# Computing the Expected Edit Distance from a String to a PFA

Jorge Calvo-Zaragoza[1], Colin de la Higuera[2], and Jose Oncina[1]

[1] DLSI, University of Alicante, Spain
{jcalvo,oncina}@dlsi.ua.es
[2] LINA Lab, UMR 6241, University of Nantes, France
cdlh@univ-nantes.fr

**Abstract.** In a number of fields one is to compare a witness string with a distribution. One possibility is to compute the probability of the string for that distribution. Another, giving a more global view, is to compute the expected edit distance from a string randomly drawn to the witness string. This number is often used to measure the performance of a prediction, the goal then being to return the *median* string, or the string with smallest expected distance.
To be able to measure this, computing the distance between a hypothesis and that distribution is necessary. This paper proposes two solutions for computing this value, when the distribution is defined with a probabilistic finite state automaton. The first is exact but has a cost which can be exponential in the length of the input string, whereas the second is a Fpras.

**Keywords:** Edit Distance, Probabilistic Finite State Automata

## 1 Introduction

The edit or Levenshtein distance is often used to measure how close one string is to another [14]. This distance has given rise to many questions: if one is given a set instead of a string, the question may be to compute rapidly the distance between the set and a string or between two sets [28, 17]. In turn, a set defines an empirical distribution which can be represented by a probabilistic finite state automaton (Pfa), a hidden Markov model or a weighted automaton [17, 27].

If the set is used as a learning sample, the distribution may be more general, again represented by the above machines, but these may now contain cycles and therefore define a distribution over all possible strings.

The following questions are then posed: what is the expected distance between a given *witness* string and such a distribution? What could a representative string be for this distribution? One possible answer to the second question is the *most probable string* [10, 11]. Another is the *median string* which is the string minimizing the expected distance to the distribution, which in turn contributes to make the first question relevant. These questions have not only a precise mathematical interest, but they have been posed in very different settings like bio-informatics [7], pattern recognition [19] or computational linguistics [24].

Alternative distances have been studied, such as the minimum cost obtaining by summing the weight of a string and its distance to the witness string [2]. Balls of strings, Levenshtein automata and other finite state machines linking regular languages and the edit distance have been introduced, discussed and studied [17, 4, 16, 23].

We prove in this paper two results.

The first is that the expected edit distance can be computed, and that if the weights of the Pfa are rational, then the result itself is rational. The construction involves building a multiplicity automaton which can be of size exponential in the length of the string $w$, but only increases polynomially with the number of states of the Pfa or the size of the alphabet.

The second result is that the problem admits a *fully polynomial time randomized schema* (Fpras), that is, a randomized algorithm which will return a probably approximatively correct value in time polynomial with the length of the string, the size of the automaton representing the distribution, and the inverse of the accepted error.

Each algorithm has its advantages and inconveniences, as we will show in the experimental section: the method involving the multiplicity automaton will give an exact result, but only for small witness strings. The Fpras, on the other hand, can only build an approximate result, but there is a guarantee on the error bound and the method can handle long witness strings and large Pfa.

After introducing notations and definitions (Section 2), we prove in Section 3 that the problem is decidable and provide an algorithm which gives the correct result; Section 4 presents a polynomial randomized computation whose result is probably approximately correct. Our experiments, described in Section 5, empirically confirm the bounds in both error and complexity of the proposed strategies. Section 6 concludes the present work.

## 2 Preliminaries

### 2.1 Basic Notations

An *alphabet* $\Sigma$ is a finite non-empty set of symbols called *letters*. A *string* $w$ over $\Sigma$ is a finite sequence $w = w_1 \dots w_m$ of letters. Letters will be indicated by $a, b, c, \dots$, and strings by $u, v, \dots, z$. Let $|w|$ denote the length of $w$. In this case we have $|w| = |w_1 \dots w_m| = m$. The *empty string* is denoted by $\lambda$.

We denote by $\Sigma^*$ the set of all strings, by $\Sigma^m$ the set of those of length $m$.

A *probabilistic language* $\mathcal{D}$ is a probability distribution over $\Sigma^*$. The probability of a string $w \in \Sigma^*$ under the distribution $\mathcal{D}$ is denoted as $Pr_{\mathcal{D}}(w)$. The distribution must satisfy $\sum_{w \in \Sigma^*} Pr_{\mathcal{D}}(w) = 1$.

If the distribution is modelled by some syntactic machine $\mathcal{M}$, the probability of $x$ according to the probability distribution defined by $\mathcal{M}$ is denoted by $Pr_{x \sim \mathcal{M}}(x)$ or simply $Pr_{\mathcal{M}}(x)$.

