Index Terms

Automata (F.1.1.a), Classes defined by grammars or automata (F.4.3.b), Machine learning (I.2.6.g), Language acquisition (I.2.6.h), Language models (I.2.7.c), Language parsing and understanding (I.2.7.d), Machine translation (I.2.7.f), Speech recognition and synthesis (I.2.7.g), Structural Pattern Recognition (I.5.1.f), Syntactic Pattern Recognition (I.5.1.g).

Probabilistic Finite-State Machines – Part I

E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta and R. C. Carrasco

Abstract

Probabilistic finite-state machines are used today in a variety of areas in pattern recognition, or in fields to which pattern recognition is linked: computational linguistics, machine learning, time series analysis, circuit testing, computational biology, speech recognition and machine translation are some of them. In part I of this paper we survey these generative objects and study their definitions and properties. In part II, we will study the relation of probabilistic finite-state automata with other well known devices that generate strings as hidden Markov models and n-grams, and provide theorems, algorithms and properties that represent a current state of the art of these objects.

I. Introduction

Probabilistic finite-state machines such as probabilistic finite-state automata (PFA) [1], hidden Markov models (HMMs) [2], [3], stochastic regular grammars [4], Markov chains [5], n-grams [3], [6], probabilistic suffix trees [7], deterministic stochastic or probabilistic automata (DPFA) [4], weighted automata [8] are some names of syntactic objects which during the past years have attempted to model and generate distributions over sets of possible infinite cardinality of strings, sequences, words, phrases but also terms and trees.

Their successes in a wide amount of fields ranging from computational linguistics [8] to pattern recognition [9]–[12], and including language modeling in speech recognition [2], [3], [13], bioinformatics [14]–[17], music modeling [18], machine translation [8], [19]–[26], circuit testing [27] or time series analysis [28] make these objects very valuable indeed. But as more and more researchers have entered this field, definitions and notations have

Dr. Vidal and Dr. Casacuberta are with Dto. Sistemas Informáticos y Computación and Instituto Tecnológico de Informática. Universitat Politècnica de València. Spain.

Dr. de la Higuera and Dr. Thollard are with EURISE and the Université Jean Monnet. France.

Dr. Carrasco is with Dto. de Lenguajes y Sistemas Informáticos. Universidad de Alicante. Spain.

varied and not enough energy has been spent to reach a common language. For the outsider, the choice of the best fitting syntactic object to describe the sort of distribution she/he is working on will seldom depend on anything else than the usual knowledge in the sub-field or on her/his own background.

There has been a number of survey papers dedicated to one or another of these models during the past thirty years [8], [29]–[33], but it is not always obvious through reading these papers how the models inter-relate, and where the difficulties lie. These difficulties have been theoretically analyzed in the computational learning theory literature [34]–[38]; alas, these results, highly technical, seem not to have reached the adequate communities. A possible exception is the very recent article by Dupont et al. [39].

Furthermore, more and more folk theorems appear: HMMS might be equivalent to PFA, parsing a string in the non-deterministic case by taking the best derivation (instead of summing up over the possible derivations) could be a good approximation; determinism might not (as in common language theory) modify the expressive power of PFA. Some of these results are true, others are not. And even in the case of the true "folk theorems", most researchers would not know why they hold.

The complexity of the objects themselves, and moreover of the underlying theories (for instance probabilities, matrix calculus, rational power series), makes many of the usual results depend on some exterior theory: For example, consider the question (studied in section IV-C) of knowing if the mean of two regular deterministic distributions is also regular deterministic. If this was so, we could merge distributions using DPFA. But this is false and a proof can be given using results on rational series. We argue that such a proof (albeit simpler than the one we propose) offers little insight for people working in the field. Knowing how to construct the counter-example is of much more use: It helps for instance to build hard cases that can be used for other problems, or to identify a sub-class of distributions where the counter-example will not hold.

The above example gives the spirit in which the paper is written. It aims to provide an up to date theory of PFA, but also a survey where the objects themselves give the answers to the questions that naturally arise.

Another preliminary question is that of justifying our interest in PFA to describe distributions rather than some other devices, among which the most popular may be the HMMs. Our choice of centering the survey on PFA instead of HMMs obeys to at least 3 reasons:

- Formal language theory appears to be today a widespread background knowledge to researchers and engineers in computer science. Adding probabilities to well known objects as automata permits to build on our intuitions and experiences. On the other hand HMMs are directly issued from probability theory. This parentage also affects the way the theory is constructed. PFA are built to deal with the problem of probabilizing a structured space by adding probabilities to structure, whereas HMMs might rather be considered as devices that structure probabilistic spaces by adding structure to probabilities. Neither choice is fundamentally better, but if concerned with a task where one wishes to use probabilistic devices in order to grasp the structure of the data, the first one seems more appropriate.
- As we will prove in the second part of our paper [40], PFA can represent the same distributions class as those modeled by the HMMs defined in that section. Furthermore, they can do so in at most as much space, and the common algorithms are at least as simple.
- A third point is that as PFA are finite-state automata with weights that verify some constraints, then if the underlying automaton is deterministic, we have a deterministic probabilistic finite-state automaton (DPFA). In formal language theory, there is a key difference between deterministic and non-deterministic finite-state machines which extends to the probabilistic case: DPFA are very much favored because parsing with them is simpler, and also because they admit a minimal object, which in turn makes the equivalence problem tractable. A probabilistic deterministic machine also exists, which we will study with special attention. Even if these machines are not as powerful as their non-deterministic counterpart, they play an important role in a number of applications.

Our first objective will be to establish correct definitions for the different sorts of probabilistic automata; this will be done in section II. The probabilistic automata we consider in this paper are generative processes. It should be noted that in the line of [41] probabilistic acceptors have also been studied.

A simple problem as that of parsing can be upsetting: We provide in section III all required equations and algorithms dealing with parsing. The goal of the section is to study the relationship between the PFA and the strings they generate [42], [43].

Section IV is devoted to study the intrinsic properties of PFA. Minimality issues are discussed in section IV-A. In section IV-B we will prove that there are distributions that cannot be represented by DPFA, whereas they can by PFA.

Topology over regular distributions will be thoroughly studied in section V. On the one hand entropy-based measures such as the Kullback-Leibler divergence or the perplexity can arguably measure the quality of a model. On the other hand, alternative mathematical distances [16], [17], [44] can be used. Some of them can effectively be computed over the representants of the distributions, at least when these are deterministic.

Part II [40] of the paper will be devoted to the comparison with other types of models, learning issues and the presentation of some of the extensions of the probabilistic automata.

In order to make the manuscript more readable the proofs of the propositions and theorems are left to the corresponding appendixes.

As all surveys this one is incomplete. In our case the completeness is particularly difficult to achieve due to the enormous and increasing amount of very different fields where these objects have been used. In advance we would like to apologize to all those whose work on the subject we have not recalled.

II. DEFINITIONS

Probabilistic finite-state automata are chosen as *key* syntactic representations of the distributions for a certain amount of reasons:

- Formal language theory appears to be today one of the most widespread background knowledges to researchers and engineers in computer science.
- PFA can represent the same distributions as those modeled by some HMMs.
- PFA admit a deterministic version for which most natural problems become tractable; Even though non-deterministic PFA are not equivalent to their deterministic counterparts, these (DPFA) have been studied by a number of authors because of their particular properties.
- In practice, PFA can be used to implement other finite-state models.

There is a variety of definitions regarding PFA in the literature. The ones we choose to give here are sufficiently general to cover most cases where the intended distribution is over the set of all strings (and not just the set of strings of some special length). The cases that do not fit in this definition will be analyzed in the second part of our paper [40].

In the general definition of such automata the probabilities are real numbers but, as they are intended for practical use, the probabilities are rather represented as rational numbers. Also, rational probabilities are needed for discussing about computational properties involving the concept of size of an automaton. A different line was successfully followed in [8], where the probabilities are just a special case of abstract weights: The algebra over which the weights are computed then allows to deal with all cases, whether computable or not.

We now give the formal definitions of probabilistic automata we are going to use in the rest of the paper.

A. Stochastic languages

Let Σ be a finite alphabet and $\Sigma^\star$ the set of all strings that can be built from Σ, including the empty string denoted by λ.

A *language* is a subset of $\Sigma^\star$. By convention, symbols in Σ will be denoted by letters from the beginning of the alphabet $(a, b, c, ...)$ and strings in $\Sigma^\star$ will be denoted by end of the alphabet letters $(..., x, y, z)$. The length of a string $x \in \Sigma^\star$ is written $|x|$. The set of all strings of length n (resp. less than, at most n) will be denoted by Σ^n (resp. $\Sigma^{<n}$, $\Sigma^{\leq n}$). A substring of x from position i to position j will be denoted as $x_i \dots x_j$. A substring $x_i \dots x_j$ with $j < i$ is the empty string λ.

A *stochastic language* $\mathcal{D}$ is a probability distribution over $\Sigma^\star$. We denote by $\Pr_{\mathcal{D}}(x)$ the probability[1] of a string $x \in \Sigma^\star$ under the distribution $\mathcal{D}$. The distribution must verify $\sum_{x \in \Sigma^\star} \Pr_{\mathcal{D}}(x) = 1$. If the distribution is modeled by some syntactic machine $\mathcal{A}$, the probability of x according to the probability distribution defined by $\mathcal{A}$ is denoted $\Pr_{\mathcal{A}}(x)$. The distribution modeled by a machine $\mathcal{A}$ will be denoted $\mathcal{D}_{\mathcal{A}}$ and simplified to $\mathcal{D}$ in a non-ambiguous context.

If L is a language over Σ, and $\mathcal{D}$ a distribution over $\Sigma^\star$, $\Pr_{\mathcal{D}}(L) = \sum_{x \in L} \Pr_{\mathcal{D}}(x)$

[1] As usual, we will use the notation of $\Pr(x)$ as $\Pr(X = x)$ and $\Pr(x \mid y)$ as $\Pr(X = x \mid Y = y)$ for any random variables X and Y.

