# Learning Stochastic Edit Distance: application in handwritten character recognition ⋆

Jose Oncina [a,1] Marc Sebban [b,2]

[a] *Departamento de Lenguajes y Sistemas Informaticos, Universidad de Alicante, E-03071 Alicante (Spain)*

[b] *EURISE, Université de Saint-Etienne, 23 rue du Docteur Paul Michelon, 42023 Saint-Eienne (France)*

**Abstract**

Many pattern recognition algorithms are based on the nearest neighbour search and use the well known edit distance, for which the primitive edit costs are usually fixed in advance. In this article, we aim at learning an *unbiased* stochastic edit distance in the form of a finite-state transducer from a corpus of (*input,output*) pairs of strings. Contrary to the other standard methods, which generally use the Expectation Maximisation algorithm, our algorithm learns a transducer independently on the marginal probability distribution of the *input* strings. Such an unbiased way to proceed requires to optimise the parameters of a *conditional* transducer instead of a *joint* one. We apply our new model in the context of handwritten digit recognition. We show, carrying out a large series of experiments, that it always outperforms the standard edit distance.

*Key words:* Stochastic Edit Distance, Finite-State Transducers, Handwritten character recognition

⋆ 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.

*Email addresses:* oncina@dlsi.ua.es (Jose Oncina ), marc.sebban@univ-st-etienne.fr (Marc Sebban ).

[1] The author thanks the Generalitat Valenciana for partial support of this work through project GV04B-631, this work was also supported by the Spanish CICyT trough project TIC2003-08496-C04, partially supported by the EU ERDF.

[2] This work was done when the author visited the Departamento de Lenguajes y Sistemas Informáticos of the University of Alicante, Spain. The visit was sponsored by the Spanish Ministry of Education trough project SAB2003-0067.

## 1 Introduction

Many applications dealing with sequences require to compute the similarity of a pair (*input,output*) of strings. A widely-used similarity measure is the well known *edit distance*, which corresponds to the minimum number of operations, *i.e. insertions, deletions*, and *substitutions*, required to transform the *input* into the *output*. If this transformation is based on a random phenomenon and then on an underlying probability distribution, edit operations become random variables. We call then the resulting similarity measure, the *stochastic edit distance*.

An efficient way to model this distance consists in viewing it as a stochastic transduction between the input and output alphabets [1]. In other words, it means that the relation constituted by the set of (*input,output*) strings can be compiled in the form of a 2-tape automaton, called a *stochastic finite-state transducer*. Such a model is able to assign a probability at each new pair of strings, and could be then very useful to tackle many problems based on edit operations, such as segmentation, DNA alignment, classification, noisy channel decoding, or more generally to handle noise in sequences. Concerning this last case, note that Sakakibara and Siromomey have characterised in [2] what they call *edit noise*, *i.e.* the result of the corruption of an input string (into an output one) by random errors of edit operations. In such a context, learning a transducer providing a probability to each couple (*input,output*) of sequences would be very useful in domains where the presence of noise has dramatic effects on the quality of the inferred models. This is the case in grammatical inference, for instance, which requires either to remove or correct noisy data to avoid overfitting phenomena. More generally, the main problem does not consist in finding domains where such a model of stochastic edit distance could be efficiently used, but rather in estimating the parameters of the transducer itself. Actually, stochastic finite-state transducers suffer from the lack of a training algorithm. To the best of our knowledge, the first published algorithm to automatically learn the parameters of a stochastic transducer has been proposed by Ristad and Yianilos [3,1]. They provide a stochastic model which allows us to learn a stochastic edit distance, in the form of a memoryless transducer (*i.e.* with only one state), from a corpus of similar examples, using the Expectation Maximisation (EM) algorithm. During the last few years, the algorithm EM has also been used for learning other transducer-based models [4–6].

Ristad and Yianilos define the stochastic edit distance between two strings $x$ and $y$ as (the minus logarithm of) the *joint* probability of the pair $(x, y)$. In this paper, we claim that it would be much more relevant to express the stochastic edit distance from a *conditional* probability.

First, in order to correctly compute the edit distance, we think that the probabilities of edit operations over a symbol must be independent of those computed over another symbol. In other words, if the transformation of a string $x$ into another one

$y$ does not require many edit operations, it is expected that the probability of the substitution of a symbol by itself should be high. But, as the sum of the probabilities of all edit operations is one, then the probability of the substitution of another symbol by itself can not obviously be too large. Thus, by using a joint distribution (summing to 1), one generates an awkward dependence between edit operations.

Moreover, we think that the primitive edit costs of the edit distance must be independent of the *a priori* distribution $p(x)$ of the input strings. However, $p(x)$ can be directly deduced from the joint distribution $p(x, y)$, as follows: $p(x) = \sum_{y \in Y^*} p(x, y)$, where $Y^*$ is the set of all finite strings over the output alphabet $Y$. This means that this information is totally included in the joint distribution. By defining the stochastic edit distance as a function of the joint probability, as done in [1], the edit costs are then dependent of $p(x)$. However, if we use a conditional distribution, this dependence is removed, since it is impossible to obtain $p(x)$ from $p(y|x)$ alone.

Finally, although it is sensible and practical to model the stochastic edit distance by a memoryless transducer, it is possible that the *a priori* distribution $p(x)$ may not be modelled by such a very simple structure. Thus, by learning a transducer defining the joint distribution $p(x, y)$, its parameters can converge to compromise values and not to the true ones. This can have dramatic effects from an application standpoint. Actually, a widely-used solution to find an optimal output string $y$ according to an input one $x$ consists in first learning the joint distribution transducer and later deducing the conditional transducer dividing by $p(x)$ (more precisely by its estimates over the learning set). Such a strategy is then irrelevant for the reason we mentioned above.

In this paper we have developed a way to learn directly the conditional transducer. After some definitions and notations (Section 2), we introduce in Section 3 the learning principle of the stochastic edit distance proposed by Ristad and Yianilos [3,1]. Then, by simulating different theoretical joint distributions, we show that the *unique way*, using their algorithm, to find them consists in sampling a learning set of $(x, y)$ pairs according to the marginal distribution (*i.e.* over the input strings) of the target joint distribution itself. Moreover, we show that for all other *a priori* distribution, the difference between the target and the learned models increases. To free the method from this bias, one must *directly* learn at each iteration of the algorithm EM the conditional distribution $p(y|x)$. Achieving this task requires to modify Ristad and Yianilos's framework. That is the goal of Section 4. Finally, in Section 5, we carry out a large series of experiments on handwritten digit recognition. Through more than 1 million of tests, we show the relevance of our new model in comparison with the standard edit distance.

## 2 Classic String Edit Distance

An alphabet $X$ is a finite nonempty set of symbols. $X^*$ denotes the set of all finite strings over $X$. Let $x \in X^*$ be an arbitrary string of length $|x|$ over the alphabet $X$. In the following, unless stated otherwise, symbols are indicated by $a, b, \ldots$, strings by $u, v, \ldots, z$, and the empty string by $\epsilon$. $\mathbb{R}^+$ is the set of non negative reals. Let $f(\cdot)$ be a function, from which $[f(x)]_{\pi(x,\ldots)}$ is equal to $f(x)$ if the predicate $\pi(x, \ldots)$ holds and 0 otherwise, where $x$ is a (set of) dummy variable(s).