## 2.2 Multiplicity Automata

An $n$-state Multiplicity Automata (MA) $\mathcal{M}$ (also known as *recognizable series* [5] or *Stochastic Sequential Machines* [20]) can be defined by a 4-tuple $\langle \Sigma, \mathbf{S}, \mathbf{M}, \mathbf{F} \rangle$here: $\Sigma$ is the alphabet, $\mathbf{S} \in \mathbb{Q}^{1\times n}$, $\mathbf{M} = \{\mathbf{M}_a \in \mathbb{Q}^{n\times n} : a \in \Sigma\}$, and $\mathbf{F} \in \mathbb{Q}^{n\times 1}$.

$\mathcal{M}$ realizes a function from $\Sigma^*$ to $\mathbb{Q}$ such that:

$$\mathcal{M}(x_1 \cdots x_k) = \mathbf{S} \sum_{i=1}^{k} \mathbf{M}_{x_i} \mathbf{F}$$

This machine can also be defined from a graph point of view as an $n$-state machine $\langle \Sigma, Q, \mathbb{S}, \mathbb{F}, \delta \rangle$ where $Q = \{q_0, \cdots, q_{n-1}\}$, $\mathbb{S} : Q \to \mathbb{Q}$ are the initial weights ($\mathbb{S}(q_i) = \mathbf{S}[i]$), $\mathbb{F} : Q \to \mathbb{Q}$ are the final weights ($\mathbb{F}(q_i) = \mathbf{F}[i]$), and $\delta : Q \times \Sigma \times Q \to \mathbb{Q}$ are the transition weights ($\delta(q_i, a, q_j) = [\mathbf{M}_a]_{i,j}$).

Given $x \in \Sigma^*$, $\Pi_{\mathcal{M}}(x)$ is the set of all paths accepting $x$: an *accepting* $x$-path is a sequence $\pi = q_{i_0} x_1 q_{i_1} x_2 \dots x_k q_{i_k}$ where $x = x_1 \cdots x_k$, $a_i \in \Sigma$, and $\forall j \in [1, k]$ such that $\delta(q_{i_{j-1}}, a_j, q_{i_j}) \neq 0$. Let $\pi = q_{i_0} x_1 q_{i_1} x_2 \dots x_k q_{i_k}$, we denote by $\delta(\pi) = \prod_{j=1}^{k} \delta(q_{i_{i-1}}, a_j, q_{i_j})$, $\alpha(\pi) = q_{i_0}$ and $\omega(\pi) = q_{i_k}$.

$$\mathcal{M}(x) = \sum_{\pi \in \Pi_{\mathcal{M}}(x)} \mathbb{S}(\alpha(\pi)) \delta(\pi) \mathbb{F}(\omega(\pi))$$

This can be computed efficiently using the Forward (or Backwards) algorithm. Obviously, the two ways to compute $\mathcal{M}(x)$ are equivalent.

Probabilistic Finite Automata (PFA) can be viewed as a special type of MA that are restricted to describe probability distributions over sets of strings. Then further restrictions should be applied. Let $\mathbf{1} \in \mathbb{Q}^{n\times 1} : \mathbf{1}[i] = 1 \forall i$, $I \in \mathbb{Q}^{n\times n}$ be the identity matrix and $\mathbf{M}_\Sigma = \sum_{a\in\Sigma} \mathbf{M}_a$, then:

- the components of $\mathbf{S}$, $\mathbf{M}$ and $\mathbf{F}$ are interpreted as probabilities, that is, they should be in $[0, 1]$
- $\mathbf{S1} = 1$: the sum of the starting probabilities should add one
- $\mathbf{M}_\Sigma \mathbf{1} + \mathbf{F} = \mathbf{1}$: for any state, the sum of the outgoing probability plus the ending probability should add one
- $(I - \mathbf{M}_\Sigma)$ should be non-singular: this is a sufficient condition to assure the non existence of absorbing states (or set of states).

## 2.3 The Edit Distance

The edit distance between two strings $d_e(x, y)$ is the minimum number of edition operations needed to transform $x$ into $y$ [14].

We will make use of the following generous bounds for the edit distance:

$$d_e(x, y) \le \max\{x, y\} \le |x| + |y| \tag{1}$$

The *relative edit distance* from $x$ to $y$ is $d_r(x, y) = d_e(x, y) - |y|$. Notice that this is not a metric.

It follows from (1) that for a fixed string $x$ the set of values that $d_r(x, y)$ can take is finite, with values ranging from $-|x|$ to $|x|$, even though the set of strings from which $y$ is chosen is infinite.

We extend the definitions to distributions over strings (*string-distribution edit distance*):

$$d_e(w, \mathcal{D}) = \sum_{y \in \Sigma^*} d_e(w, y) Pr_{\mathcal{D}}(y) = \sum_{y \in \Sigma^*} d_r(w, y) Pr_{\mathcal{D}}(y) + \sum_{y \in \Sigma^*} |y| Pr_{\mathcal{D}}(y) \quad (2)$$

When $\mathcal{D}$ is given by a PFA $\mathcal{A}$, we can also write $d_e(w, \mathcal{A})$.