A *sample* S is a multi-set of strings: as they are usually built through sampling, one string may appear more than once. We will write $x \in S$ to indicate that (all instances of the) string x is (are) represented in the sample. The *size* $|S|$ of sample S, is the total number of strings in the sample and $\|S\|$ is the total sum of lengths of the all strings in S. It should be noted that neither of these measures of sizes corresponds to the actual number of bits needed to encode a sample. The empirical finite-support distribution associated with S will be denoted as $\mathcal{D}_S$; i.e. $\Pr_{\mathcal{D}_S}(x) = f(x)/|S|$, where $f(x)$ is the frequency (number of repetitions) of x in S and $\Pr_{\mathcal{D}_S}(x) = 0$ if $x \notin S$.

B. Probabilistic automata

We present in this section formal definitions about probabilistic automata. These are directly inspired by a number of works in machine learning and pattern recognition, including [1], [4], [13], [45]–[47].

Definition 1: A PFA is a tuple $\mathcal{A}$=$\langle Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}}, P_{\mathcal{A}}\rangle$, where:

- $Q_{\mathcal{A}}$ is a finite set of *states*;
- Σ is the alphabet;
- $\delta_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \Sigma \times Q_{\mathcal{A}}$ is a *set of transitions*;
- $I_{\mathcal{A}} : Q_{\mathcal{A}} \to \mathbb{R}^+$ (initial-state probabilities);
- $P_{\mathcal{A}} : \delta_{\mathcal{A}} \to \mathbb{R}^+$ (transition probabilities);
- $F_{\mathcal{A}} : Q_{\mathcal{A}} \to \mathbb{R}^+$ (final-state probabilities).

It should be noted that probabilities may be null ($0 \in \mathbb{R}^+$) and therefore functions $I_{\mathcal{A}}$, $F_{\mathcal{A}}$ and $P_{\mathcal{A}}$ can be considered as total. Similarly, for the sake of notation simplification, $P_{\mathcal{A}}$ is assumed to be extended with $P_{\mathcal{A}}(q, a, q') = 0$ for all $(q, a, q') \notin \delta_{\mathcal{A}}$.

$I_{\mathcal{A}}$, $P_{\mathcal{A}}$ and $F_{\mathcal{A}}$ are functions such that:

$$\sum_{q \in Q_{\mathcal{A}}} I_{\mathcal{A}}(q) = 1,$$

and

$$\forall q \in Q_{\mathcal{A}},\ F_{\mathcal{A}}(q) \ + \sum_{a \in \Sigma,\ q' \in Q_{\mathcal{A}}} P_{\mathcal{A}}(q, a, q') \ = \ 1.$$

In what follows, the subscript $_{\mathcal{A}}$ will be dropped when there is no ambiguity. A generic state of Q will be denoted by q without subindex, the specific states in Q will be denoted as $q_0, q_1 \ldots, q_{|Q|-1}$ and a sequence of states of length j will be denoted by $(s_1, s_2, \ldots, s_j)$, where $s_i \in Q$ for $1 \leq i \leq j$.

As will be seen in the next section, the above automata definition corresponds to models which are *generative* in nature. This is in contrast with the standard definition of automata in the conventional (non probabilistic) formal language theory, where strings are generated by *grammars* while the automata are the *accepting* devices. It is not difficult to prove that the definition adopted in this article is equivalent to the definition of *stochastic regular grammar* [9], [32]. From a probabilistic point of view, the process of (randomly) accepting a *given* string is essentially different from the process of generating a (random) string. Probabilistic acceptors are defined in [9], [41], but they have only seldom been considered in syntactic pattern recognition or in (probabilistic) formal language theory.

Typically, PFA are represented as directed labeled graphs. Figure 1 shows of a PFA with four states, $Q = \{q_0, q_1, q_2, q_3\}$, only one initial state (i.e. a state q with $I(q) > 0$), q_0, and a four-symbol alphabet, $\Sigma = \{a, b, c, d\}$. The real numbers in the states and in the arrows are the final-state and the transition probabilities, respectively.

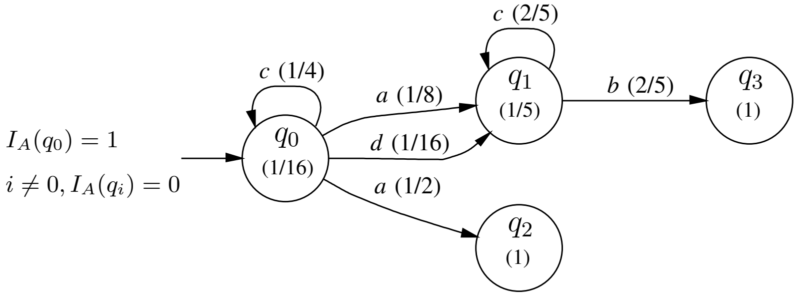

Fig. 1. Graphical representation of a PFA.

A particular case of PFA arises when the underlying graph is acyclic. This type of models are known as *acyclic probabilistic finite-state automata* (APFA) [48]. On the other hand, a more general model is defined in the next subsection.

C. λ-Probabilistic finite-state automata (λ-PFA)

Definition 2:

A λ-PFA $\mathcal{A}$ is a tuple $\langle Q, \Sigma, \delta, I, F, P \rangle$, where Q, Σ, I and F are defined as for PFA, but δ is extended to $\delta \subset Q \times (\Sigma \cup \{\lambda\}) \times Q$.

P and F verify a similar normalization as for PFA with the sum for all $a \in \Sigma$ extended to include λ:

$$\forall q \in Q,\ F(q)\ + \sum_{a \in \Sigma \cup \{\lambda\},\ q' \in Q} P(q,a,q')\ =\ 1.$$

λ-PFA appear as natural objects when combining distributions. They are nevertheless not more powerful than PFA in the sense that they generate the same distributions (see section IV-C). λ-PFA introduce specific problems, in particular, when sequences of transitions labeled with λ are considered. In section III-C some of these problems are analyzed. When considering λ-PFA a few concepts will be needed:

Definition 3:

For any λ-PFA $\mathcal{A} = \langle Q, \Sigma, \delta, I, F, P \rangle$

- A λ-transition is any transition labeled by λ;
- A λ-loop is a transition of the form (q, λ, q);
- A λ-cycle is a sequence of λ-transitions from δ: $(s_{i_1}, \lambda, s_{i_2}), (s_{i_2}, \lambda, s_{i_3}), \ldots, (s_{i_k}, \lambda, s_{i_1})$, where $\forall j : 0 < j < k \ \ (s_{i_j},\, \lambda, s_{i_{j+1}}) \in \delta$.

D. Deterministic probabilistic finite-state automata (DPFA)

Even though determinism (as we shall show later) restricts the class of distributions that can be generated, we introduce deterministic probabilistic finite-state automata because of the following reasons:

- Parsing is easier as only one path has to be followed.
- Some intractable problems (finding the most probable string, comparing two distributions) become tractable.
- There are a number of positive learning results for DPFA that do not hold for PFA.

Definition 4:

A PFA $\mathcal{A} = \langle Q, \Sigma, \delta, I, F, P \rangle$ is a DPFA, if:

- $\exists q_0 \in Q$ (*initial state*), such that $I(q_0) = 1$;
- $\forall q \in Q,\ \forall a \in \Sigma, \quad |\, \{q' : (q,a,q') \in \delta\} \,| \ \ \leq\ 1.$

In a DPFA, a transition (q, a, q') is completely defined by q and a and a DPFA can be more simply denoted by $\langle Q, \Sigma, \delta, q_0, F, P \rangle$.

A particular case of DPFA is the *probabilistic prefix tree automaton* (PPTA) where the underlying graph is a tree rooted at the initial state q_0.

E. Size of a PFA

If PFA are to be implemented then we are concerned with two issues. On the one hand all probabilities have to be encoded and thus the range of functions I, F and P should be $\mathbb{Q}^+$ instead of $\mathbb{R}^+$. A second point is that in order to compute the complexity of an algorithm, we must be able to give the size of a PFA (or DPFA, λ-PFA). The complexity should be polynomially linked with the number of bits needed to encode the PFA in a reasonable way. It follows that in the case of DPFA a correct measure of the size is the sum of the number n of states, the size $|\Sigma|$ of the alphabet and the number of bits needed to encode all the non null probabilities in the automaton. In the case of PFA or λ-PFA, because of non-determinism, the number of transitions in the automaton should also appear.

F. Distributions modeled by PFA

PFA are stochastic machines that may not generate a probability space but a subprobability space over the set of finite-strings $\Sigma^\star$. Given a PFA (or λ-PFA) $\mathcal{A}$, the process of generating a string proceeds as follows:

- Initialization: Choose (with respect to a distribution I) one state q_0 in Q as the initial state. Define q_0 as the current state.
- Generation: Let q be the current state. Decide whether to *stop*, with probability $F(q)$, or to produce a *move* (q, a, q') with probability $P(q, a, q')$ where $a \in \Sigma \cup \{\lambda\}$ and $q' \in Q$. Output a and set the current state to q'.

In some cases, this process may never end; i.e. it may generate strings of unbounded length (see section II-G). If PFA generates finite-length strings, a relevant question is that of computing the probability that a PFA $\mathcal{A}$ generates a string $x \in \Sigma^\star$. To deal with this problem, let $\theta = (s_0, x'_1, s_1, x'_2, s_2, \ldots, s_{k-1}, x'_k, s_k)$ be a *path* for x in $\mathcal{A}$; that is, there is a sequence of transitions (s_0, x'_1, s_1), (s_1, x'_2, s_2), $\ldots, (s_{k-1}, x'_k, s_k) \in \delta$ such that $x = x'_1 x'_2 \ldots x'_k$ (note that in general $|x| \leq k$ because some x'_j can be λ). To simplify the notation, the symbols x'_j in the sequences of transitions will be omitted if not needed.