A string edit distance is characterised by a triple $(X, Y, c_e)$ consisting of the finite alphabets $X$ and $Y$ and the primitive cost function $c_e : E \rightarrow \mathbb{R}^+$ where $E = E_s \cup E_d \cup E_i$ is the alphabet of primitive edit operations, $E_s = X \times Y$, is the set of substitutions, $E_d = X \times \{\epsilon\}$ is the set of deletions, $E_i = \{\epsilon\} \times Y$ is the set of insertions. Each such triple $(X, Y, c_e)$ induces a distance function $d : X^* \times Y^* \rightarrow \mathbb{R}^+$ that maps a pair of strings to a non negative real value. The edit distance $d(x, y)$ between two strings $x \in X$ and $y \in Y$ is defined recursively as:

$$d(x,y) = \min \begin{cases} [c_e(a,b) + d(x',y')]_{x=x'a \wedge y=y'b} \\ [c_e(a,\epsilon) + d(x',y)]_{x=x'a} \\ [c_e(\epsilon,y) + d_c(x,y')]_{y=y'b} \end{cases}$$

Note that $d(x, y)$ can be computed in $O(|x| \cdot |y|)$ time using dynamic programming.

## 3 Stochastic Edit Distance and Memoryless Transducers

If the edit operations are achieved according to a random process, the edit distance is then called the *stochastic edit distance*, and noted $d_s(x, y)$. Since the underlying probability distribution is unknown, one solution consists in learning the primitive edit costs by means of a suited model. In this paper, we used memoryless transducers. Transducers are currently used in many applications ranging from lexical analysers, language and speech processing, etc. They are able to handle large amount of data, in the form of pairs of $(x, y)$ sequences, in a reasonable time complexity. Moreover, assuming that edit operations are randomly and independently achieved (that is the case in the edit noise [2]), a memoryless transducer is sufficient to model the stochastic edit distance.

### 3.1 Joint Memoryless Transducers

A joint memoryless transducer defines a joint probability distribution over the pairs of strings. It is denoted by a tuple $(X, Y, c, \gamma)$ where $X$ is the input alphabet, $Y$ is the output alphabet, $c$ is the *primitive* joint probability function, $c : E \rightarrow [0, 1]$ and $\gamma$ is the probability of the termination symbol of a string. As $(\epsilon, \epsilon) \notin E$, in order to simplify the notations, we are going to use $c(\epsilon, \epsilon)$ and $\gamma$ as synonyms.

Let us assume for the moment that we know the probability function $c$ (in fact, we will learn it later). We are then able to compute the joint probability $p(x, y)$ of a pair of strings $(x, y)$. Actually, the joint probability $p : X^* \times Y^* \rightarrow [0, 1]$ of the strings $x, y$ can be recursively computed by means of an auxiliary function (forward) $\alpha : X^* \times Y^* \rightarrow \mathbb{R}^+$ as:

$$\begin{aligned}\alpha(x, y) = \quad & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(a, b) \cdot \alpha(x', y')]_{x=x'a \wedge y=y'b} \\ & + [c(a, \epsilon) \cdot \alpha(x', y)]_{x=x'a} \\ & + [c(\epsilon, b) \cdot \alpha(x, y')]_{y=y'b}.\end{aligned}$$

And then,

$$p(x, y) = \alpha(x, y)\gamma.$$

In a symmetric way, $p(x, y)$ can be recursively computed by means of an auxiliary function (backward) $\beta : X^* \times Y^* \rightarrow \mathbb{R}^+$ as:

$$\begin{aligned}\beta(x, y) = \quad & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(a, b) \cdot \beta(x', y')]_{x=ax' \wedge y=by'} \\ & + [c(a, \epsilon) \cdot \beta(x', y)]_{x=ax'} \\ & + [c(\epsilon, b) \cdot \beta(x, y')]_{y=by'}.\end{aligned}$$

And then,

$$p(x, y) = \beta(x, y)\gamma.$$

Both functions (forward and backward) can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique. This model defines a probability distribution over the pairs $(x, y)$ of strings. More precisely,

$$\sum_{x \in X^*} \sum_{y \in Y^*} p(x, y) = 1,$$

that is achieved if the following conditions are fulfilled [1],

$$\gamma > 0, c(a,b), c(\epsilon,b), c(a,\epsilon) \geq 0 \quad \forall a \in X, b \in Y$$
$$\sum_{\substack{a \in X \cup \{\epsilon\} \\ b \in Y \cup \{\epsilon\}}} c(a,b) = 1$$

Given $p(x,y)$, we can then compute, as mentioned in [1], the stochastic edit distance between $x$ and $y$. Actually, the stochastic edit distance $d_s(x,y)$ is defined as being the negative logarithm of the probability of the string pair $p(x,y)$ according to the memoryless stochastic transducer.

$$d_s(x,y) = -\log p(x,y), \forall x \in X^*, \forall y \in Y^*$$

In order to compute $d_s(x,y)$, a remaining step consists in learning the parameters $c(a,b)$ of the memoryless transducer, *i.e.* the primitive edit costs.

### *3.2 Optimisation of the parameters of the joint memoryless transducer*

Let $S$ be a finite set of $(x,y)$ pairs of *similar* strings. Ristad and Yianilos [1] propose to use the expectation-maximisation (EM) algorithm to find the optimal joint stochastic transducer. The EM algorithm consists in two steps (expectation and maximisation) that are repeated until a convergence criterion is achieved.

Given an auxiliary $(|X|+1) \times (|Y|+1)$ matrix $\delta$, the expectation step can be described as follows: $\forall a \in X, b \in Y$,

$$\delta(a,b) = \sum_{(xax',yby') \in S} \frac{\alpha(x,y)c(a,b)\beta(x',y')\gamma}{p(xax',yby')}$$
$$\delta(\epsilon,b) = \sum_{(xx',yby') \in S} \frac{\alpha(x,y)c(\epsilon,b)\beta(x',y')\gamma}{p(xx',yby')}$$
$$\delta(a,\epsilon) = \sum_{(xax',yy') \in S} \frac{\alpha(x,y)c(a,\epsilon)\beta(x',y')\gamma}{p(xax',yy')}$$
$$\delta(\epsilon,\epsilon) = \sum_{(x,y) \in S} \frac{\alpha(x,y)\gamma}{p(x,y)} = |S|,$$

and the maximisation:

$$c(a,b) = \frac{\delta(a,b)}{N} \quad \forall a \in X \cup \{\epsilon\}, \forall b \in Y \cup \{\epsilon\}$$

| $c^*(a,b)$ | $\epsilon$ | a | b | c | d | $c^*(a)$ |
|---|---|---|---|---|---|---|
| $\epsilon$ | 0.00 | 0.05 | 0.08 | 0.02 | 0.02 | 0.17 |
| a | 0.01 | 0.04 | 0.01 | 0.01 | 0.01 | 0.08 |
| b | 0.02 | 0.01 | 0.16 | 0.04 | 0.01 | 0.24 |
| c | 0.01 | 0.02 | 0.01 | 0.15 | 0.00 | 0.19 |
| d | 0.01 | 0.01 | 0.01 | 0.01 | 0.28 | 0.32 |

Table 1
Target joint distribution $c^*(a,b)$ and its corresponding marginal distribution $c^*(a)$.

where

$$N = \sum_{\substack{a \in X \cup \{\epsilon\} \\ b \in Y \cup \{\epsilon\}}} \delta(a,b).$$

### *3.3 Limits of Ristad and Yianilos's algorithm*

To analyse the ability of Ristad and Yianilos's algorithm to correctly estimate the parameters of a target joint memoryless transducer, we have implemented it and carried out a series of experiments. Since the joint distribution $p(x,y)$ is a function of the learned edit costs $c(a,b)$, we only focused here on the function $c$ of the transducer.

The experimental setup was the following. We simulated a target joint memoryless transducer from the alphabets $X = Y = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, such as $\forall a \in X \cup \{\epsilon\}, \forall b \in Y \cup \{\epsilon\}$, the target model is able to return the primitive theoretical joint probability $c^*(a,b)$. The target joint distribution we used is described in Table 1 [3]. The marginal distribution $c^*(a)$ can be deduced from this target such that: $c^*(a) = \sum_{b \in X \cup \{\epsilon\}} c^*(a,b)$.