## 2.4 Complexity Issues

Let us recall that a decision problem is one for which the possible answers are **true** and **false**. Such a problem is in class **P** if there is a deterministic Turing machine solving any instance in polynomial time, in **NP** if this machine is non-deterministic, **NP**-complete if it as hard as any of the other **NP**-complete problems.

An optimization problem asks for a numerical value to be computed given an instance. This value can sometimes be approximated by a polynomial-time approximation scheme (PTAS) which can compute a value within a factor $1+\epsilon$ of the optimum in time polynomial in the size of the approximation scheme. If the runtime also depends polynomially of $1/\epsilon$, the scheme is called a fully polynomial-time approximation scheme or FPTAS. For more about approximation algorithms, see [26].

Sometimes, deterministic algorithms are unable to approximate, but randomized algorithms [18] can solve the problem in the following sense: an algorithm **A** is a *fully polynomial time randomized schema* or FPRAS if it can return a solution which is at distance $\epsilon$ of the optimum, with confidence at least $1 - \delta$ and runs in time polynomial in the size of the instance, $1/\epsilon$ and $1/\delta$.

The key problem in this work is called EDD:

**Name:** Computing the edit distance to a distribution (EDD)
**Instance:** A distribution $\mathcal{D}$ over an alphabet $\Sigma$. A string $w$ over $\Sigma$.
**Question:** Compute $d_e(w, \mathcal{D})$.

If we need to only consider the decision problem we will be also taking a rational input $r$ and asking if $d_e(w, \mathcal{D}) \leq r$. And the associated approximation problem consists in computing a value $x$ such $|x - d_e(w, \mathcal{D})| < \epsilon$.

The exact status of EDD is an open question. We conjecture it is **NP**-hard.

## 3 EDD is Decidable

We first prove that there exists an algorithm which takes a string $w$ and a PFA $\mathcal{A}_{\mathcal{D}}$ and computes $d_e(w, \mathcal{A}_{\mathcal{D}})$. The computation cannot be bounded by a polynomial, but it terminates. The construction we propose follows three steps:

1. We first (Sect. 3.1) build from $w$ an MA $\mathcal{A}_w$ which can compute $d_r(w,x)$.
2. We next (Sect. 3.2) build from $\mathcal{A}_\mathcal{D}$ and $\mathcal{A}_w$ an MA $\mathcal{A}_{\mathcal{D},w}$ which computes the product of the relative edit distance and the probability of the string.
3. Using the matrix representation of $\mathcal{A}_{\mathcal{D},w}$ and $\mathcal{A}_\mathcal{D}$ we are able to compute the values of the infinite series $\sum_{x\in\Sigma^*}|x|Pr_{\mathcal{A}_\mathcal{D}}(x)$ and $\sum_{x\in\Sigma^*}d_r(w,x)Pr_{\mathcal{A}_\mathcal{D}}(x)$.

## 3.1 Building a Multiplicity Automaton Computing the Edit Distance to a String (Step 1)

Given a string $w$, we build (with Algorithm 1 MA_Build) an MA, $\mathcal{A}_w$, which will allow to parse any other string $x$ and in linear time obtain $d_r(w,x)$.

The states of the MA are the different columns one may obtain when running the classical edit distance algorithm for strings $w$ (used to index the lines) and $u$ (used to index the columns), and substracting, in each cell, the length $u$, ie, computing $d_r(w,u)$, with $w$ fixed and $u$ being any string. The number of states is finite, because $d_r(w,\cdot)\in[-|w|,|w|]$, so the number of possible columns is bounded by $(2|w|)^{|w|}$. Moreover, if we take into account that the difference between two consecutive elements in a column is in $\{-1,0,1\}$, the number of different columns, hence of states, is bounded by $3^{|w|}$.

There is a transition in the MA labelled by symbol $a$ between the state corresponding to the last column of $d_r(w,u)$ to the state corresponding to the last column of $d_r(w,ua)$, for some string $u$.

There is no guarantee that the construction terminates in polynomial time. We give an example of this construction in Appendix A (Fig. 4) and in Appendix B of [6] we provide a counter-example, ie a parameterized string such that the size of $\mathcal{A}_w$ increases faster than any polynomial in $|w|$.

Yet even when exponential, the construction does terminate, and the following result can be given:

**Proposition 1.** *Given any string $x$, $d_r(w,x)=\mathcal{A}_w(x)$*

## 3.2 Computing the Product Automaton (Step 2)

We are now given a Pfa $\mathcal{A}_\mathcal{D}=\langle\Sigma,Q_\mathcal{D},\mathbb{S}_\mathcal{D},\mathbb{F}_\mathcal{D},\delta_\mathcal{D}\rangle$ and a multiplicity automaton $\mathcal{A}_w=\langle\Sigma,Q_w,\mathbb{S}_w,\mathbb{F}_w,\delta_w\rangle$.