The probability of generating such a path is:

$$\mathrm{Pr}_{\mathcal{A}}(\theta) = I(s_0) \cdot \left(\prod_{j=1}^{k} P(s_{j-1}, x'_j, s_j) \right) \cdot F(s_k). \tag{1}$$

Definition 5: A *valid path* in a PFA $\mathcal{A}$ is a path for some $x \in \Sigma^\star$ with probability greater than zero. The set of valid paths in $\mathcal{A}$ will be denoted as $\Theta_{\mathcal{A}}$.

In general, a given string x can be generated by $\mathcal{A}$ through multiple valid paths. Let $\Theta_{\mathcal{A}}(x)$ denote[2] the set of all the valid paths for x in $\mathcal{A}$. The *probability of generating x with $\mathcal{A}$* is

$$\mathrm{Pr}_{\mathcal{A}}(x) = \sum_{\theta \in \Theta_{\mathcal{A}}(x)} \mathrm{Pr}_{\mathcal{A}}(\theta). \tag{2}$$

If $\sum_x \mathrm{Pr}_{\mathcal{A}}(x) = 1$, then $\mathcal{A}$ defines a distribution $\mathcal{D}$ on $\Sigma^\star$; otherwise the model does not have much interest. The conditions which guarantee this will be discussed in section II-G.

A probabilistic finite-state automaton is *ambiguous* if a string x exists such that $\mid \Theta_{\mathcal{A}}(x) \mid > 1$.

For the PFA of figure 1 there is only one *valid path* for the string $accb$: $\Theta_{\mathcal{A}}(accb) = \{(q_0, a, q_1, c, q_1, c, q_1, b, q_3)\}$. The probability of $accb$ is :

$$\begin{aligned} \mathrm{Pr}_{\mathcal{A}}(accb) &= I(q_0) \cdot P(q_0, a, q_1) \cdot P(q_1, c, q_1) \cdot P(q_1, c, q_1) \cdot P(q_1, b, q_3) \cdot F(q_3) \\ &= 1.0 \cdot 0.125 \cdot 0.4 \cdot 0.4 \cdot 0.4 \cdot 1.0 \\ &= 0.008\ . \end{aligned}$$

For the string a, there are two *valid paths*: $\Theta_{\mathcal{A}}(a) = \{(q_0, a, q_1), (q_0, a, q_2)\}$. Therefore, the PFA of figure 1 is ambiguous. The probability of a is then:

$$\begin{aligned} \mathrm{Pr}_{\mathcal{A}}(a) &= I(q_0) \cdot P(q_0, a, q_1) \cdot F(q_1) \ + I(q_0) \cdot P(q_0, a, q_2) \cdot F(q_2) \\ &= 1.0 \cdot 0.125 \cdot 0.2 + 1.0 \cdot 0.5 \cdot 1.0 \\ &= 0.525\ . \end{aligned}$$

The definition of DPFA directly yields:

Proposition 1: No DPFA is ambiguous.

We conclude this section by defining classes of string distributions on the base of the corresponding generating automata.

[2] In unambiguous context, $\Theta_{\mathcal{A}}(z)$ will be extended in section III to also mean the set of sub-paths that generate a substring z, these sub-paths will be allowed to start or end in states with null initial or final probabilities, respectively.

Definition 6: A distribution is *regular* if it can be generated by some PFA.

An alternative definition could be used: A regular distribution is a probabilistic distribution on a regular language. However, we do not assume this definition because it would present the following problem: There would exist regular distributions which could not be generated by any PFA. This result can be easily derived from [32].

Definition 7: A distribution is *regular deterministic* if it can be generated by some DPFA.

Definition 8: Two PFA are *equivalent* if they generate the same distribution.

From the definition of PFA and DPFA the following hierarchy follows:

Proposition 2: A regular deterministic distribution is also a regular distribution

The reverse of this proposition is not always true (see proposition 10, section IV).

It is interesting to note that APFA and PPTA only generate distributions on finite sets of strings. Moreover, given any finite sample S, a PPTA can be easily constructed which generates the empirical distribution $\mathcal{D}_S$ [4].

G. Consistency of PFA

The question of consistency is “do the probabilities provided by an automaton according to equation (2) sum up to 1?”. In early papers in the field the question was supposed to be simple [31] or on the contrary complex when concerned with stochastic context-free grammars; in that setting the consistency can be checked by analyzing the behavior of the underlying probability matrix [32], [49].

The conditions needed for a PFA to be consistent are established as follows [39]:

Definition 9: A state of a PFA $\mathcal{A}$ is *useful* if it appears in at least one valid path of $\Theta_{\mathcal{A}}$.

Proposition 3: A PFA is consistent if all its states are useful.

Note that the condition of proposition 3 is sufficient but not necessary: A non useful state is harmless if it is *inaccessible*; i.e., if no string can reach it with probability greater than zero.

Once the syntactic models and the corresponding string distributions have been defined, we discuss in the next section how to compute the probability of a given string in the distribution modeled by a given probabilistic automaton.

III. Parsing issues

We understand *parsing* as the computation of equation (2) in page 11. In the case of DPFA, the algorithms are simpler than for non-deterministic PFA. In the first case, the time computation cost of equation (2) (and that of equation (6) in this section) are in $\mathcal{O}(|x|)$. This computational cost does not depend on the number of states since at each step the only possible next state is computed with a cost in $\mathcal{O}(1)$. In general, as will be discussed below, the probability that a string x is generated by a PFA, given by equation (2), can be computed efficiently by using *dynamic programming*. Another problem related with parsing is the computation of the probability of a substring in a PFA [46].

A. *Parsing with* PFA

The probabilities assigned to the paths in $\Theta_{\mathcal{A}}$ (section II-F) can be used to compute efficiently $\Pr_{\mathcal{A}}(x)$. The idea is similar to the one proposed for HMMs [50] by defining $\alpha_x(i,q)\ \forall q \in Q$ and $0 \leq i \leq |x|$ as the probability of generating the prefix $x_1 \ldots x_i$ and reaching state q:[3]

$$\alpha_x(i,q) = \sum_{(s_0,s_1,\ldots,s_i)\in\Theta_{\mathcal{A}}(x_1\ldots x_i)} I(s_0) \cdot \prod_{j=1}^{i} P(s_{j-1},x_j,s_j) \cdot 1(q,s_i)\ , \tag{3}$$

where $1(q,q') = 1$ if $q = q'$ and 0 if $q \neq q'$. In this case, the extended $\Theta_{\mathcal{A}}$ to sub-paths is used.

Equation (3) can be calculated with the following algorithm:

Algorithm 3.1: Forward algorithm

$$\begin{aligned} \alpha_x(0,q) &= I(q), \\ \alpha_x(i,q) &= \sum_{q'\in Q} \alpha_x(i-1,q') \cdot P(q',x_i,q), \quad 1 \leq i \leq |x|. \end{aligned}$$

For a string $x \in \Sigma^\star$, the following proposition is straightforward:

Proposition 4:

$$\Pr_{\mathcal{A}}(x) = \sum_{q\in Q} \alpha_x(|x|,q) \cdot F(q)\ . \tag{4}$$

[3] It is assumed the notation: $\prod_{i=k}^{j} a_i = 1$, if $j < k$.

There is another way of computing $\Pr_{\mathcal{A}}(x)$ by introducing $\beta_x(i,q)$ as the probability of generating the suffix $x_{i+1}\dots x_{|x|}$ from the state q:

$$\beta_x(i,q) = \sum_{(s_i,\dots,s_{|x|})\in\Theta_{\mathcal{A}}(x_{i+1}\dots x_{|x|})} 1(q,s_i)\cdot\left(\prod_{j=i+1}^{|x|} P(s_{j-1},x_j,s_j)\right)\cdot F(s_{|x|}), \qquad (5)$$

that can be calculated by

Algorithm 3.2: Backward algorithm

$$\begin{aligned}\beta_x(|x|,q) &= F(q),\\ \beta_x(i,q) &= \sum_{q'\in Q}\beta_x(i+1,q')\cdot P(q,x_i,q'), \quad 0\le i\le |x|-1.\end{aligned}$$

And the corresponding proposition:

Proposition 5:

$$\Pr_{\mathcal{A}}(x) = \sum_{q\in Q} I(q)\cdot\beta_x(0,q).$$

The computation of α and β can be performed with a time complexity of $\mathcal{O}(|x|\cdot|\delta|)$, where $|x|$ is the length of x and $|\delta|$ is the number of transitions in $\mathcal{A}$.

B. *Searching for the optimal path for a string in a* PFA

In equation (2), the probability of generating x with $\mathcal{A}$ is defined as a sum of the probabilities of all valid paths that deal with x. However, it can be interesting to search for a valid path $\tilde{\theta}$ that generates x with highest probability,

$$\tilde{\theta} = \underset{\theta\in\Theta_{\mathcal{A}}(x)}{\operatorname{argmax}} \Pr_{\mathcal{A}}(\theta). \qquad (6)$$

The probability of this *optimal path* $\tilde{\theta}$ will be denoted as $\widetilde{\Pr}_{\mathcal{A}}(x)$. The relation between $\widetilde{\Pr}_{\mathcal{A}}(x)$ and $\Pr_{\mathcal{A}}(x)$ from equation (2) has been studied in [51] and [52]. When “good” models are used in practice, the probability given by equation (2) is often mainly distributed among a few paths close to the optimal one. In that case, the probability of the optimal path is an adequate approximation to the probability given by equation (2).

The optimal path $\tilde{\theta}$ is of practical interest in many pattern recognition applications, since useful information can be attached to the states and in many cases the problem is to search for the information that is in the optimal path. This path is also useful for an efficient estimation of the parameters of the model from a training sample (see section III-A of part II).

