc. 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 388–400, Arlington, Virginia, 2005. AUAI Press.
8. J. Oncina and M. Sebban. Learning stochastic edit distance: application in handwritten character recognition. *Journal of Pattern Recognition*, to appear, 2006.
9. E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998.
10. F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA inference using kullback-leibler divergence and minimality. In *Proc. 17th Int. Conf. on Machine Learning (ICML'00)*, pages 975–982. Morgan Kaufmann, San Francisco, CA, 2000.
11. E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite-state machines. *IEEE Trans. in Pattern Analysis and Machine Intelligence*, 27(7):1013–1039, 2005.
12. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.