The new machine, denoted by $\mathcal{A}_{\mathcal{D},w}$ has as states pairs $\langle q,q'\rangle$ with $q\in Q_\mathcal{D}$, $q'\in Q_w$. $\mathcal{A}_{\mathcal{D},w}=\langle\Sigma,Q_{\mathcal{D},w},\mathbb{S}_{\mathcal{D},w},\mathbb{F}_{\mathcal{D},w},\delta_{\mathcal{D},w}\rangle$:

– $\delta_{\mathcal{D},w}(\langle q,q'\rangle,a,\langle s,s'\rangle)=\delta_\mathcal{D}(q,a,s)\delta_w(q',a,s')$,
– $\mathbb{S}_{\mathcal{D},w}(\langle q,q'\rangle)=\mathbb{S}_\mathcal{D}(q)\,\mathbb{S}_w(q')$,
– $\mathbb{F}_{\mathcal{D},w}(\langle q,q'\rangle)=\mathbb{F}_\mathcal{D}(q)\,\mathbb{F}_\mathcal{D}(q')$.

By construction, $\mathcal{A}_{\mathcal{D},w}(x)=d_r(w,x)Pr_\mathcal{A}(x)$.

An example is proposed in Appendix A, Fig. 6 of [6].

```
Algorithm MA_Build($w$)
    Data: $w = w_1 \dots w_m$ of length $m$
    Result: a multiplicity automaton $\mathcal{A}_w = \langle \Sigma, Q, \mathbb{S}, \mathbb{F}, \delta \rangle$
    $q_0 \leftarrow [0, 1, 2, \dots, m]$; $Q \leftarrow \{q_0\}$; $\mathbb{S}(q_0) \leftarrow 1$; $\mathbb{F}(q_0) \leftarrow m$;
    unmarked $\leftarrow \{q_0\}$;
    while unmarked $\neq \emptyset$ do
        Choose $q$ in unmarked ;
        unmarked $\leftarrow$ unmarked $- \{q\}$;
        for $a \in \Sigma$ do
            $q'[0] \leftarrow 0$;
            for $i = 1$ to $m$ do
                if $w_i = a$ then $x \leftarrow 0$ ;
                else $x \leftarrow 1$ ;
                $q'[i] \leftarrow \min\{q[i], q[i-1] + x - 1, q'[i-1]\}$;
            if $q' \in Q$ then $\delta(q, a, q') \leftarrow 1$ ;
            else
                $Q \leftarrow Q \cup \{q'\}$; $\delta(q, a, q') \leftarrow 1$; $\mathbb{F}(q'); \leftarrow q'[m]$; $\mathbb{S}(q') \leftarrow 0$;
                unmarked $\leftarrow$ unmarked $\cup \{q'\}$;
```

**Algorithm 1:** Algorithm MA_Build($w$) computing, given a string $w$, the deterministic MA $\mathcal{A}_w$ such that on input $x$, $d_r(w, x)$ is computed as $\mathcal{A}_w(x)$.

## 3.3 Computing the Distance (Step 3)

We have to compute $d_e(w, \mathcal{A}_\mathcal{D}) = \sum_{x \in \Sigma^*} |x| Pr_\mathcal{A}(x) + \sum_{x \in \Sigma^*} d_r(w, x) Pr_\mathcal{A}(x)$.

Let $(\Sigma, \overset{\mathcal{D}}{\mathbf{S}}, \overset{\mathcal{D}}{\mathbf{M}}, \overset{\mathcal{D}}{\mathbf{F}})$ be the matrix representation of the Pfa $\mathcal{A}_\mathcal{D}$. Since $(I - \overset{\mathcal{D}}{\mathbf{M}})$ is non-singular by definition of Pfa, the average length of the strings generated by $\mathcal{A}_\mathcal{D}$ can be computed as in [11]:

$$\sum_{x \in \Sigma^*} |x| Pr_\mathcal{A}(x) = \sum_{i=0}^{\infty} i \, Pr_\mathcal{A}(\Sigma^i) = \sum_{i=0}^{\infty} i \, \overset{\mathcal{D}}{\mathbf{S}} \overset{\mathcal{D}}{\mathbf{M}}_\Sigma{}^i \overset{\mathcal{D}}{\mathbf{F}} = \overset{\mathcal{D}}{\mathbf{S}} \overset{\mathcal{D}}{\mathbf{M}}_\Sigma (I - \overset{\mathcal{D}}{\mathbf{M}}_\Sigma)^{-2} \overset{\mathcal{D}}{\mathbf{F}}$$