The computation of $\widetilde{\Pr}_{\mathcal{A}}(x)$ can be efficiently performed by defining a function $\gamma_x(i,q)$ $\forall q \in Q$, $0 \leq i \leq |x|$, as the probability of generating the prefix $x_1 \dots x_i$ through the best path and reaching state q:

$$\gamma_x(i,q) = \max_{(s_0,s_1,\dots,s_i)\in\Theta_{\mathcal{A}}(x_1\dots x_i)} I(s_0)\cdot\prod_{j=1}^{i} P(s_{j-1},x_j,s_j)\cdot 1(q,s_i), \tag{7}$$

An algorithmic solution is given by the following algorithm.

Algorithm 3.3: Viterbi algorithm

$$\begin{aligned}\gamma_x(0,q) &= I(q),\\ \gamma_x(i,q) &= \max_{q'\in Q}\gamma_x(i-1,q')\cdot P(q',x_i,q), \quad 1\leq i\leq |x|,\end{aligned}$$

with the corresponding proposition:

Proposition 6:

$$\widetilde{\Pr}_{\mathcal{A}}(x) = \max_{q\in Q}\gamma_x(|x|,q)\cdot F(q)\ .$$

The computation of γ presents the same time complexity as the computation of α or β, but the implementation of the last ones may lead to numerical precision problems, which can be easily circumvented in the implementation of the first one by using logarithms.

C. Parsing with $\lambda -$ PFA

Given a $\lambda -$ PFA $\mathcal{A} = \langle Q,\Sigma,\delta,I,F,P\rangle$ and a string $x\in\Sigma^\star$, we want to compute the probability $\Pr_{\mathcal{A}}(x)$ that $\mathcal{A}$ generates x.

We can introduce $\alpha'_x(i,q)$ in a similar way as for equation (3) as the probability of generating the prefix $x_1 \dots x_i$ and reaching state q:

$$\alpha'_x(i,q) = \sum_{(s_0,s_1,\dots,s_{i'})\in\Theta_{\mathcal{A}}(x_1\dots x_i)} I(s_0)\cdot\prod_{j=1}^{i'} P(s_{j-1},x'_j,s_j)\cdot 1(q,s_{i'}), \tag{8}$$

Here, $\Theta_{\mathcal{A}}$ denotes the set of sub-paths rather than full paths. On the other hand, $x'_{l'} = x_l$ or $x'_{l'} = \lambda$ with $1 \leq l \leq i \leq i'$, $1 \leq l' \leq i'$ and $x'_1 \dots x'_{i'} = x_1 \dots x_i$. In this case the computation of $\alpha'_x(i,q)$ can be performed from $\alpha'_x(i-1,q')$ through a new function $\alpha^\lambda_x(i,j,q)$, that represents the probability of generating (maybe with $\lambda-$transitions) the prefix $x_1 \dots x_i$ of x

and then to use j $\lambda-$transitions to reach q (that is, the last j transitions are $\lambda-$transitions). This function can be defined as:

$$\alpha^{\lambda}_x(i,0,q) \;=\; \begin{cases} I(q) & \text{if } i=0 \\ \sum_{q'} \alpha'_x(i-1,q') \cdot P(q',x_i,q) & \text{if } i \neq 0 \ , \end{cases} \tag{9}$$

$$\alpha^{\lambda}_x(i,j,q) \;=\; \sum_{q'} \alpha^{\lambda}_x(i,j-1,q') \cdot P(q',\lambda,q) \ \text{ if } i \geq 0 \text{ and } j > 0 \ . \tag{10}$$

By successive application of equation (10) ending by equation (9),

$$\alpha^{\lambda}_x(i,j,q) = \sum_{q'} \alpha^{\lambda}_x(i,0,q') \cdot T^{j}_{q',q} \ , \tag{11}$$

where $T^{j}_{q',q}$ is the (q',q) element in the j-th power of a matrix T. This matrix is defined as $T_{q',q} = P(q',\lambda,q)$ for all $q,q' \in Q$. Therefore, $T^{j}_{q',q}$ is the probability to reach q from q' using only j $\lambda-$transitions.

From equations (9)-(11) and taking into account the existence of all possible sequence of $\lambda-$transitions:

$$\alpha'_x(i,q) = \sum_{j=0}^{\infty} \alpha^{\lambda}_x(i,j,q). \tag{12}$$

Finally, by applying equations (11) and (9) in equation (12)

$$\alpha'_x(i,q) \;=\; \sum_{q''} \alpha'_x(i-1,q'') \cdot \sum_{q'} P(q'',x_i,q') \cdot \sum_{j=0}^{\infty} T^{j}_{q',q} \ .$$

By taking $T^0 = U$, the identity matrix, $U - T$ can be inverted in most cases and

$$\alpha'_x(i,q) \;=\; \sum_{q''} \alpha'_x(i-1,q'') \cdot \sum_{q'} P(q'',x_i,q') \cdot [U-T]^{-1}_{q',q} \ .$$

The probability of generating x by a $\lambda-$ PFA $\mathcal{A}$ is:

Proposition 7:

$$\mathrm{Pr}_{\mathcal{A}}(x) = \sum_{q \in Q} \alpha'_x(|x|,q) \cdot \ F(q). \tag{13}$$

Analogous results can be obtained for the backward and Viterbi algorithms.

The algorithms and propositions presented in the last subsections can also be derived from some results of the theory of discrete stochastic process [53].

D. The most probable string problem

In the previous problems a string is given and one wishes to compute its probability or that of a generating path. Other related interesting problems are the *most probable string* and the *most probable constrained string* in a PFA $\mathcal{A}$ [43]. The first problem consists in searching for the string with highest probability in $\mathcal{D}_{\mathcal{A}}$:

$$\operatorname*{argmax}_{x\in\Sigma^\star} \Pr_{\mathcal{A}}(x). \tag{14}$$

The second problem is the search for a string of upper bounded length with highest probability in $\mathcal{D}_{\mathcal{A}}$:

$$\operatorname*{argmax}_{x\in\Sigma^{\leq n}} \Pr_{\mathcal{A}}(x). \tag{15}$$

When I, F and P are defined over $\mathbb{Q}^+$, the following theorem holds.

Proposition 8: The computation of the most probable string and the computation of the most probable constrained string in a PFA are **NP**-Hard problems.

The formal proof of proposition 8 can be found in [43]. It should be noted that the problem is at least **NP**-complete but that the membership to **NP** is an open question. A similar question is proved to be undecidable for PFA *acceptors* which are different to the PFA covered in this paper [54].

However the problem of searching for the string associated to the most probable derivation in a PFA, that is, given a PFA $\mathcal{A}$, compute

$$\operatorname*{argmax}_{x\in\Sigma^\star} \widetilde{\Pr}_{\mathcal{A}}(x). \tag{16}$$

is polynomial [43].

IV. PROPERTIES

We now turn to study the properties of these models. What are the normal forms? Are they equivalent one to the other? Which are more expressive?. . . . These questions may be standard in formal language theory [55], but can lead to unexpected results in the probabilistic case.

A. A minimal form for DPFA

A central question that arises when considering any finite devices is that of being able to decide the equivalence between two such devices. This is important for learning, as it is known [56] that a class is hard to learn if the equivalence problem is not tractable. In terms of probabilistic automata the question one wishes to answer is: "Given two PFA (resp. two DPFA), are they equivalent?".

In the case of probabilistic objects a more natural question may be: Given two PFA (respectively two DPFA), and $\epsilon > 0$, are they ϵ-equivalent, *i.e.* is the distance between their distributions at most ϵ?

While the second question requires a concept of distance, and then will be discussed in section V, part of the first question can be answered here.

A Nerode theorem for DPFA*:* For any DPFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F, P\rangle$ the following equivalence relation over $\Sigma^\star$ has trivially finite index [57]:

$$x \equiv y \iff \forall u \in \Sigma^\star : \frac{\mathrm{Pr}'_{\mathcal{A}}(xu)}{\mathrm{Pr}'_{\mathcal{A}}(x)} = \frac{\mathrm{Pr}'_{\mathcal{A}}(yu)}{\mathrm{Pr}'_{\mathcal{A}}(y)} \ , \tag{17}$$

where $\mathrm{Pr}'_{\mathcal{A}}(z)$ is the probability of the unique path of states $(s_0, s_1, \ldots, s_{|z|})$ for z from the initial state $q_0 = s_0$:

$$\mathrm{Pr}'_{\mathcal{A}}(z) = \prod_{j=1}^{|z|} P(s_{j-1}, z_j, s_j). \tag{18}$$

The construction of a minimal DPFA can be found in [57]. This construction is based on the definition of an equivalence relation between strings on one hand and between states on another. Extending the Nerode relation over the states of the automaton goes as follows:

$$q \equiv q' \iff \begin{cases} \forall x : (s_0, x_1, \ldots, x_k, s_k) \text{ is a path with } x = x_1 \ldots x_k,\ s_0 = q_0 \text{ and } s_k = q \\ \forall x' : (s'_0, x'_1, \ldots, x'_{k'}, s'_{k'}) \text{ is a path with } x' = x'_1 \ldots x'_{k'},\ s'_0 = q_0 \text{ and } s'_{k'} = q', \\ \end{cases}$$

This relation has finite index, and from it the minimal canonical DPFA can be constructed by merging equivalent states, unique up to a state-isomorphism. This can be done in polynomial time. In [8] an efficient algorithm that does this is given. Also cases where even non-deterministic PFA can be put into canonical form (for instance if they are acyclic) are studied.

This enables us therefore to test the equivalence between two DPFA: minimize each and compare. If the corresponding minimal DPFA are isomorphic (a simple relabeling of the states through their minimum prefixes is enough to test this) then the initial DPFA are equivalent.