Then, we sampled an increasing set of learning input strings (from 0 to 4000 sequences) of variable length generated from a given probability distribution $p(a)$ over the input alphabet $X$. In order to simplify, we modelled this distribution in the form of an automaton with only one state [4] and $|X|$ output transitions with randomly chosen probabilities satisfying that $\sum_{a \in X} p(a) + p(\#) = 1$, where $p(\#)$ corresponds to the probability of a termination symbol of a string (see Figure 1).

---

[3] Note that we carried out many series of experiments with various target joint distributions, and all the results we obtained follow the same behaviour as the one presented in this section.

[4] Here also, we tested other configurations leading to the same results.

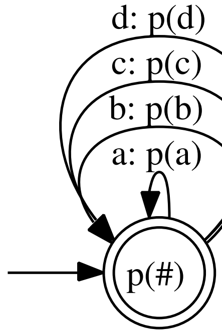


Fig. 1. Automaton used for generating the input sequences. The probability $p(\#)$ corresponds to the probability of a termination symbol of a string, or in other words the probability of the state to be final.

We used different settings for this automaton to analyse the impact of the input distribution $p(a)$ on the learned joint model. Then, given an input sequence $x$ (generated from this automaton) and the target joint distribution $c^*(a,b)$, we sampled a corresponding output $y$. Finally, the set $S$ of generated $(x,y)$ pairs was used by Ristad and Yianilos's algorithm to learn an estimated primitive joint distribution $c(a,b)$.

We compared the target and the learned distributions to analyse the behaviour of the algorithm to correctly assess the parameters of the target joint distribution. We computed an average difference between the both, defined as follows:

$$d(c,c^*) = \frac{\sum_{a \in X \cup \{\epsilon\}} \sum_{b \in Y \cup \{\epsilon\}} |c(a,b) - c^*(a,b)|}{2}$$

Normalised in this way, $d(c,c^*)$ is a value in the range $[0,1]$. Figure 2 shows the behaviour of this difference according to various configurations of the automaton of Figure 1. We can note that the unique way to converge towards a difference near from 0 consists in using the marginal distribution $c^*(a)$ of the target for generating the input strings. For all the other ways, the difference becomes very large.

As we said at the beginning of this article, we can easily explain this behaviour. By learning the primitive joint probability function $c(a,b)$, Ristad and Yianilos learn at the same time the marginal distribution $c(a)$. The learned edit costs (and the stochastic edit distance) are then dependent of the *a priori* distribution of the input strings, that is obviously awkward. To free of this statistical bias, we have to learn the primitive conditional probability function independently of the marginal distribution. That is the goal of the next section.

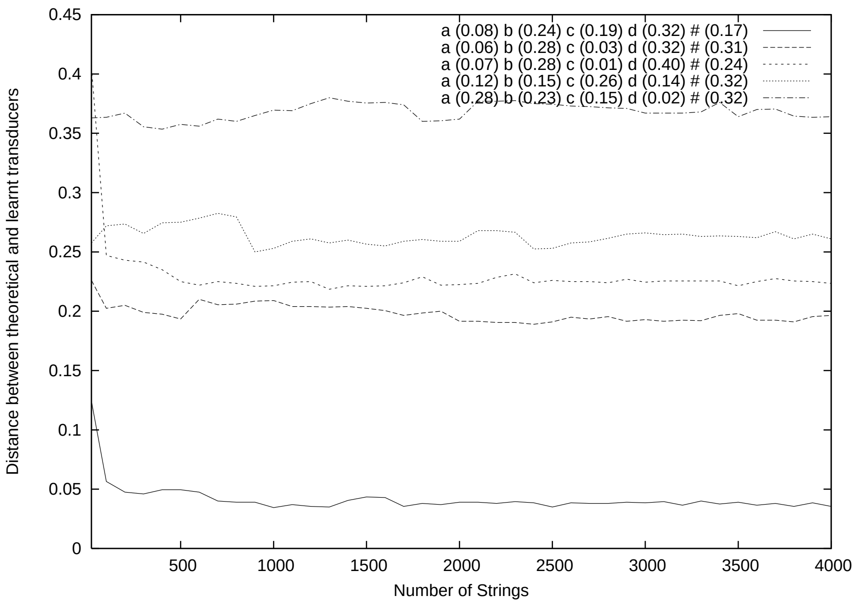


Fig. 2. Average difference between the target and the learned distributions according to various generations of the input strings.

## 4 Unbiased Learning of a Conditional Memoryless Transducer

A conditional memoryless transducer is denoted by a tuple $(X, Y, c, \gamma)$ where $X$ is the input alphabet, $Y$ is the output alphabet, $c$ is the primitive conditional probability function $c : E \to [0, 1]$ and $\gamma$ is the probability of the termination symbol of a string. As in the joint case, since $(\epsilon, \epsilon) \notin E$, in order to simplify the notation we use $\gamma$ and $c(\epsilon|\epsilon)$ as synonyms.

The probability $p : X^* \times Y^* \to [0, 1]$ of the string $y$ assuming the input one was a $x$ (noted $p(y|x)$) can be recursively computed by means of an auxiliary function (forward) $\alpha : X^* \times Y^* \to \mathbb{R}^+$ as:

$$\begin{aligned}\alpha(y|x) = {} & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(b|a) \cdot \alpha(y'|x')]_{x=x'a \wedge y=y'b} \\ & + [c(\epsilon|a) \cdot \alpha(y|x')]_{x=x'a} \\ & + [c(b|\epsilon) \cdot \alpha(y'|x)]_{y=y'b}.\end{aligned}$$

And then,

$$p(y|x) = \alpha(y|x)\gamma.$$

In a symmetric way, $p(y|x)$ can be recursively computed by means of an auxiliary function (backward) $\beta : X^* \times Y^* \to \mathbb{R}^+$ as:

$$\begin{aligned}\beta(y|x) = \quad & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(b|a) \cdot \beta(y'|x')]_{x=ax' \wedge y=by'} \\ & + [c(\epsilon|a) \cdot \beta(y|x')]_{x=ax'} \\ & + [c(b|\epsilon) \cdot \beta(y'|x)]_{y=by'}.\end{aligned}$$

And then,

$$p(y|x) = \beta(y|x)\gamma.$$

As in the joint case, both functions can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique. In this model a probability distribution is assigned conditionally to each input string. Then

$$\sum_{y \in Y^*} p(y|x) \in \{1, 0\} \quad \forall x \in X^*.$$

The $0$ is in the case the input string $x$ is not in the domain of the function [5] . It can be show (see Annex) that the normalisation of each conditional distribution can be achieved if the following conditions over the function $c$ and the parameter $\gamma$ are fulfilled,

$$\gamma > 0, c(b|a), c(b|\epsilon), c(\epsilon|a) \geq 0 \quad \forall a \in X, b \in Y \tag{1}$$

$$\sum_{b \in Y} c(b|\epsilon) + \sum_{b \in Y} c(b|a) + c(\epsilon|a) = 1 \quad \forall a \in X \tag{2}$$

$$\sum_{b \in Y} c(b|\epsilon) + \gamma = 1 \tag{3}$$

As in the joint case, the expectation-maximisation algorithm can be used in order the find the optimal parameters. The expectation step deals with the computation of the matrix $\delta$:

$$\begin{aligned}\delta(b|a) &= \sum_{(xax', yby') \in S} \frac{\alpha(y|x)c(b|a)\beta(y'|x')\gamma}{p(yby'|xax')} \\ \delta(b|\epsilon) &= \sum_{(xx', yby') \in S} \frac{\alpha(y|x)c(b|\epsilon)\beta(y'|x')\gamma}{p(yby'|xx')} \\ \delta(\epsilon|a) &= \sum_{(xax', yy') \in S} \frac{\alpha(y|x)c(\epsilon|a)\beta(y'|x')\gamma}{p(yy'|xax')} \\ \delta(\epsilon|\epsilon) &= \sum_{(x,y) \in S} \frac{\alpha(y|x)\gamma}{p(y|x)} = |S|.\end{aligned}$$

[5] If $p(x) = 0$ then $p(x, y) = 0$ and as $p(y|x) = \frac{p(x,y)}{p(x)}$ we have a $\frac{0}{0}$ indeterminism. We chose to solve it taking $\frac{0}{0} = 0$, in order to maintain $\sum_{y \in Y^*} p(y|x)$ finite.