Let $(\Sigma, \overset{w}{\mathbf{S}}, \overset{w}{\mathbf{M}}, \overset{w}{\mathbf{F}})$ be the matrix representation of $\mathcal{A}_{D,w}$. Each addend of the series $\sum_{x \in \Sigma^*} d_r(w, x) Pr_\mathcal{A}(x)$ can be computed as:

$$\sum_{x \in \Sigma^*} d_r(w, x) Pr_\mathcal{A}(x) = \sum_{x \in \Sigma^*} \overset{w}{\mathbf{S}} \overset{w}{\mathbf{M}}_x \overset{w}{\mathbf{F}} = \sum_{i=0}^{\infty} \overset{w}{\mathbf{S}} \overset{w}{\mathbf{M}}{}^i \overset{w}{\mathbf{F}} = \overset{w}{\mathbf{S}} (I - \overset{w}{\mathbf{M}})^{-1} \overset{w}{\mathbf{F}}$$

One point to check is that the matrix $(I - \overset{w}{\mathbf{M}})$ is non-singular.

By construction, $[\overset{w}{\mathbf{M}}]_{i,j} \geq 0$. Moreover, in any adjacency matrix, $[\mathbf{M}^k]_{i,j}$ is the sum of the weights of all the paths of length exactly $n$ that goes from node $i$ to node $j$. In our case, by construction, $[\overset{\mathcal{D}}{\mathbf{M}}{}^k]_{i,j} = \sum_{q,s} [\overset{w}{\mathbf{M}}{}^k]_{<i,q>,<j,s>}$ hence $[\overset{\mathcal{D}}{\mathbf{M}}{}^k]_{i,j} \geq [\overset{w}{\mathbf{M}}{}^k]_{<i,q>,<j,s>}$. We also know that $(I - \overset{\mathcal{D}}{\mathbf{M}})$ is non-singular so

$\lim_{k\to\infty}[\overset{\mathcal{D}}{\mathbf{M}}^k]_{i,j} = 0$. Summarising, we have that, $0 \leq \lim_{k\to\infty}[\overset{w}{\mathbf{M}}^k]_{<i,q>,<j,s>} \leq \lim_{k\to\infty}[\overset{\mathcal{D}}{\mathbf{M}}^k]_{i,j} = 0$, so $\lim_{k\to\infty}[\overset{w}{\mathbf{M}}^k]_{i,j} = 0$ and then, $(I - \overset{w}{\mathbf{M}})$ is non-singular.

Therefore, $d_e(w, \mathcal{A}_{\mathcal{D}}) = \overset{\mathcal{D}}{\mathbf{S}}\overset{\mathcal{D}}{\mathbf{M}}_{\Sigma}(I - \overset{\mathcal{D}}{\mathbf{M}}_{\Sigma})^{-2}\overset{\mathcal{D}}{\mathbf{F}} + \overset{w}{\mathbf{S}}(I - \overset{w}{\mathbf{M}})^{-1}\overset{w}{\mathbf{F}}$. It follows:

**Theorem 1.** *EDD is decidable and the edit distance between a witness string and a* PFA *with rational weights is rational.*

The construction described here is not polynomially bounded. The final computation is (with arbitrary precision and unit computation time for all arithmetic operations) cubic in the size of the product finite state machine. In turn, the size of this machine essentially depends on the length of the input string.

## 4 An FPRAS for EDD

As can be seen in the experiments (or in the theoretical analysis from Appendix B of [6]), the method described in Sect. 3 may lead to a combinatorial explosion during the construction of $\mathcal{A}_w$. In this section we propose an FPRAS to approximate the value of $d_e(w, \mathcal{A}_{\mathcal{D}})$.

Alternatively, the result can be seen as a Probably Approximate Correct (PAC) algorithm [25]. The goal of this framework is to learn (in this case, to compute) a concept for which, with high probability, we obtain a sufficiently good approximation of it.

We are given a PFA $\mathcal{A}_{\mathcal{D}}$, a string $w$ and two values $\epsilon > 0$, $\delta > 0$.

An FPRAS would be an algorithm which, in time polynomial in $|\mathcal{A}_{\mathcal{D}}|$, $|w|$, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ computes a value $v$ such that, with probability at least $1 - \delta$,

$$\left| v - d_e(w, \mathcal{A}_{\mathcal{D}}) \right| \leq \epsilon$$

**Theorem 2.** *There exists an* FPRAS *computing the expected distance between a string and a distribution given by a* PFA.

*Proof.* A full description of Algorithm COMPUTE_BOUNDS is given in Appendix C of [6]. This algorithm returns $L$ which is the length at which the generation process of the PFA should be stopped. The goal is to have a polynomial limit to the length of the strings without this impacting the quality of the result. Then, for this $L$ a value $N$, also polynomial, is computed. These numbers are used by Algorithm BUILD_SAMPLE which with high probability and complexity in $O(NL)$ is going to return a correct sample. The main Algorithm EXPECTED_DISTANCE uses this sample and computes the distance.