In the non-deterministic case Tzeng [45] proposes an algorithm that directly tests if two PFA are equivalent, but no result concerning a minimal PFA is known.

B. *Equivalence of* PFA *and* DPFA

One expects to find standard automata results when dealing with regular stochastic languages. For instance, that determinism does not imply a loss of expressive power. We prove here that this is not true. The result is mostly known and sometimes proved elsewhere (for instance in [8], [39]) but the construction of the counter-example is of use: it informs us that the mean of two deterministic regular distributions may not be regular deterministic.

We first define the mean of deterministic regular distributions and argue that this distribution is not deterministic.

Definition 10 (Mean of two distributions): Given two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over $\Sigma^\star$, we denote $\mathcal{D}_1 \Cup \mathcal{D}_2$ the distribution $\mathcal{D}$ such that:

$$\forall x \in \Sigma^\star, \Pr_{\mathcal{D}}(x) = 0.5 \cdot \Pr_{\mathcal{D}_1}(x) + 0.5 \cdot \Pr_{\mathcal{D}_2}(x). \tag{19}$$

Proposition 9: Given two regular deterministic distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, $\mathcal{D}_1 \Cup \mathcal{D}_2$ may not be regular deterministic.

The proof of this proposition is in the appendix A.

Proposition 10: There exist distributions that can be generated by PFA but not by DPFA.

The proof is a simple consequence of proposition 9: Take PFA from figure 2 as counter-example (see appendix A).

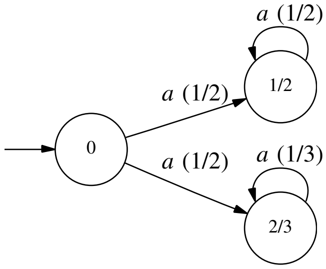

Fig. 2. A counter-example about distributions that can be generated by PFA but not by DPFA.

C. *Equivalence of* λ-PFA *and* PFA

Given a λ-PFA, there is an equivalent PFA with no λ-transitions [58]:

Proposition 11: Given a λ-PFA $\mathcal{A}$, representing distribution $\mathcal{D}_A$, there exists a PFA $\mathcal{B}$ with just one initial state such that $\mathcal{D}_\mathcal{A} = \mathcal{D}_\mathcal{B}$. Moreover $\mathcal{B}$ is of size at most $n \cdot \text{size}(\mathcal{A})$ with at most n states, where n is the number of states of $\mathcal{A}$. Also, $\mathcal{B}$ can be constructed from $\mathcal{A}$ in polynomial time.

We illustrate this proposition in figure 3. The first λ-PFA has been transformed into the second one that does not contain λ-transitions.

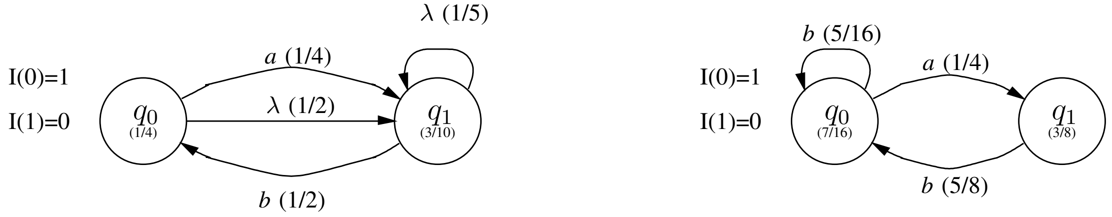

Fig. 3. A λ-PFA and its equivalent PFA

V. COMBINING DISTRIBUTIONS: AUTOMATA PRODUCT

There can be many simple ways of combining non-deterministic PFA. But because of the special interest DPFA represent, it would be of singular use to have some means of modifying deterministic regular distributions, of combining them. We give two results in this section, one relating to the product of two automata (the *co-emission probability*), and the second to the computation of the weight of a language inside a distribution.

From proposition 9, we know that the mean of two regular deterministic distributions may not be regular deterministic. Thus combining two DPFA has to be done in a different way. We can compute the product automaton as follows: Let $\mathcal{A}_1 = \langle Q_1, \Sigma, \delta_1, q_1^0, P_1, F_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma, \delta_2, q_2^0, P_2, F_2 \rangle$ be two DPFA.

Consider the automaton $\mathcal{A} = \langle Q_1 \times Q_2, \langle q_1^0, q_2^0 \rangle, \Sigma, \delta, F, P \rangle$ where

$$\delta = \{(\langle q_1, q_2 \rangle, a, \langle q_1', q_2' \rangle) : (q_1, a, q_1') \in \delta_1 \text{ and } (q_2, a, q_2') \in \delta_2\} \ ,$$

$$F(\langle q, q' \rangle) = F_1(q) \cdot F_2(q) \ ,$$

$$P(\langle q_1, q_2 \rangle, a, \langle q_1', q_2' \rangle) = P_1(q_1, a, q_1') \cdot P_2(q_2, a, q_2') \ .$$

This automaton affects to each string x the following *score*: $\Pr_{\mathcal{A}_1}(x) \cdot \Pr_{\mathcal{A}_2}(x)$. This product is called the *co-emission probability* of x by $\mathcal{A}_1$ and $\mathcal{A}_2$ [16]. The score corresponds to the probability of generating simultaneously x by $\mathcal{A}_1$ and $\mathcal{A}_2$.

The sum over $\Sigma^\star$ of these scores defines the co-emission (denoted $\mathcal{C}$) between $\mathcal{A}_1$ and $\mathcal{A}_2$. This quantity is of use when computing the distance between two distributions, but is also of interest as it measures the interactions between two distributions. In [16], it is proved that this is computable for APFA. Intractability results for more complicated architectures are proved in [17].

Formally:

$$\mathcal{C}(\mathcal{A}_1, \mathcal{A}_2) = \sum_{x \in \Sigma^\star} \Pr_{\mathcal{A}_1}(x) \cdot \Pr_{\mathcal{A}_2}(x). \tag{20}$$

We introduce one variable X_i per state in the product automata, with intended meaning:

$$X_i = \sum_{u:(q_0,u,q_i) \in \Theta_{\mathcal{A}}} \Pr(u\Sigma^\star).$$

Computing $\mathcal{C}(\mathcal{A}_1, \mathcal{A}_2)$ can be done through solving the following system:

$$X_i = \sum_{q_j \in Q, a \in \Sigma : (q_j,a,q_i) \in \delta} X_j \cdot P(q_j, a, q_i),$$

and when $i = 0$

$$X_0 = 1 + \sum_{q_j \in Q, a \in \Sigma : (q_j,a,q_i) \in \delta} X_j \cdot P(q_j, a, q_i).$$

Solving this system of equations enables us to solve equation (20) by

$$\mathcal{C}(\mathcal{A}_1, \mathcal{A}_2) = \sum_{q_i \in Q} X_i \cdot F(q_i).$$

The same sort of techniques allows us to consider the product automaton obtained by taking a deterministic finite-state automaton[4] $\mathcal{A}_1$ and a DPFA $\mathcal{A}_2$: Let $\mathcal{A}_1 = \langle Q_1, \Sigma, \delta_1, q_1^0, F_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma, \delta_2, q_2^0, F_2, P_2 \rangle$.

Consider the automaton $\mathcal{A} = \langle Q_1 \times Q_2, \Sigma, \delta_1, \langle q_1^0, q_2^0 \rangle, F, P \rangle$ where

$$\begin{aligned}
&\delta = \{ (\langle q_1, q_2 \rangle, a, \langle q_1', q_2' \rangle) : (q_1, a, q_1') \in \delta_1, \quad (q_2, a, q_2') \in \delta_2 \} \ , \\
&F(\langle q, q' \rangle) = \begin{cases} F_2(q') & \text{if } q \in F_1 \\ 0 & \text{if } q \notin F_1 \ , \end{cases} \\
&P(\langle q_1, q_2 \rangle, a, \langle q_1', q_2' \rangle) = \ P_2(q_2, a, q_2') \text{ if } (q_1, a, q_1') \in \delta_1.
\end{aligned}$$

[4] We will not define these formally here, but recommend the reader refers to usual textbooks [55] if needed.

Note that the construction does not necessarily yield a consistent DPFA: at every state the sum of probabilities might be less than 1. The construction is nevertheless of interest and yields the following result:

Proposition 12: $\Pr_{\mathcal{A}_2}(L_{\mathcal{A}_1}) = \Pr_{\mathcal{A}}(\Sigma^\star)$.

The proof follows from the construction of automaton $\mathcal{A}$.

This enables us to give a direct method of computing the weight of a regular language for a regular distribution, with a complexity which is linear in the product of the sizes of the two automata. It should be noted that this problem has been solved for special cases of the language $L_{\mathcal{A}_2}$ by more efficient algorithms in [46].

VI. COMPARING DISTRIBUTIONS: SIMILARITY MEASURES

Defining similarity measures between distributions is the most natural way of comparing them. Even if the question of exact equivalence (discussed in section IV-A) is of interest, in practical cases we wish to know if the distributions are close or not. In tasks involving the learning of PFA or DPFA one wants to measure the quality of the result or of the learning process. When learning takes place from a training sample, measuring how far the learned automaton is from a (test) sample can also be done by comparing distributions as a sample can be encoded as a PPTA.

There are 2 families of distance measures. Those that are true distances, and those that measure a cross entropy. We study both types.

A. Mathematical distances

All the definitions hereafter are seen as definitions of distances between distributions over $\Sigma^\star$. In doing so they implicitly define distances between automata, but also between automata and samples, or even between samples.