The maximisation step allows us to deduce the current edit costs.

$$c(b|\epsilon) = \frac{\delta(b|\epsilon)}{N} \qquad\qquad \gamma = \frac{N - N(\epsilon)}{N}$$
$$c(b|a) = \frac{\delta(b|a)}{N(a)}\frac{N - N(\epsilon)}{N} \qquad\qquad c(\epsilon|a) = \frac{\delta(\epsilon|a)}{N(a)}\frac{N - N(\epsilon)}{N}$$

where:

$$N = \sum_{\substack{a \in X \cup \{\epsilon\} \\ b \in Y \cup \{\epsilon\}}} \delta(b|a) \qquad N(\epsilon) = \sum_{b \in Y} \delta(b|\epsilon) \qquad N(a) = \sum_{b \in Y \cup \{\epsilon\}} \delta(b|a)$$

For further details about these two stages see Annex 2.

We carried out experiments to assess the relevance of our new learning algorithm to correctly estimate the parameters of target transducers. We followed exactly the same experimental setup as the one of Section 3.3, except to the definition of our difference $d(c, c^*)$. Actually, as we said before, our new framework estimates $|X|$ conditional distributions. So $d(c, c^*)$ is defined as follows :

$$d(c, c^*) = \frac{(A + B\,|X|)}{2\,|X|}$$

where
$$A = \sum_{a \in X} \sum_{b \in Y \cup \{\epsilon\}} |c(b|a) - c^*(b|a)|$$
and
$$B = \sum_{b \in Y \cup \{\epsilon\}} |c(b|\epsilon) - c^*(b|\epsilon)|$$

The results are shown in Figure 3. We can make the two following remarks. First, the different curves clearly show that the convergence toward the target distribution is independent of the distribution of the input strings. Using different parameter configurations of the automaton of Figure 1, the behaviour of our algorithm remains the same, *i.e* the difference between the learned and the target conditional distributions tends to 0. Second, we can note that $d(c, c^*)$ rapidly decreases, *i.e.* the algorithm requires few learning examples to learn the target.

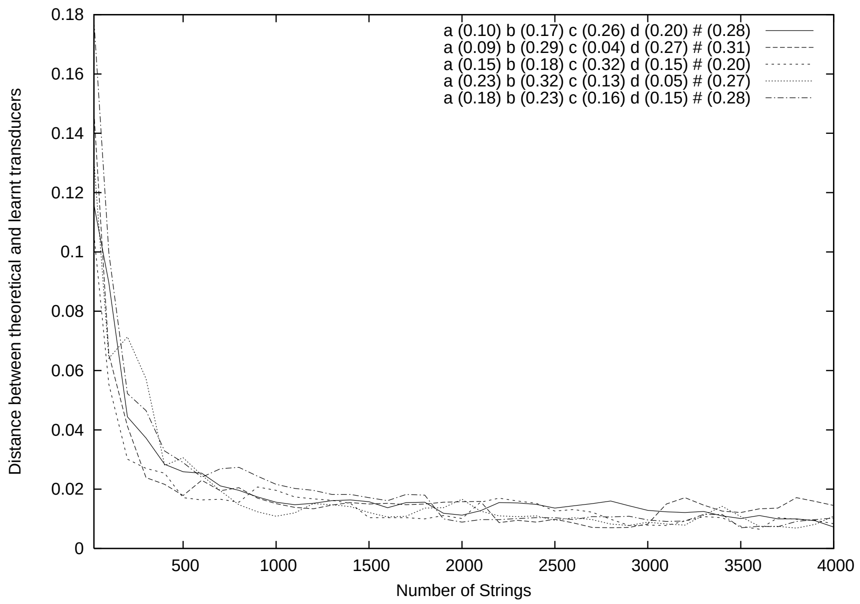


Fig. 3. Average difference between the target and the learned conditional distributions according to various generations of the input strings.

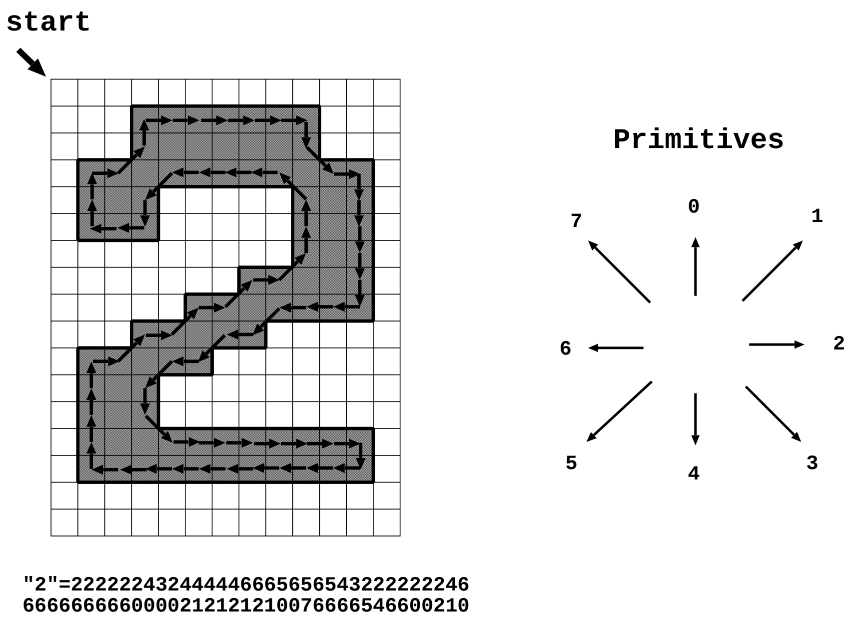


Fig. 4. Example of string coding character.

## 5 Application to the handwritten character recognition

### 5.1 Description of the database - Constitution of the set of pairs

In order to assess the relevance of our model in a pattern recognition task, we applied it on the real world problem of handwritten digit classification. We used the NIST Special Database 3 of the National Institute of Standards and Technology, already used in several articles such as [7–9]. This database consists in $128 \times 128$ bitmap images of handwritten digits and letters. In this series of experiments, we only focus on digits written by 100 different writers. Each class of digit (from 0 to 9) has about 1,000 instances, then the whole database we used contains about 10,000 handwritten digits. As we will explain later, a part of them will be used as a learning set $LS$, the remaining digits being kept in a test sample $TS$. Since our model handles strings, we coded each digit as an octal string, following the feature extraction algorithm proposed in [7]. It consists in scanning the bitmap left-to-right and starting from the top. When the first pixel is found, it follows the border of the character until it returns to the first pixel. During this traversal, the algorithm builds a string with the absolute direction of the next pixel in the border. Fig. 4 describes an example on a given "2". The vector of features in the form of a octal string is presented at the bottom of the figure.

As presenting throughout this article, our method requires a set of (input,output) pairs of strings for learning the probabilistic transducer. As we claimed before, the deduced stochastic edit distance can then be efficiently used for classification, sequence alignment, or noise correction. While it is rather clear in this last case that pairs in the form of (noisy,unnoisy) strings constitute the most relevant way to learn an edit distance useful in a noise correction model, what must they represent in a pattern recognition task, with various classes, such as in handwritten digit classification? As already proposed in [1], a possible solution consists in building pairs of "similar" strings that describe the possible variations or distortions between instances of each class. Such pairs can be drawn by an expert of the area. In this series of experiments on handwritten digits, we decided rather to automatically build pairs of (input,output) strings, where an input is a learning string of $LS$, and the output is a prototype of the input. We used as prototype the corresponding 1-nearest-neighbour in $LS$ of each input. On the one hand, this choice is motivated from an algorithmic standpoint. Actually, with a learning set constituted of $|LS|$ examples, such a strategy does not increase the complexity of the algorithm using $|LS|$ pairs of strings too. On the other hand, by attributing the nearest digit to each character, we ensure to model the main possible distortions between digits in each class.