The complexity of Algorithm BUILD_SAMPLE is in $O(NL)$. There is a (non null, but lower than $\frac{\delta}{2}$) probability that the number of generated samples is less than $N$. □

```
Algorithm Build_Sample($\mathcal{A}_\mathcal{D}$, $L$, $N$)
    Data: a Pfa $\mathcal{A}_\mathcal{D}$
    Result: a sample $S$ which, with probability $> 1 - \frac{\delta}{2}$, contains $N$ strings
    $S \leftarrow \emptyset$;
    for $i : 1 \le i \le N$ do
        generate a string of length at most $L$, using $\mathcal{A}_\mathcal{D}$ and add it to $S$. If
        during the generation the string becomes too long, generate nothing
    return $S$
```

**Algorithm 2:** Algorithm Build_Sample($\mathcal{A}_\mathcal{D}$)

```
Algorithm Expected_Distance($w$, $\mathcal{A}_\mathcal{D}$, $\epsilon$, $\delta$)
    Data: a string $w$, a Pfa $\mathcal{A}_\mathcal{D}$, $\epsilon$, $\delta$
    Result: the expected distance between $w$ and $\mathcal{A}_\mathcal{D}$
    $\langle N, L\rangle \leftarrow$ Compute_Bounds($\mathcal{A}_\mathcal{D}, \epsilon, \delta, w$) ;
    $S \leftarrow$ Build_Sample($\mathcal{A}_\mathcal{D}$, $L$, $N$);
    $Res \leftarrow 0$;
    for $x \in S$ do $Res \leftarrow Res + d_e(w, x)$;
    return $Res/N$
```

**Algorithm 3:** Algorithm Expected_Distance($w$, $\mathcal{A}_\mathcal{D}$, $\epsilon$, $\delta$)

## 5 Experiments

As a preliminary evaluation, we ran our Fpras with a fixed value of $\delta = 0.01$ and varying values of $\epsilon$ on 100 pairs of Pfa and strings $w$. In all cases, the difference between the real value and the one computed with the Fpras was always less than $\epsilon$, which confirms that the values computed by Compute_Bounds represent a pessimistic lower bound.

In the series of experiments we want to empirically confirm the time complexity of the algorithms. We showed that the MA-based method grows with the size of the witness string, whereas the Fpras is bounded by $N$, the number of necessary samples, which is closely related to the expected length of a string from the Pfa. In order to focus these experiments on the most relevant issues, we are using the small Pfa shown in Fig. 1. Parameter $p_f \in (0, 0.9)$ allows us to tune the expected lengths nicely: the lower $p_f$, the higher the length.

The first experiment examines the time complexity using the method described in Sect. 3. Parameter $p_f$ varies so that the expected lengths of the strings are 6.22, 7.75, 9.71, 12.33, and 16. For this experiment, we generate strings randomly and uniformly of lengths ranging from 1 to 13 from an alphabet of size 2. Then, we measure the execution time consumed to compute the distance between the string and the Pfa, including the construction of the $\mathcal{A}_w$. The experiment is repeated 100 times for each Pfa and each witness string length considered. Average results are shown in Fig. 2.

As expected, the complexity of the procedure grows very fast as the length of the witness string is increased. Note that the $y$-axis is shown in logarithmic

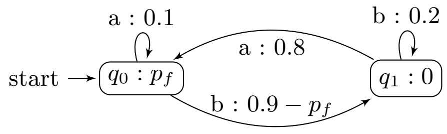

**Fig. 1.** Parametrizable Pfa used in the experiments.

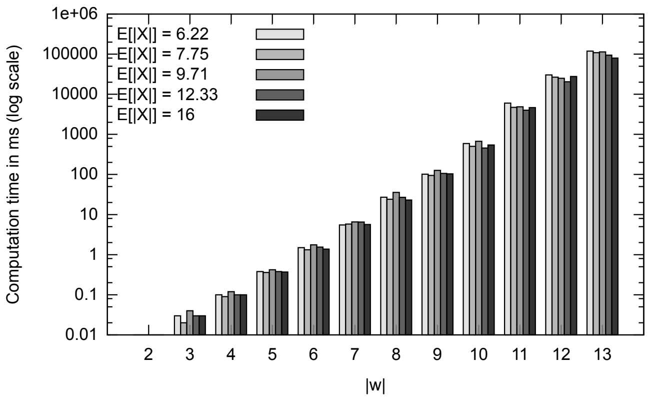

**Fig. 2.** Average execution time in milliseconds needed for the method based on a MA to compute the distance between a witness string $w$ and a Pfa.

scale, so the curve suggests an exponential growth. According to the empirical curve shown in these experiments, computing the distance of a witness string of length 50 would mean an execution time in the order of $10^7$ years. Another thing to remark is that this method is not dependent upon the configuration of the Pfa, as long as its number of states does not change.