The most general family of distances are referred to as the d_n distances or distances for the norm L_n:

$$d_n(\mathcal{D}, \mathcal{D}') = \left(\sum_{x \in \Sigma^\star} |\Pr_{\mathcal{D}}(x) - \Pr_{\mathcal{D}'}(x)|^n \right)^{\frac{1}{n}}.$$

For n=1 we get a natural distance also known as the d_1 distance [36] or distance for the norm L_1.

$$d_1(\mathcal{D},\mathcal{D}') = \sum_{x\in\Sigma^\star} \left|\Pr_{\mathcal{D}}(x) - \Pr_{\mathcal{D}'}(x)\right|.$$

In the special case where $n = 2$ we obtain

$$d_2(\mathcal{D},\mathcal{D}') = \sqrt{\sum_{x\in\Sigma^\star} (\Pr_{\mathcal{D}}(x) - \Pr_{\mathcal{D}'}(x))^2}.$$

The following distance is used in [34] (under the name d_* or distance for the L_∞ norm):

$$d_{\max}(\mathcal{D},\mathcal{D}') = \max_{x\in\Sigma^\star} \left|\Pr_{\mathcal{D}}(x) - \Pr_{\mathcal{D}'}(x)\right|.$$

When concerned with very small probabilities such as those that may arise when an infinite number of strings have non null probability, it may be more useful to use logarithms of probabilities. In this way two strings with very small probabilities may influence the distance because their relative probabilities are very different: suppose $\Pr_1(x) = 10^{-7}$ and $\Pr_2(x) = 10^{-9}$, then the effect for d_1 of this particular string will be of $99 \cdot 10^{-9}$ whereas for the logarithmic distance the difference will be the same as if probabilities had been 10^{-1} and 10^{-3}.

The *logarithmic distance* is defined as

$$d_{\log}(\mathcal{D},\mathcal{D}') = \max_{x\in\Sigma^\star} \left|\log\Pr_{\mathcal{D}}(x) - \log\Pr_{\mathcal{D}'}(x)\right|.$$

It should be noticed that the logarithmic distance is infinite when the probability of a string is null in one distribution and strictly positive in the other one.

B. *Entropy based measures*

Similar to the $\log$ distance is the well-known Kullback-Leibler divergence:

$$d_{KL}(\mathcal{D},\mathcal{D}') = \sum_{x\in\Sigma^\star} \Pr_{\mathcal{D}}(x) \cdot \log\frac{\Pr_{\mathcal{D}}(x)}{\Pr_{\mathcal{D}'}(x)}.$$

We set in a standard way that $0\log 0 = 0$ and $\frac{0}{0} = 1$ and assume $\log$ to represent base 2 logarithms.

It should be noticed that in the case where some string has a null probability in $\mathcal{D}'$, but not in $\mathcal{D}$, then the Kullback-Leibler divergence becomes infinite.

Rewriting the Kullback-Leibler divergence as

$$d_{KL}(\mathcal{D},\mathcal{D}') = \sum_{x\in\Sigma^\star} (\Pr_{\mathcal{D}}(x)\cdot\log\Pr_{\mathcal{D}}(x) - \Pr_{\mathcal{D}}(x)\cdot\log\Pr_{\mathcal{D}'}(x)),$$

one can note the first term is the entropy of $\mathcal{D}$ and does not depend on $\mathcal{D}'$ and the second term is the cross entropy of $\mathcal{D}$ and $\mathcal{D}'$. From the information theory interpretation [59] the first term measures the optimal number of bits needed to encode $\mathcal{D}$ and the second one measures the cost (in number of bits of encoding) one must pay when estimating $\mathcal{D}$ using $\mathcal{D}'$. To fix the ideas a divergence of 1 (*i.e.* $d_{KL}(\mathcal{D},\mathcal{D}') = 1$) will mean that the average optimal number of bits needed to code a message of $\Sigma^\star$ distributed according to $\mathcal{D}$ using $\mathcal{D}'$ will be one more than the optimal code obtained using $\mathcal{D}$.

Let us now consider the random variables X and X' from $\Sigma^\star$ to $[0,1]$ such that $X(x) = \Pr_{\mathcal{D}}(x)$ and $X'(x) = \Pr_{\mathcal{D}'}(x)$, the Kullback-Leibler divergence can be expressed as:

$$d_{KL}(\mathcal{D},\mathcal{D}') = E_{\mathcal{D}}\left[\log\frac{X}{X'}\right]. \tag{21}$$

From this writing, we can see that the Kullback-Leibler divergence has some of the logarithmic distance properties.

C. Some properties

- d_n, $d_{\max}$, $d_{\log}$ are distances, *i.e.*, they comply with the usual properties.
  $\forall\mathcal{D},\mathcal{D}',\ \forall X\in\{n,\max,\log\}$:
  1) $d_X(\mathcal{D},\mathcal{D}') = 0 \iff \mathcal{D}=\mathcal{D}'$;
  2) $d_X(\mathcal{D},\mathcal{D}') = d_X(\mathcal{D}',\mathcal{D})$;
  3) $d_X(\mathcal{D},\mathcal{D}') + d_X(\mathcal{D}',\mathcal{D}'') \geq d_X(\mathcal{D},\mathcal{D}'')$ (for $d_{\log}$, assume $\mathcal{D}$, $\mathcal{D}'$ and $\mathcal{D}''$ are null on the same subset of $\Sigma^\star$).
- Obviously $\forall\ \mathcal{D},\mathcal{D}' : d_{\max}(\mathcal{D},\mathcal{D}') \leq d_1(\mathcal{D},\mathcal{D}')$.
- d_{KL} is not a mathematical distance. It nevertheless verifies the following properties $\forall\mathcal{D},\mathcal{D}'$:
  1) $d_{KL}(\mathcal{D},\mathcal{D}') \geq 0$
  2) $d_{KL}(\mathcal{D},\mathcal{D}') = 0 \iff \mathcal{D}=\mathcal{D}'$
  3) $d_{KL}(\mathcal{D},\mathcal{D}') \geq \frac{1}{2ln2}(d_1(\mathcal{D},\mathcal{D}'))^2$

D. Computing distances

We consider the following problem: given $\mathcal{D}$ and $\mathcal{D}'$, compute the distance $d_X(\mathcal{D}, \mathcal{D}')$ between them.

Main positive results include:

Proposition 13: If $\mathcal{D}$ and $\mathcal{D}'$ are given by DPFA, the computation of $d_2(\mathcal{D}, \mathcal{D}')$ can be done in polynomial time.

The proof of this proposition is reported in the appendix B.

Proposition 14 ([44]): If $\mathcal{D}$ and $\mathcal{D}'$ are given by DPFA, the computation of $d_{KL}(\mathcal{D}, \mathcal{D}')$ can be done in polynomial time.

E. Estimating distances

In some places, it is interesting either to compare a theoretical distribution with the empirical one, or to compare different distributions with respect to an empirical one. For the first purpose, we can use the following lemma:

Lemma 1 ([34], lemma 14): Let $\mathcal{D}$ be any distribution on $\Sigma^\star$, and S a sample of size $|S|$, then for $a > 1$,

$$\Pr\left(d_{\max}(\mathcal{D}, S) \leq \sqrt{6a(\log|S|)/|S|}\right) \geq 1 - 4\ |S|^{-a}.$$

In case one wants to learn —or estimate— distributions, this result is commonly used to compare the different learning algorithms: a sample of the target distribution is built and a distance between the learned distribution and the sample is computed.

In applications such as language modeling [60] or statistical clustering [61], [62], a distance based on the Kullback-Leibler divergence is commonly used to compare estimators. Let $\mathcal{D}$ be the target distribution and $\mathcal{A}$ a model. As previously noted this distance can be decomposed as the *entropy* of $\mathcal{D}$ and the *cross-entropy* of $\mathcal{D}_\mathcal{A}$ with respect to $\mathcal{D}$, $\mathcal{X}(\mathcal{D}, \mathcal{D}_\mathcal{A})$:

$$\mathcal{X}(\mathcal{D}, \mathcal{D}_\mathcal{A}) \;=\; -\sum_{x \in \Sigma^\star} \Pr_\mathcal{D}(x) \cdot \log \Pr_\mathcal{A}(x)$$

Since $\mathcal{D}$ is generally unknown, it is replaced by an adequate empirical estimate $\mathcal{D}_S$, based on a sample S. Let S' denote the set which contains the unique elements of the sample S (removing the repetitions). The corresponding empirical cross-entropy can then be written as:

$$\widehat{\mathcal{X}}(S, \mathcal{D}_\mathcal{A}) \;=\; -\sum_{x \in S'} \Pr_{\mathcal{D}_S}(x) \cdot \log \Pr_\mathcal{A}(x) \;=\; -\sum_{x \in S'} \frac{f(x)}{|S|} \cdot \log \Pr_\mathcal{A}(x)$$

where $f(x)$ is the number of occurrences of x in S. Finally, using "$x \in S$" in *multi-set* notation, we have:

$$\hat{\mathcal{X}}(S, \mathcal{D}_{\mathcal{A}}) \;=\; -\frac{1}{|S|} \sum_{x \in S} \log \Pr_{\mathcal{A}}(x). \tag{22}$$

Another measure often used in the language model community is the *perplexity* of S for a given model $\mathcal{A}$. It is computed using equation (22) as:

$$PP(S|\mathcal{A}) \;=\; 2^{\hat{\mathcal{X}}(S,\mathcal{D}_{\mathcal{A}})} \;. \tag{23}$$

which can be also written as:

$$PP(S|\mathcal{A}) \;=\; \left[\prod_{x \in S} P_{\mathcal{A}}(x)\right]^{-\frac{1}{|S|}} \;. \tag{24}$$

In practice, rather than the entropy (or perplexity) *per string* given by the previous equations, the entropy (or the perplexity) *per symbol* is often preferred [3]. It can be obtained approximately by replacing $|S|$ with $||S||$ in equation (22) (or equation (24)).

The properties of the perplexity can be summarized as follows:

- Equation (22) says that the cross-entropy measures the average number of *bits* one must pay by using the model $\mathcal{A}$ instead of $\mathcal{D}$ while coding the sample S.
- From equation (23), the perplexity measures the corresponding average number of *choices* entailed by this coding.
- From equation (24), the perplexity can be seen as the inverse of the geometric mean of the probabilities of the sample strings according to the model.