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 can not be considered as a distortion of a "closed" zero, but rather as a different manner (a sort of sub-class) to design this digit. That explains that a nearest-neighbor based strategy is much more relevant.

To achieve this task, we used here a classic edit distance for computing the nearest-neighbour, *i.e.* with the same edit cost for an insertion, deletion or a substitution. The objective is then to learn a stochastic transducer that allows to optimise the conditional probabilities $p(output/input)$.

### *5.2 Experimental setup*

We claim that learning the primitive edit costs of an edit distance in the form of a conditional transducer is more relevant not only than learning a joint transducer, but also than fixing these costs in advance by an expert. Therefore, in the following series of experiments, we aim at comparing our approach (i) to the one of Ristad and Yianilos, and (ii) to the classic edit distance. The experimental setup, graphically described in Fig. 5, is the following:

(1) Learning Stage
- Step 1: each set $i$ of digits ($i = 0, .., 9$) is divided in two parts: a learning set $LS_i$ and a test set $TS_i$.
- Step 2: from each $LS_i$, we build a set of string pairs $PS_i$ in the form $(x, NN(x))$, $\forall x \in LS_i$, where $NN(x) = argmin_{y \in LS_i - \{x\}} d_E(x, y)$ ($d_E$ is the classic edit distance).
- Step 3: we learn a unique *conditional* transducer from $\cup_i PS_i$.
- Step 4: we learn a unique *joint* transducer from $\cup_i PS_i$.

(2) Test Stage
- Step 5: we classify each test digit $x' \in \cup_i TS_i$,
  - · by the class $i$ of the learning string $y \in \cup_i LS_i$ maximising $p(y|x')$ (using the conditional transducer)
  - · by the class $i$ of the learning string $y \in \cup_i LS_i$ maximising $p(x', y)$ (using the joint transducer)
  - · by the class $i$ of its nearest-neighbour $NN(x') \in \cup_i LS_i$

Using the previous experimental setup, we can then compare the three approaches under exactly the same conditions. Actually, during the test stage, each algorithm uses:

- one matrix concerning the primitive edit operations (an *a priori* fixed matrix of *edit costs* for the nearest-neighbour algorithm, a learned matrix of *edit probabilities* for the two others),
- the union of the learning sets $LS_i$.
- the classic edit distance algorithm (for the nearest-neighbour algorithm), or its probabilistic version (for the others).

Note that for the standard edit distance, we used two different matrices of edit costs. The first one is the most classic one, *i.e.* each edit operation has the same cost (here, 1). According to [8], a more relevant strategy would consist in taking costs proportionally to the relative angle between the directions used for describing a

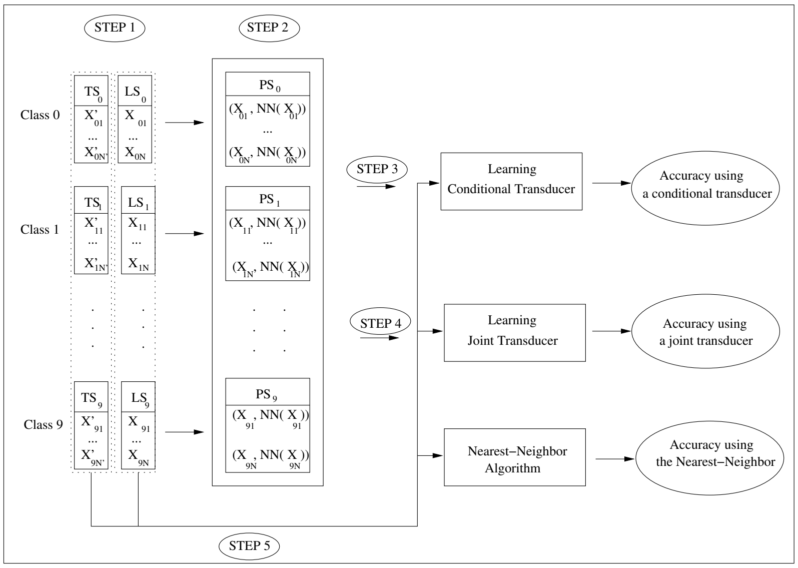


Fig. 5. Experimental setup in 5 steps. $x'_{ij}$ (resp. $x_{ij}$) is the $j^{th}$ test (resp. learning) string of the class $i$.

| $W_s$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 |
| 1 | 1 | 0 | 1 | 2 | 3 | 4 | 3 | 2 |
| 2 | 2 | 1 | 0 | 1 | 2 | 3 | 4 | 3 |
| 3 | 3 | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| 4 | 4 | 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| 5 | 3 | 4 | 3 | 2 | 1 | 0 | 1 | 2 |
| 6 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | 1 |
| 7 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 |

Table 2
Substitution costs $W_s$. The insertion and deletion costs are fixed to 1.

digit. To assess the efficiency of these other costs, we also used the matrix described in Table 2.

In order to assess each algorithm in different configurations, the number of learning strings varied from 200 (20 for each class of digits) to 6,000 (600 for each class), with a step of 20 strings per class (resulting in 30 step iterations). The test accuracy was computed with a test set containing always 2,000 strings (*i.e.* $|\cup_i TS_i| = 2,000$). For each learning size, we run 5 times each algorithm using 5 different randomly generated learning sets and we computed the average. Therefore, the results presented in Fig. 6 were computed from 30 (# of steps) $\times$ 4 (# of methods) $\times$ 5 (# of iterations) = 600 learning processes. During the test stage, 2,000 $\times$ 600 = 1,200,000 test strings were labelled.

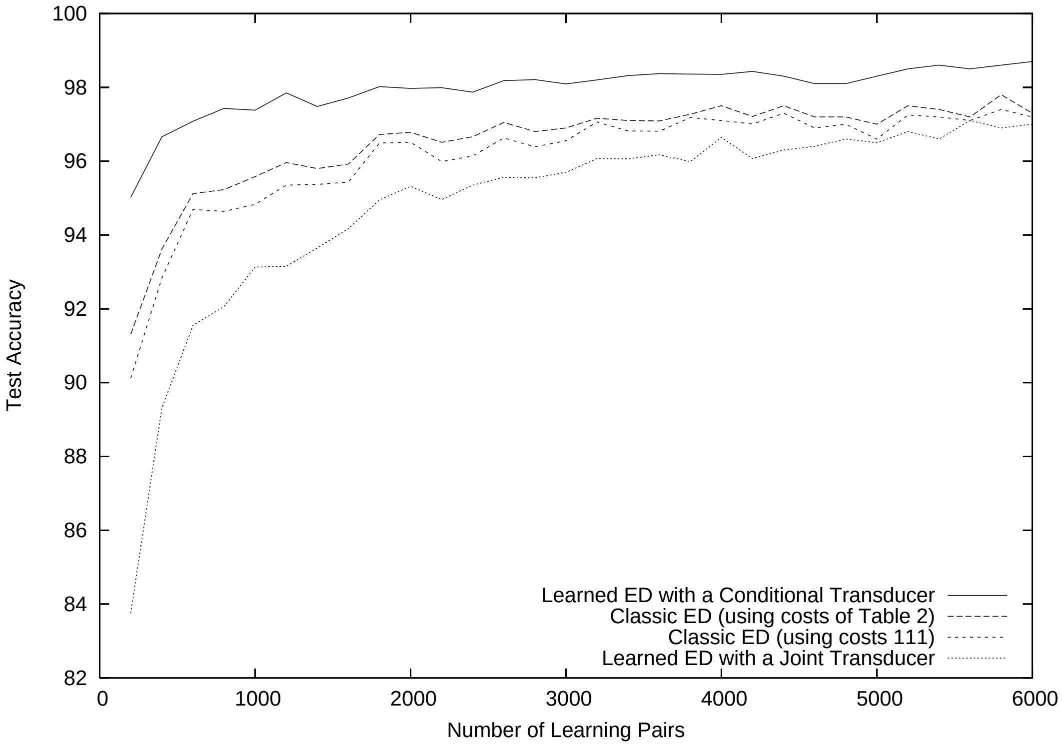


Fig. 6. Test accuracy on the handwritten digits.