On the other hand, the same experiments is repeated using the Fpras for $\epsilon$ and $\delta$ fixed to 0.01. The length of the witness strings are 4, 7, 10, 13 and 16. The parameter $p_f$ is configured so that the expected length of the strings of the Pfa are 7.75, 12.33, 16, 21.5, 30.66, 36.31, 44, and 49. Figure 3 illustrates the average results of these experiments.

It can be noticed that the configuration of the Pfa is the most relevant factor for the Fpras. As might be expected by the relationship between the size of the Pfa measured by $c_{\mathcal{A}}$ and the expected length of the strings, the empirical growth seems to be polynomial [3]. It is also observed that the length of the witness string is a factor that can vary the time complexity since the distances to compute are more expensive. Nevertheless, it is important to emphasize that the Fpras scales relatively well: for a Pfa whose expected length of strings is 100 and a witness string of length 100, the result (89.3235) was computed in around 25 minutes fixing both $\epsilon$ and $\delta$ to 0.01.

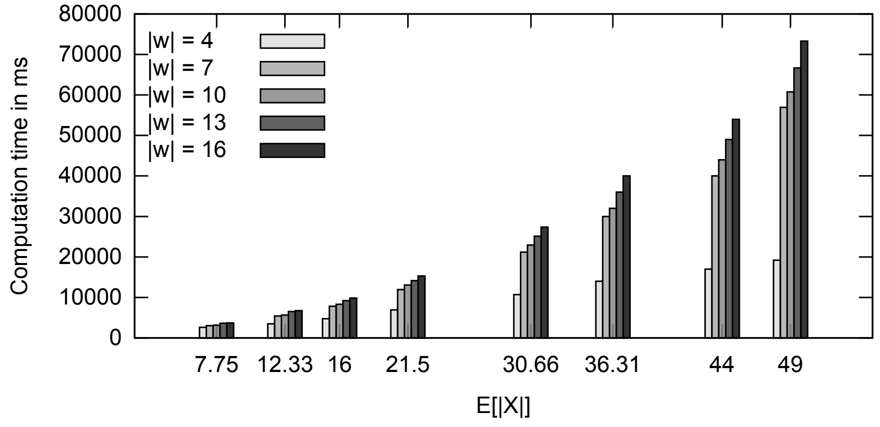

**Fig. 3.** Average execution time in milliseconds needed for the FPRAS method to compute the distance between a witness string $w$ and a PFA.

## 6 Conclusion

Two algorithms have been proposed to deal with the question of computing the expected edit distance from a witness string to a distribution given by a PFA. Whereas one is able to compute this value exactly, it is limited to cases where the length of the witness string is short, as the construction involves building a multiplicity whose size can increase in an exponential way with the length of this string. The first one also shows that the question is decidable and that the solution can be expressed with rational weights.

On the other hand we have a FPRAS which will return with high confidence a value ($\epsilon$)-close to the correct result. It has been shown that its complexity essentially depends on the expected length of the strings of the distribution.

The above results raise several extra questions:

- Computing the expected edit distance between two PFA. In [17] a technique is proposed for the special cases where these PFA correspond to finite languages, or can be determinized. A randomized technique in which strings are drawn from both distributions is likely to work.
- The exact status of EDD remains unclear. The (decision) problem is decidable, as witnessed by Theorem 1. But is it in **NP**?
- A more technical puzzling question concerns the size of the multiplicity automaton built in Sect. 3. The experiments and the construction proposed in Appendix B of [6] shows that polynomial bounds are not going to be met. But the proof relies on an alphabet whose size increases with the length of string $w$. Having a construction with a fixed size (ideally 2) is an open question.

More importantly, the really crucial question is that of computing the median string, given a PFA.

When given a distribution, a prediction system will often attempt to return the most probable string in order to minimize the empirical risk by following a *maximum a posteriori probability* (MAP) criterion.

Nevertheless, while this may be applicable in a large number of applications, other loss functions can be better suited than the 0/1 loss. For instance, very often the final goal is to reduce the number of post-processing corrections required to transform a hypothesis. This is usually counted by means of the Levenshtein or edit distance ($d_e$), or a related metric like the Word Error Rate (WER). Then, the empirical risk becomes

$$R(w|x) = \sum_{v \in \Sigma^*} Pr(v|x) d_e(w, v)$$

In which case, the optimum string is the *median string*. Yet most often the most probable string (or an approximation of it) is proposed instead of the median string, whose search is related to a $\mathcal{NP}$-hard problem [9] even in a finite case. This inconsistency is well known [12], and there have been a number of studies addressing this issue [8, 21], with recently a specific analysis of the relationship between 0/1 loss functions and other discrete loss functions [22]. Other approaches include the introduction of heuristics to approximate the median string [13, 15, 1].