On the other hand, in practical work, the following properties must be carefully taken into account:

- Perplexity and entropy diverge as soon as one of the probabilities according to $\mathcal{A}$ is zero. In practice, this implies that the perplexity can only be used if $\mathcal{A}$ is *smoothed*; *i.e.*, it provides a non null probability for every string of $\Sigma^\star$.
- Obviously, perplexity and entropy only make sense if (the smoothed version of) $\mathcal{A}$ is really a probabilistic model; *i.e.*, $\sum_{x \in \Sigma^\star} \Pr_{\mathcal{A}}(x) = 1$.
- The perplexity can compare models only using the same sample S.

VII. Conclusion

We have provided in this first part a number of results centered on the probabilistic automata and distributions themselves. It remains to study the relationships between these models and other important models that can be found in the literature. Also the important task of approximating, learning or identifying these models, all central problems to structural pattern recognition, need to be explored. All this will be done in part II of this paper [40].

Acknowledgments

The authors wish to thank the anonymous reviewers for their careful reading and in-depth criticisms and suggestions.

Appendix

A. *Proof of the Proposition 9*

Proposition 9: *Given two regular deterministic distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, $\mathcal{D}_1 \Cup \mathcal{D}_2$ may not be regular deterministic.*

Proof: Consider distribution $\mathcal{D}_1$ defined by the following DPFA

$$\begin{aligned} (q_0, a, q_1) &\in \delta_1, & P_1(q_0, a, q_1) &= 1, & F_1(q_0) &= 0 \\ (q_1, a, q_1) &\in \delta_1, & P_1(q_1, a, q_1) &= \tfrac{1}{2}, & F_1(q_1) &= \tfrac{1}{2}\ , \end{aligned}$$

and distribution $\mathcal{D}_2$ defined by

$$\begin{aligned} (q'_0, a, q'_1) &\in \delta_2, & P_2(q'_0, a, q'_1) &= 1, & F_2(q'_0) &= 0 \\ (q'_1, a, q'_1) &\in \delta_2, & P_2(q'_1, a, q'_1) &= \tfrac{1}{3}, & F_2(q'_1) &= \tfrac{2}{3}\ . \end{aligned}$$

But no DPFA can implement $\mathcal{D}_1 \Cup \mathcal{D}_2$. Suppose such an automaton exists and call it $\mathcal{A}$ with $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F, P \rangle$; There would (because of determinism) have to be some n for which:

$$(q_i, a, q_{i+1}) \in \delta, \text{ for all } 0 \leq i < n$$

$$(q_n, a, q_k) \in \delta, \text{ with } 0 \leq k \leq n.$$

Denote $l = n - k + 1$ (the length of the cycle). The automaton (see figure 4) consists of a string of k states (with eventually $k = 0$) followed by a cycle of l states (with $l > 0$, as there is an infinity of strings with non null probability).

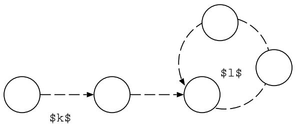

Fig. 4. The general shape of a DPFA over a one letter alphabet

Let $h : k \leq h \leq n \wedge \Pr_{\mathcal{A}}(a^h) \neq 0$ (such an h exists because of consistency).

$$\Pr(a^h) = p_h \cdot f_h\ ,$$

where

$$\begin{aligned} p_h &= \prod_{i=0}^{h-1} P_{\mathcal{A}}(q_i, a, q_{i+1}) \text{ and } f_h = F(q_h) \\ &= \frac{1}{2}(\frac{1}{2^h}) + \frac{1}{2}(\frac{2}{3^h}) = \frac{1}{2^{h+1}} + \frac{1}{3^h}\text{and } \Pr(a^{h+l}) = p_h \cdot p_{loop} \cdot f_h \ , \\ \text{where} \quad & p_{loop} = P(q_n, a, q_k) \cdot \prod_{i=k}^{n-1} P(q_i, a, q_{i+1}) \\ \text{and} \quad & \Pr(a^{h+2l}) = p_h \cdot p_{loop} \cdot p_{loop} \cdot f_h \ . \\ \text{It follows that:} \quad & p_{loop} = \frac{\Pr_{\mathcal{A}}(a^{h+l})}{\Pr_{\mathcal{A}}(a^h)} = \frac{\Pr_{\mathcal{A}}(a^{h+2l})}{\Pr_{\mathcal{A}}(a^h + l)} \ . \end{aligned}$$

Thus

$$\frac{\frac{1}{2^{h+l+1}} + \frac{1}{3^{h+l}}}{\frac{1}{2^{h+1}} + \frac{1}{3^h}} = \frac{\frac{1}{2^{h+2l+1}} + \frac{1}{3^{h+2l}}}{\frac{1}{2^{h+l+1}} + \frac{1}{3^{h+l}}} \ .$$

Simplifying:

$$\begin{aligned} (\tfrac{1}{2^{h+l+1}} + \tfrac{1}{3^{h+l}})^2 &= (\frac{1}{2^{h+2l+1}} + \frac{1}{3^{h+2l}}) \cdot (\frac{1}{2^{h+1}} + \frac{1}{3^h}) \\ \Rightarrow \quad 2 \cdot \tfrac{1}{2^{h+l+1}} \cdot \tfrac{1}{3^{h+l}} &= \frac{1}{2^{h+2l+1}} \cdot \frac{1}{3^h} + \frac{1}{2^{h+1}} \cdot \frac{1}{3^{h+2l}} \\ \Rightarrow \quad 2 \cdot \tfrac{1}{2^l} \cdot \tfrac{1}{3^l} &= \frac{1}{2^{2l}} + \frac{1}{3^{2l}} \\ \Rightarrow \quad 2^{l-1} \cdot 3^l &= 2^{2l} + 3^{2l} \ . \end{aligned}$$

If $l > 1$ the right hand side of the equation is odd, and the the left hand side is even, so we end up to a clear contradiction. And if $l = 1$ we solve and reach 3=13 which is also a contradiction. ■

B. Proof of the Proposition 13

Proposition 13: *If $\mathcal{D}$ and $\mathcal{D}'$ are given by* DPFA*, the computation of $d_2(\mathcal{D}, \mathcal{D}')$ can be done in polynomial time.*

Proof: In the following, $\mathcal{C}(A_1, A_2)$ matches the definition (20) in page 21.

$$
\begin{aligned}
&d_2(\mathcal{D},\mathcal{D}') = \left(\sum_{w\in\Sigma^\star} |\Pr_{\mathcal{D}}(w) - \Pr_{\mathcal{D}'}(w)|^2\right)^{\frac{1}{2}}\\
&= \left(\sum_{w\in\Sigma^\star} (\Pr_{\mathcal{D}}(w) - \Pr_{\mathcal{D}'}(w))^2\right)^{\frac{1}{2}}\\
&= \left(\sum_{w\in\Sigma^\star} \quad \Pr_{\mathcal{D}}(w)\cdot\Pr_{\mathcal{D}}(w) + \Pr_{\mathcal{D}'}(w)\cdot\Pr_{\mathcal{D}'}(w) - 2\ \Pr_{\mathcal{D}}(w)\cdot\Pr_{\mathcal{D}'}(w)\right)^{\frac{1}{2}}\\
&= \left(\sum_{w\in\Sigma^\star} \Pr_{\mathcal{D}}(w)^2 + \sum_{w\in\Sigma^\star} \Pr_{\mathcal{D}'}(w)^2 - 2\ \sum_{w\in\Sigma^\star} \Pr_{\mathcal{D}}(w)\cdot\Pr_{\mathcal{D}'}(w)\right)^{\frac{1}{2}}\\
&= (\mathcal{C}(\mathcal{D},\mathcal{D}) + \mathcal{C}(\mathcal{D}',\mathcal{D}') - 2\ \mathcal{C}(\mathcal{D},\mathcal{D}'))^{\frac{1}{2}}\ .
\end{aligned}
$$

If $\mathcal{D}$ and $\mathcal{D}'$ are given by DPFA the above can be solved in polynomial time. ■

REFERENCES

[1] A. Paz, *Introduction to Probabilistic Automata.* Academic Press, New York, NY, 1971.

[2] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recoginition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, 1989.

[3] F. Jelinek, *Statistical Methods for Speech Recognition.* Cambridge, Massachusetts: The MIT Press, 1998.

[4] R. Carrasco and J. Oncina, "Learning stochastic regular grammars by means of a state merging method," ser. Lecture Notes in Computer Science, R. C. Carrasco and J. Oncina, Eds., no. 862. Berlin, Heidelberg: Springer Verlag, 1994, pp. 139–150.

[5] L. Saul and F. Pereira, "Aggregate and mixed-order Markov models for statistical language processing," in *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, C. Cardie and R. Weischedel, Eds. Somerset, New Jersey: Association for Computational Linguistics, 1997, pp. 81–89.

[6] H. Ney, S. Martin, and F. Wessel, *Corpus-Based Statiscal Methods in Speech and Language Processing.* S. Young and G. Bloothooft, Kluwer Academic Publishers, 1997, ch. Statistical Language Modeling Using Leaving-One-Out, pp. 174–207.

[7] D. Ron, Y. Singer, and N. Tishby, "Learning probabilistic automata with variable memory length," in *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*. New Brunswick, New Jersey: ACM Press, 1994, pp. 35–46.

[8] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 3, pp. 269–311, 1997.

[9] K. S. Fu, *Syntactic pattern recognition and applications.* Prentice Hall, 1982.

[10] L. Miclet, *Structural Methods in Pattern Recognition.* Springer-Verlag, 1987.