### *5.3 Results and Discussion*

From Fig. 6, we can make the following remarks.

First of all, learning an edit distance in the form of a conditional transducer is indisputably relevant to achieve a pattern recognition task. Whatever the size of the learning set, the test accuracy obtained using the stochastic edit distance is higher than the others. However, note that the difference decreases logically with the size of the learning set. Actually, from a theoretical standpoint, $lim_{|LS| \rightarrow \infty} P(d(x, NN(x)) > \epsilon) = 0, \forall \epsilon > 0$. In other words, it means that whatever the distance we choose, when the number of examples increases, the nearest-neighbour of an example $x$ tends to be $x$ itself. Interestingly, we can also note that for reaching approximately the same accuracy rate, the standard edit distance (using costs of Table 2) needs much more learning strings, and therefore requires a higher time complexity, than our approach.

Second, the results obtained with Ristad and Yianilos's method are logical and easily interpretable. When the number of learning string pair is small, all the drawbacks we already mentioned in the first part of this paper occur. Actually, while a nearest-neighbour is always a string belonging to the learning set, many learning strings are not present in the current (small) set of nearest-neighbours. There-

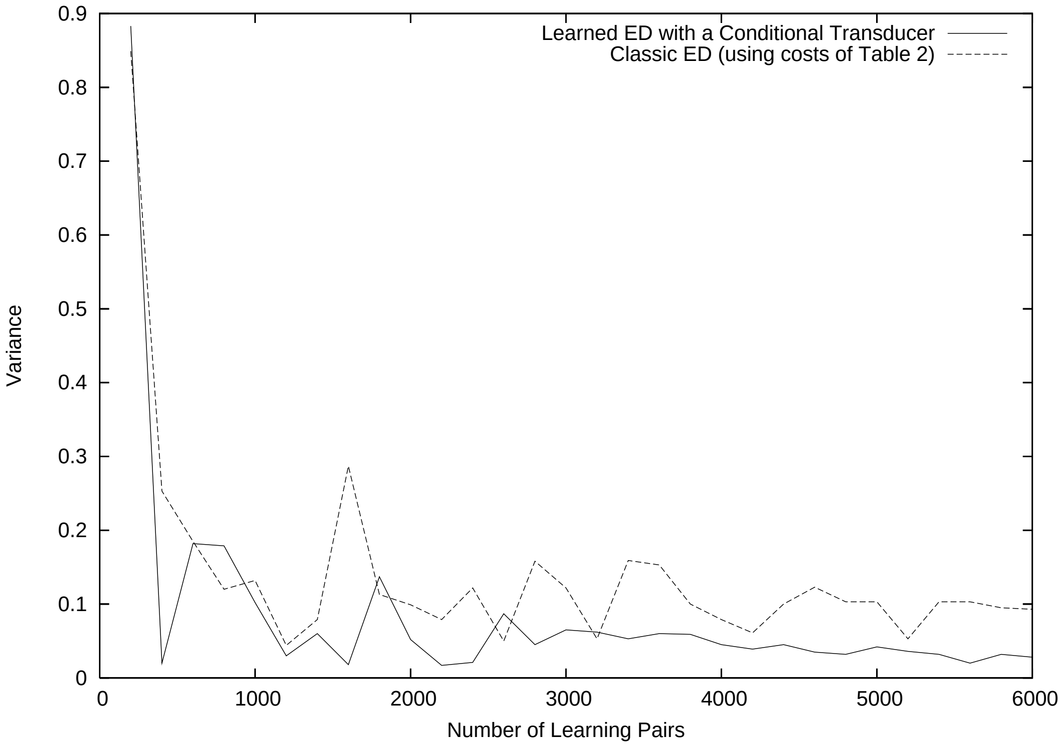


Fig. 7. Evolution of the variance throughout the iterations.

fore, while all these strings (inputs and outputs) come from the same set of digits ($\cup_i LS_i$), the distribution over the outputs (the nearest-neighbours) is not the same as the distribution over the inputs (the learning strings). Of course, this bias decreases with the rise of the learning set size, but not sufficiently in this series of experiments for improving the performances of the classic edit distance.

Moreover, as already noted in [8], the use of the matrix of costs of Table 2 provides better results than the naïve configuration consisting in using the same cost for the three edit operations. Even if the difference is not important between the two curves, the first one is always higher than the second. However, it is not sufficient to beat the learned edit distance with a conditional transducer. To assess the level of stability of the approaches, we have computed a measure of dispersion on the results provided by the standard edit distance (with costs of Table 2) and our learned distance. Fig. 7 shows the behaviour of the variance of the test accuracy throughout the iterations. Interestingly, we can note that in the large majority of the cases, our method gives a smaller variance.

## 6 Conclusion

In this paper, we proposed a relevant approach for learning the stochastic edit distance in the form of a memoryless transducer. While the standard techniques aim at learning a joint distribution over the edit operations, we showed that such a strategy induces a bias in the form of a statistical dependence on the input string distribution. We overcame this drawback by directly learning a conditional distribution of the primitive edit costs. The experimental results on a handwritten digit recognition task bring to the fore the interest of our approach.

We think that this work deserves further investigations. First, we believe that the way to build the pairs of strings can be efficiently improved. So far, we used as prototype, the nearest-neighbour of each learning string. The $k$-nearest-neighbours, or clustering-based strategies should be studied in our future works. We have also to study an adaptive strategy which would update the learning set of pairs by using at each iteration of the EM algorithm the edit costs learned during the previous stage. Second, beyond its good behaviour for dealing with a classification task, our model can be also particularly suited for handling noisy data. Actually, it can be used to correct noisy learning instances before any inference process. Moreover, we also plan to extend our work on semi-structured data, such as trees. One of our objective consists in improving classification performances for applications in music retrieval, which handles tree-based representations for identifying new melodies.

## Annex 1

We are going to show that eq. 1, 2 and 3 are sufficient to satisfy

$$\sum_{y \in Y^*} p(y|x) = 1.$$

Let us first consider the case when $x = \epsilon$.

$$\begin{aligned}
\sum_{y \in Y^*} \alpha(y|\epsilon) &= 1 + \sum_{yb \in Y^*} \alpha(yb|\epsilon) \\
&= 1 + \sum_{yb \in Y^*} c(b|\epsilon)\alpha(y|\epsilon) \\
&= 1 + \sum_{b \in Y} c(b|\epsilon) \sum_{y \in Y^*} \alpha(y|\epsilon)
\end{aligned}$$

then

$$\sum_{y \in Y^*} \alpha(y|\epsilon)(1 - \sum_{b \in Y} c(b|\epsilon)) = 1$$
$$\sum_{y \in Y^*} \alpha(y|\epsilon) = \left(1 - \sum_{b \in Y} c(b|\epsilon)\right)^{-1}$$