This constitutes of course a real challenge.

**Acknowledgements.** The authors wish to acknowledge the help of Borja Balle in establishing the proof of Theorem 2, and the comments of the 3 anonymous reviewers of this paper. Also the financial help of the Spanish Ministerio de Educación, Cultura y Deporte through a FPU grant (Ref. AP2012-0939) and the Spanish Ministerio de Economía y Competitividad through Project No. TIN2013-48152-C2-1-R (supported by UE FEDER funds). This work was partly done while the second author was supported by the University of Kyoto.

## References

1. Abreu, J., Rico-Juan, J.R.: A new iterative algorithm for computing a quality approximate median of strings based on edit operations. Patt. Rec. Letters 36, 74–80 (2014)
2. Allauzen, C., Mohri, M.: Linear-Space Computation of the Edit-Distance between a String and a Finite Automaton. CoRR abs/0904.4686 (2009)
3. Balle, B.: Learning Finite-State Machines: Algorithmic and Statistical Aspects. Ph.D. thesis, Universitat Politécnica de Catalunya (2013)
4. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Learning balls of strings from edit corrections. Journal of Machine Learning Research 9, 1841–1870 (2008)
5. Berstel, J., Reutenauer, C.: Rational Series and their Languages. Springer-Verlag (1988)

6. Calvo-Zaragoza, J., de la Higuera, C., Oncina, J.: Computing the expected edit distance from a string to a pfa. Complete version with appendices. (2016), `https://hal.archives-ouvertes.fr/hal-01308549`
7. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological Sequence Analysis: Probalistic Models of Proteins and Nucleic Acids. Cambridge University Press, Cambridge, UK (1998)
8. Ehling, N., Zens, R., Ney, H.: Minimum Bayes risk decoding for BLEU. In: In Proc. 45th Annual Meeting of the Assoc. for Computational Linguistics (ACL) (2007)
9. de la Higuera, C., Casacuberta, F.: Topology of strings: median string is NP-complete. Theoretical Computer Science 230, 39–48 (2000)
10. de la Higuera, C., Oncina, J.: Computing the Most Probable String with a Probabilistic Finite State Machine. In: Proceedings of Fsmnlp (2013)
11. de la Higuera, C., Oncina, J.: The most probable string: an algorithmic study. Journal of Logic and Computation 24(2), 311–330 (2014)
12. Jelinek, F.: Statistical Methods for Speech Recognition. MIT Press, Cambridge, MA, USA (1997)
13. Kruzslicz, F.: Improved Greedy Algorithm for Computing Approximate Median Strings. Acta Cybernetica 14(2) (Dec 1999)
14. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR 163(4), 845–848 (1965)
15. Martínez-Hinarejos, C.D., Juan, A., Casacuberta, F.: Median strings for k-nearest neighbour classification. Patt. Rec. Lett. 24(1-3), 173–181 (2003)
16. Mihov, S., Schulz, K.U.: Fast Approximate Search in Large Dictionaries. Computational Linguistics 30(4), 451–477 (2004)
17. Mohri, M.: Edit-Distance Of Weighted Automata: General Definitions and Algorithms. Int. J. Found. Comput. Sci. 14(6), 957–982 (2003)
18. Motwani, R., Raghavan, P.: Randomized algorithm. Springer, Berlin (1995)
19. Navarro, G., Raffinot, M.: Flexible pattern matching. Cambridge University Press, Cambridge, UK (2002)
20. Paz, A.: Introduction to probabilistic automata. Academic Press, New York (1971)
21. Schluter, R., Nussbaum-Thom, M., Ney, H.: On the Relationship Between Bayes Risk and Word Error Rate in ASR. Audio, Speech, and Language Processing, IEEE Transactions on 19(5), 1103–1112 (July 2011)
22. Schluter, R., Nussbaum-Thom, M., Ney, H.: Does the Cost Function Matter in Bayes Decision Rule? Pattern Analysis and Machine Intelligence, IEEE Transactions on 34(2), 292–301 (2012)
23. Schulz, K.U., Mihov, S.: Fast string correction with Levenshtein automata. IJDAR 5(1), 67–85 (2002)
24. Stolcke, A., Konig, Y., Weintraub, M.: Explicit Word Error Minimization in N-Best List Rescoring. In: 5th. European Conference on Speech Communication and Technology (1997)
25. Valiant, L.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)
26. Vazirani, V.: Approximation Algorithms. Springer, Berlin (2003)
27. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic Finite State Automata – Part I and II. Pattern Analysis and Machine Intelligence 27(7), 1013–1039 (2005)
28. Wagner, R.A.: Order-$n$ correction for regular languages. Communications of the ACM 17(5), 265—-268 (1974)