[11] S. Lucas, E. Vidal, A. Amari, S. Hanlon, and J. C. Amengual, "A comparison of syntactic and statistical techniques for off-line OCR," ser. Lecture Notes in Computer Science, R. C. Carrasco and J. Oncina, Eds., no. 862. Berlin, Heidelberg: Springer Verlag, 1994, pp. 168–179.

[12] D. Ron, Y. Singer, and N. Tishby, "On the learnability and usage of acyclic probabilistic finite automata," in *Proceedings of COLT 1995*, 1995, pp. 31–40.

[13] H. Ney, "Stochastic grammars and pattern recognition," in *Proceedings of the NATO Advanced Study Institute*, P. Laface and R. D. Mori, Eds. SpringerVerlag, 1992, pp. 313–344.

[14] N. Abe and H. Mamitsuka, “Predicting protein secondary structure using stochastic tree grammars,” *Machine Learning*, vol. 29, pp. 275–301, 1997.

[15] Y. Sakakibara, M. Brown, R. Hughley, I. Mian, K. Sjolander, R. Underwood, and D. Haussler, “Stochastic context-free grammars for tRNA modeling,” *Nuclear Acids Res.*, vol. 22, pp. 5112–5120, 1994.

[16] R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen, “Metrics and similarity measures for hidden Markov models,” in *Proceedings of ISMB’99*, 1999.

[17] R. B. Lyngsø and C. N. S. Pedersen, “Complexity of comparing hidden Markov models,” in *Proceedings of ISAAC ’01*, 2001.

[18] P. Cruz and E. Vidal, “Learning regular grammars to model musical style: Comparing different coding schemes,” ser. Lecture Notes in Computer Science, V. Honavar and G. Slutski, Eds., no. 1433. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 211–222.

[19] M. G. Thomason, “Regular stochastic syntax-directed translations,” Computer Science Department. University of Tennesse, Knoxville, Tech. Rep. CS-76-17, 1976.

[20] M. Mohri, F. Pereira, and M. Riley, “The design principles of a weighted finite-state transducer library,” *Theoretical Computer Science*, vol. 231, pp. 17–32, 2000.

[21] H. Alshawi, S. Bangalore, and S. Douglas, “Learning dependency translation models as collections of finite state head transducers,” *Computational Linguistics*, vol. 26, 2000.

[22] ——, “Head transducer model for speech translation and their automatic acquisition from bilingual data,” *Machine Translation*, 2000.

[23] J. C. Amengual, J. M. Benedí, F. Casacuberta, A. C. no, A. Castellanos, V. M. Jiménez, D. Llorens, A. Marzal, M. Pastor, F. Prat, E. Vidal, and J. M. Vilar, “The EUTRANS-I speech translation system,” *Machine Translation Journal*, vol. 15, no. 1-2, pp. 75–103, 2000.

[24] S. Bangalore and G. Riccardi, “Stochastic finite-state models for spoken language machine translation,” in *Proceedings of the Workshop on Embeded Machine Translation Systems, NAACL*, Seattle, USA, May 2000, pp. 52–59.

[25] ——, “A finite-state approach to machine translation,” in *Proceedings of the North American ACL2001*, Pittsburgh, USA, May 2001.

[26] F. Casacuberta, H. Ney, F. J. Och, E. Vidal, J. M. Vilar, S. Barrachina, I. García-Varea, D. Llorens, C. Martínez, S. Molau, F. Nevado, M. Pastor, D. Picó, A. Sanchis, and C. Tillmann, “Some approaches to statistical and finite-state speech-to-speech translation,” *Computer Speech and Language*, 2003.

[27] L. Bréhélin, O. Gascuel, and G. Caraux, “Hidden Markov models with patterns to learn boolean vector sequences and application to the built-in self-test for integrated circuits,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 9, pp. 997–1008, 2001.

[28] Y. Bengio, V.-P. Lauzon, and R. Ducharme, “Experiments on the application of IOHMMs to model financial returns series,” *IEEE Transaction on Neural Networks*, vol. 12, no. 1, pp. 113–123, 2001.

[29] K. S. Fu, *Syntactic Methods in Pattern Recognition.* New-York: Academic Press, 1974.

[30] J. J. Paradaens, “A general definition of stochastic automata,” *Computing*, vol. 13, pp. 93–105, 1974.

[31] K. S. Fu and T. L. Booth, “Grammatical inference: Introduction and survey. part I and II,” *IEEE Transactions on System Man and Cybernetics*, vol. 5, pp. 59–72 and 409–423, 1975.

[32] C. S. Wetherell, “Probabilistic languages : A review and some open questions,” *Computing Surveys*, vol. 12, no. 4, 1980.

[33] F. Casacuberta, "Some relations among stochastic finite state networks used in automatic speech recogntion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 691–695, 1990.

[34] D. Angluin, "Identifying languages from stochastic examples," Yale University, Tech. Rep. YALEU/DCS/RR-614, March 1988.

[35] M. Kearns and L. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," in *21st ACM Symposium on Theory of Computing*, 1989, pp. 433–444.

[36] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie, "On the learnability of discrete distributions," in *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, 1994, pp. 273–282.

[37] M. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*. MIT press, 1994.

[38] N. Abe and M. Warmuth, "On the computational complexity of approximating distributions by probabilistic automata," in *Proceedings of the Third Workshop on Computational Learning Theory*. Morgan Kaufmann, 1998, pp. 52–66.

[39] P. Dupont, F. Denis, and Y. Esposito, "Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms," *Pattern Recognition*, 2004, to appear.

[40] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite state automata – part II," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. Special Issue-Syntactic and Structural Pattern Recognition, 2004.

[41] M. O. Rabin, "Probabilistic automata," *Information and Control*, vol. 6, no. 3, pp. 230–245, 1963.

[42] G. D. Forney, "The Viterbi algorithm," in *IEEE Procedings*, vol. 3, 1973, pp. 268–278.

[43] F. Casacuberta and C. de la Higuera, "Computational complexity of problems on probabilistic grammars and transducers," ser. Lecture Notes in Computer Science, A. de Oliveira, Ed., vol. 1891. Berlin, Heidelberg: Springer-Verlag, 2000, pp. 15–24.

[44] R. C. Carrasco, "Accurate computation of the relative entropy between stochastic regular grammars," *RAIRO (Theoretical Informatics and Applications)*, vol. 31, no. 5, pp. 437–444, 1997.

[45] W.-G. Tzeng, "A polynomial-time algorithm for the equivalence of probabilistic automata," *SIAM J. Comput.*, vol. 21, no. 2, pp. 216–227, 1992.

[46] A. Fred, "Computation of substring probabilities in stochastic grammars," in *Grammatical Inference: Algorithms and Applications*, ser. Lecture Notes in Computer Science, A. de Oliveira, Ed. Berlin, Heidelberg: Springer-Verlag, 2000, vol. 1891, pp. 103–114.

[47] M. Young-Lai and F. W. Tompa, "Stochastic grammatical inference of text database structure," *Machine Learning*, vol. 40, no. 2, pp. 111–137, 2000.

[48] D. Ron and R. Rubinfeld, "Learning fallible deterministic finite automata," *Machine Learning*, vol. 18, pp. 149–185, 1995.

[49] C. Cook and A. Rosenfeld, "Some experiments in grammatical inference," *NATO ASI on Computer Orientation Learning Process*, pp. 157–171, 1974, bonas, France.

[50] K. Knill and S. Young, *Corpus-Based Statistical Methods in Speech and Language Processing. Eds. S. Young and G. Bloothoof*. Kluwer Academic Publishers, 1997, ch. Hidden Markov Models in Speech and Language Processing, pp. 27–68.

[51] N. Merhav and Y. Ephraim, "Hidden Markov modeling using a dominant state sequence with application to speech recognition," *Computer Speech and Language*, vol. 5, pp. 327–339, 1991.

[52] ——, “Maximum likelihood hidden Markov modeling using a dominant state sequence of states,” *IEEE Transaction on Signal Processing*, vol. 39, no. 9, pp. 2111–2115, 1991.

[53] R. G. Galleguer, *Discrete Stochastic Processes*. Kluwer Academic Publisher, 1996.

[54] V. C. V. D. Blondel, “Undecidable problems for probabilistic automata of fixed dimension,” *Theory of Computing Systems*, vol. 36, no. 3, pp. 231–245, 2003.

[55] M. H. Harrison, *Introduction to Formal Language Theory*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1978.

[56] C. de la Higuera, “Characteristic sets for polynomial grammatical inference,” *Machine Learning*, vol. 27, pp. 125–138, 1997.

[57] R. Carrasco and J. Oncina, “Learning deterministic regular grammars from stochastic samples in polynomial time,” *RAIRO (Theoretical Informatics and Applications)*, vol. 33, no. 1, pp. 1–20, 1999.

[58] C. de la Higuera, “Why ϵ-transitions are not necessary in probabilistic finite automata,” EURISE, University of Saint-Etienne, Tech. Rep. 0301, 2003.

[59] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley Interscience Publication, 1991.

[60] J. Goodman, “A bit of progress in language modeling,” Microsoft Research, Tech. Rep., 2001.

[61] R. Kneser and H. Ney, “Improved clustering techniques for class-based language modelling,” in *European Conference on Speech Communication and Technology*, Berlin, 1993, pp. 973–976.

[62] P. Brown, V. Della Pietra, P. de Souza, J. Lai, and R. Mercer, “Class-based N-gram models of natural language,” *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[63] R. C. Carrasco and J. Oncina, Eds., *Grammatical Inference and Applications, ICGI-94*, ser. Lecture Notes in Computer Science, no. 862. Berlin, Heidelberg: Springer Verlag, 1994.

[64] A. de Oliveira, Ed., *Grammatical Inference: Algorithms and Applications, ICGI ’00*, ser. Lecture Notes in Computer Science, vol. 1891. Berlin, Heidelberg: Springer-Verlag, 2000.