Let us now consider the complete case

$$\begin{aligned}
&\sum_{y \in Y^*} \alpha(y|xa) = \\
&\quad \alpha(\epsilon|xa) + \sum_{yb \in Y^*} \alpha(yb|xa) = \\
&\quad c(\epsilon|a)\alpha(\epsilon|x) \\
&\qquad + \sum_{yb \in Y^*} (c(b|a)\alpha(y|x) + c(b|\epsilon)\alpha(y|xa) + c(\epsilon|a)\alpha(yb|x)) = \\
&\quad c(\epsilon|a)\alpha(\epsilon|x) + \sum_{b \in Y} c(b|a) \sum_{y \in Y^*} \alpha(y|x) \\
&\qquad + \sum_{b \in Y} c(b|\epsilon) \sum_{y \in Y^*} \alpha(y|xa) + c(\epsilon|a) \sum_{yb \in Y^*} \alpha(yb|x) = \\
&\quad c(\epsilon|a) \sum_{y \in Y^*} \alpha(y|x) + \sum_{b \in Y} c(b|a) \sum_{y \in Y^*} \alpha(y|x) \\
&\qquad + \sum_{b \in Y} c(b|\epsilon) \sum_{y \in Y^*} \alpha(y|xa) = \\
&\quad \left(c(\epsilon|a) + \sum_{b \in Y} c(b|a)\right) \sum_{y \in Y^*} \alpha(y|x) + \sum_{b \in Y} c(b|\epsilon) \sum_{y \in Y^*} \alpha(y|xa)
\end{aligned}$$

then

$$\sum_{y \in Y^*} \alpha(y|xa) \left(1 - \sum_{b \in Y} c(b|\epsilon)\right) = \left(c(\epsilon|a) + \sum_{b \in Y} c(b|a)\right) \sum_{y \in Y^*} \alpha(y|x)$$

and

$$\sum_{y \in Y^*} \alpha(y|xa) = \left(1 - \sum_{b \in Y} c(b|\epsilon)\right)^{-1} \left(c(\epsilon|a) + \sum_{b \in Y} c(b|a)\right) \sum_{y \in Y^*} \alpha(y|x)$$

Applying this equation recursively on the length of $x$ and taking in account that the base case is

$$\sum_{y \in Y^*} \alpha(y|\epsilon) = \left(1 - \sum_{b \in Y} c(b|\epsilon)\right)^{-1}$$

we have

$$\sum_{y\in Y^*} \alpha(y|a_1\dots a_n) =$$
$$\prod_{i=1}^{n}\left[\left(1-\sum_{b\in Y} c(b|\epsilon)\right)^{-1}\left(c(\epsilon|a_i)+\sum_{b\in Y} c(b|a_i)\right)\right]\cdot\left(1-\sum_{b\in Y} c(b|\epsilon)\right)^{-1}$$

and

$$\sum_{y\in Y^*} p(y|a_1\dots a_n) =$$
$$\prod_{i=1}^{n}\left[\left(1-\sum_{b\in Y} c(b|\epsilon)\right)^{-1}\left(c(\epsilon|a_i)+\sum_{b\in Y} c(b|a_i)\right)\right]\cdot\left(1-\sum_{b\in Y} c(b|\epsilon)\right)^{-1}\gamma$$

A sufficient condition for $\sum_{y\in Y^*} p(y|a_1\dots a_n) = 1$ is that each of the terms that appear in the productory is equal to 1 and that the final product is also 1. Then,

$$\left(1-\sum_{b\in Y} c(b|\epsilon)\right)^{-1}\left(c(\epsilon|a_i)+\sum_{b\in Y} c(b|a_i)\right) = 1$$
$$1-\sum_{b\in Y} c(b|\epsilon) = c(\epsilon|a_i)+\sum_{b\in Y} c(b|a_i)$$
$$\sum_{b\in Y} c(b|\epsilon) + c(\epsilon|a_i)+\sum_{b\in Y} c(b|a_i) = 1$$

and we have equation 2, and

$$\left(1-\sum_{b\in Y} c(b|\epsilon)\right)^{-1}\gamma = 1$$
$$1-\sum_{b\in Y} c(b|\epsilon) = \gamma$$
$$\gamma+\sum_{b\in Y} c(b|\epsilon) = 1$$

and we have equation 3.

Note that these equations are not valid if $\sum_{b\in Y} c(b|\epsilon) = 1$ but this is impossible since $\gamma > 0$.

## Annex 2

Let us assume that a problem can be represented in terms of two measure spaces: $\mathcal{O}$, a space of observable data, and $\mathcal{U}$, one of unobservable data. Suppose that there is a parameter vector $\theta$ on which the distributions $\mathcal{O}$ and $\mathcal{U}$ depend. The aim is to find that $\theta$ that maximises the likelihood function $l(\mathrm{O},\theta) = \ln(p(\mathrm{O}|\theta))$, for a given set O of $\mathcal{O}$ of observed data.

In general, finding $\theta$ is not possible analytically, and so a given approximating algorithm should be used instead. The Expectation Maximisation algorithm produces iteratively estimates of $\theta$, each one producing a greater value of $l$. The procedure can then be run until the convergence of $\theta$. Dempster *et al.* [10] showed that, given an estimate $\theta_n$ of $\theta$, a better estimate $\theta_{n+1}$ can be produced by maximising:

$$Q(\theta_n,\theta_{n+1}) = \mathbf{E}[\ln(p(\mathrm{O},\mathcal{U}|\theta_{n+1}))|\mathrm{O},\theta_n]$$

where $\mathbf{E}$ is a conditional expectation over the distribution $\mathcal{U}$. The two parts of the algorithm are therefore the *Expectation* step, in which this expectation is found, and the *Maximisation* step, in which a new parameter $\theta_{n+1}$ that maximises it is deduced.

Let $S \subset X^* \times Y^*$ be a multiset of pairs of strings [6] (the learning (multi)set), let $S_i = \{x : (x,y) \in S\}$ and let $S_o = \{y : (x,y) \in S\}$ the input and output multisets. In the case of conditional transducer learning we are interested in finding the parameters ($\theta$) of the transducer that maximises the probability of the observed multiset of $S_o$ output strings provided the $S_i$ multiset of input strings. Then the likelihood function to maximise is:

$$l(S_o,\theta,S_i) = \ln(p(S_o|\theta,S_i)) = \ln \prod_{(x,y)\in S} p(y|\theta,x)$$

with respect to the parameter vector $\theta$.

In the following, a path allowing us to transform an input into an output will be represented by a string $z$ belonging to the set $E^*$. In other words, the string $z$ is the sequence of the edit operations that have been iteratively used during the transformation. The set of all the paths $E^*$ characterises then our unobservable data.

Given $z = (x_1,y_1)\dots(x_n,y_n) \in E^*$, we say that $x$ is the input string of $z$ (noted $x = i(z)$) iff $x = x_1 \dots x_n$. Note that $x$ is the concatenation of $n$ strings of length smaller or eq ual to one, among them some can be the empty string $\epsilon$. Therefore, the length of $x$ is smaller or equal to $n$. Symmetrically, we say that $y$ is the output string of $z$ (noted $y = o(z)$) iff $y = y_1 \dots y_n$.

On the following, given a $e = (x,y) \in \mathbf{E}$ and any function $f : E \to \mathbb{R}$ we are going to denote indistinctly $f(e)$, $f((x,y))$ or $f(y|x)$. Remember that we are using the notation $c(\epsilon|\epsilon)$ as a synonym of $\gamma$, then we are going to use also $c((\epsilon,\epsilon))$ as a synonym of $c(\epsilon|\epsilon)$.

[6] Although in the following we are going to use the set notation for multisets, we have to take into account that multisets admit repetitions of their components.

Let $(X, Y, c, \gamma)$ be a memoryless transducer and let $z = (x_1, y_1) \dots (x_n, y_n) \in E^*$, then the conditional probability of the path $z$ is:

$$p(z|i(z)) = \prod_{i=1}^{n} c(y_i|x_i)c(\epsilon|\epsilon) = \prod_{i=1}^{n} c(z_i)c((\epsilon, \epsilon))$$

For each input-output $(x, y)$ pair, we define the path set as:

$$E(x, y) = \{z \in E^* : x = i(z), y = o(z)\}$$

It is easy to see that

$$p(y|x) = \sum_{z \in E(x,y)} p(z|x)$$

Given a multiset $S \subset X^* \times Y^*$, we define the multiset

$$E(S) = \cup_{(x,y) \in S} E(x, y)$$

In our case, the $Q$ function can be written as:

$$\begin{aligned} Q(\theta_n, \theta_{n+1}) &= \mathbf{E}[\ln(p(S_o, z|\theta_{n+1}, S_i))|S_o, \theta_n, S_i] \\ &= \sum_{z \in E^*} p(z|S_o, \theta_n, S_i) \ln p(S_o, z|\theta_{n+1}, S_i) \end{aligned}$$

as $p(z|y, \theta_n, x) = 0$ if $x \neq i(z)$ or $y \neq o(z)$

$$\begin{aligned} &= \sum_{z \in E(S)} p(z|o(z), \theta_n, i(z)) \ln p(o(z), z|\theta_{n+1}, i(z)) \\ &= \sum_{z \in E(S)} p(z|o(z), \theta_n, i(z)) \ln p(z|\theta_{n+1}, i(z)) \\ &= \sum_{z \in E(S)} p(z|o(z), \theta_n, i(z)) \left( \sum_{i=0}^{|z|} \ln c(o(z_i)|\theta_{n+1}, i(z_i)) + \ln c(\epsilon|\theta_{n+1}, \epsilon) \right) \\ &= \sum_{e \in E} \sum_{zez' \in E(S)} p(zez'|o(zez'), \theta_n, i(zez')) \ln c(e|\theta_{n+1}) \\ &\quad + \sum_{z \in E(S)} p(z|o(z), \theta_n, i(z)) c((\epsilon, \epsilon)|\theta_{n+1}) \\ &= \sum_{e \in E} \delta(c) \ln c(e|\theta_{n+1}) + |S| \ln c((\epsilon, \epsilon)|\theta_{n+1}) \end{aligned}$$

where

$$
\begin{aligned}
\delta(e) &= \sum_{zez' \in E(S)} p(zez'|o(zez'), \theta_n, i(zez')) \\
&= \sum_{zez' \in E(S)} \frac{p(o(zez'), zez'|\theta_n, i(zez'))}{p(o(zez')|\theta_n, i(zez'))} \\
&= \sum_{zez' \in E(S)} \frac{p(zez'|\theta_n, i(zez'))}{p(o(zez')|\theta_n, i(zez'))}
\end{aligned}
$$

Giving

$$
\begin{aligned}
\delta(b|a) &= \sum_{(xax', yby') \in S} \frac{\alpha(y|x)c(b|a)\beta(y'|x')\gamma}{p(yby'|xax')} \\
\delta(b|\epsilon) &= \sum_{(xx', yby') \in S} \frac{\alpha(y|x)c(b|\epsilon)\beta(y'|x')\gamma}{p(yby'|xx')} \\
\delta(\epsilon|a) &= \sum_{(xax', yy') \in S} \frac{\alpha(y|x)c(\epsilon|a)\beta(y'|x')\gamma}{p(yy'|xax')}
\end{aligned}
$$

as required.

Now we have to choose $\theta_{n+1}$ that minimises the $Q(\theta_n, \theta_{n+1})$ function with the restrictions:

$$
\sum_{b \in Y} c((a,b)|\theta_{n+1}) + \sum_{b \in Y} c((\epsilon,b)|\theta_{n+1}) + c((a,\epsilon)|\theta_{n+1}) = 1, \quad \forall a \in X
$$

$$
\sum_{b \in Y} c((\epsilon,b)|\ \theta_{n+1}) + c((\epsilon,\epsilon)|\theta_{n+1}) = 1
$$

Using the Lagrange multipliers

$$
\begin{aligned}
L = \quad & \sum_{e \in E} \delta(e) \ln c(e|\theta_{n+1}) + |S| \ln c((\epsilon,\epsilon)|\theta_{n+1}) \\
& - \sum_{a \in X} \mu_a \left( \sum_{b \in Y} c((a,b)|\theta_{n+1}) + \sum_{b \in Y} c((\epsilon,b)|\theta_{n+1}) + c((a,\epsilon)|\theta_{n+1}) - 1 \right) \\
& - \mu \left( \sum_{b \in Y} c((\epsilon,b)|\theta_{n+1}) + c((\epsilon,\epsilon)|\theta_{n+1}) - 1 \right)
\end{aligned}
$$

Computing the derivatives and equating to zero we have:

$$
\begin{aligned}
c((a,b)|\theta_{n+1}) &= \frac{\delta((a,b))}{\mu_a} & \qquad c((\epsilon,b)|\theta_{n+1}) &= \frac{\delta((\epsilon,b))}{\sum_a \mu_a + \mu} \\
c((a,\epsilon)|\theta_{n+1}) &= \frac{\delta((a,\epsilon))}{\mu_a} & \qquad c((\epsilon,\epsilon)|\theta_{n+1}) &= \frac{|S|}{\mu}
\end{aligned}
$$

Substituting in the normalisation equation we obtain:

$$
\begin{gathered}
\frac{\sum_b \delta((\epsilon,b))}{\sum_a \mu_a + \mu} + \frac{\sum_b \delta((a,b))}{\mu_a} + \frac{\delta((a,\epsilon))}{\mu_a} = 1, \quad \forall a \in X \\
\frac{\sum_b \delta((\epsilon,b))}{\sum_a \mu_a + \mu} + \frac{|S|}{\mu} = 1
\end{gathered}
$$

Now we have a system with $|X|+1$ equations and $|X|+1$ unknowns. It is easy to see that

$$
\mu = |S|\frac{N}{N - N(\epsilon)} \qquad\qquad \mu_a = N(a)\frac{N}{N - N(\epsilon)}
$$

with

$$
N = \sum_{e \in E} \delta(e) + |S| \qquad N(\epsilon) = \sum_{b \in Y} \delta((\epsilon,b)) \qquad N(a) = \sum_{b \in Y \cup \{\epsilon\}} \delta((a,b))
$$

is a solution to the system.

## References


[1] E. S. Ristad, P. N. Yianilos, Learning string-edit distance, IEEE Trans. Pattern Anal. Mach. Intell. 20 (5) (1998) 522–532.

[2] Y. Sakakibara, R. Siromoney, A noise model on learning sets of strings, in: COLT ’92: Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 295–302.

[3] E. S. Ristad, P. N. Yianilos, Finite growth models, Tech. Rep. CS-TR-533-96, Princeton University Computer Science Department (1996).

[4] F. Casacuberta, Probabilistic estimation of stochastic regular syntax-directed translation schemes, in: Proceedings of th VIth Spanish Symposium on Pattern Recognition and Image Analysis, 1995, pp. 201–207.

[5] A. Clark, Memory-based learning of morphology with stochastic transducers, in: Proceedings of the Annual meeting of the association for computational linguistic, 2002.

[6] J. Eisner, Parameter estimation for probabilistic finite-state transducers, in: Proceedings of the Annual meeting of the association for computational linguistic, 2002, pp. 1–8.

[7] E. Gómez, L. Micó, J. Oncina, Testing the linear approximating eliminating search algorithm in handwritten character recognition tasks, in: VI Symposium Nacional de reconocimiento de Formas y Análisis de Imágenes, 1995, pp. 212–217.

[8] L. Micó, J. Oncina, Comparison of fast nearest neighbour classifiers for handwritten character recognition, Pattern Recognition Letters 19 (1998) 351–356.

[9] J. R. Rico-Juan, L. Micó, Comparison of aesa and laesa search algorithms using string and tree-edit-distances, Pattern Recognition Letters 24 (2003) 1417–1426.

[10] A. Dempster, M. Laird, D. Rubin, Maximun likelihood from incomplete data via the em algorithm, J. R. Stat. Soc. B (39) (1977) 1–38.