Towards Feasible PAC-Learning of Probabilistic Deterministic Finite Automata *

Jorge Castro and Ricard Gavaldà

Departament de Llenguatges i Sistemes Informàtics
LARCA Research Group
Universitat Politècnica de Catalunya, Barcelona
{castro|gavalda}@lsi.upc.edu

Abstract. We present an improvement of an algorithm due to Clark and Thollard (Journal of Machine Learning Research, 2004) for PAC-learning distributions generated by Probabilistic Deterministic Finite Automata (PDFA). Our algorithm is an attempt to keep the rigorous guarantees of the original one but use sample sizes that are not as astronomical as predicted by the theory. We prove that indeed our algorithm PAC-learns in a stronger sense than the Clark-Thollard. We also perform very preliminary experiments: We show that on a few small targets (8-10 states) it requires only hundreds of examples to identify the target. We also test the algorithm on a web logfile recording about a hundred thousand sessions from an ecommerce site, from which it is able to extract some nontrivial structure in the form of a PDFA with 30-50 states. An additional feature, in fact partly explaining the reduction in sample size, is that our algorithm does not need as input any information about the distinguishability of the target.

1 Introduction

1.1 Context

Probabilistic Finite-State Automata (PFA) are thoroughly studied objects, both because of its inherent theoretical interest and their applications. Probabilistic Deterministic Finite-State Automata (PDFA) are a robust and natural subclass of PFA: See [6] for a study of the relations among these models, as well as HMM and POMDP.

These devices generate distributions on strings, and learning to approximate them from a sample is one of the central associated problems. A good number of algorithms have been proposed to infer PDFA. Some of

* Research supported in part by the EU PASCAL2 Network of Excellence and by the Spanish Ministry of Education and Science under projects MOISES-TA and TRANGRAM.

them are only empirically evaluated while, for others, convergence in the limit to the target PDFA can be proven; see among others [1, 4, 2, 15, 11].

In the more demanding PAC model, some evidence that learning PDFA is hard was provided by Kearns *et al.* [10]. More precisely, it is shown in [10] that assuming that noisy parities are hard to PAC-learn, distributions generated by 2-letter PDFA cannot be PAC-learned in time polynomial in n, $1/\epsilon$, and $1/\delta$, where from now on n denotes an upper bound on the number of states in the target machine, and ϵ and δ are the usual accuracy and confidence parameters in the PAC framework.

On the other hand, Ron *et al.* [13] gave an algorithm to PAC learn acyclic PDFA if polynomiality is measured in an additional parameter, the *distinguishability* of the target states - which we will denote as μ from now on. This formalized the observation, present already e.g. in [2], that one of the reasons that made some PDFA hard to learn was the presence of states with very similar suffix distributions. Clark and Thollard [3] showed how to extend this result to cyclic PDFA if still another parameter, the expected length of the generated strings L is taken into account. Their work is the culmination of a line of research, in the sense of identifying a set of parameters that make polynomial-time learning possible. We will state their result precisely in Section 2.

A number of papers have since presented variations or extensions of Clark and Thollard's algorithm (for brevity, called the C-T algorithm from now on). The related paper [14] by the same authors presents a more algorithmic view of the same ideas, with emphasis on the structure identification part. Palmer and Goldberg [12] showed the analogous result for learning with respect to the variation (L_1) distance rather than the KL-distance as C-T. Guttman *et al* [9] showed that the class of μ_2-distinguishable PDFA is also learnable with respect to the KL-distance; C-T uses the easier μ_∞-distinguishability measure. See also the related results in Guttman's thesis [8]. Denis *et al* [5] gave a quite deep PAC-style result for the full class of PFA, although the parameters in which the algorithm is polynomial are not completely identified there. Gavaldà *et al.* [7] give another variation of C-T that adapts to the complexity of the target, in the sense that it may stop earlier than the worst-case bound if convergence is achieved.

While the Clark-Thollard result proves polynomial-time learnability of PDFA, the actual polynomial is huge for interesting parameter values. For example, it is in the order of 10^{24} for $|\Sigma| = 2$, $n = L = 6$, and $\epsilon = \delta = \mu = 0.1$. For values similar to these ones, the algorithm in [7] uses sample sizes in the order of 10^5 in their experiments. On the other

hand, these algorithms do not look that different from other state-merging algorithms in the literature, which often do pretty well in practice with much smaller samples.

We believe that it is an interesting question whether these huge numbers are unavoidable if PAC-like guarantees are required or whether one can design algorithms with guaranteed polynomiality in well-defined parameters that have use realistic sample sizes; or, let us say, about as realistic as those of the algorithms which have been validated empirically only. It is important to note that we discuss algorithms having no prior knowledge about the structure of the state space; otherwise, the problem is much simpler.

1.2 Our Results

Our initial intention in this work was to produce an algorithm that does not ask for a bound on the distinguishability μ of the target PDFA. This value is in practice very hard to guess, and basically only trial-and-error can be used. As the work progressed, we incorporated other optimizations, all of which can still be rigorously justified. Yet, our algorithm is as easy to describe, if not more, than the original C-T algorithm.

We show that our algorithm PAC-learns in the same sense as that the C-T algorithm. In fact it learns with respect to a more demanding notion of distinguishability than the L_∞-distance as C-T, which we call prefL_∞-distance. This proof is the core of the paper.

While all our improvements are technical rather than conceptual, their combination could lead to dramatic experimental speedups. We describe a few experiments with an implementation of our algorithm that, we admit, are still far from being "an experimental evaluation". We first use the example PFAs in [7], having 10 states each, for which the algorithm in [7] required about $4 \cdot 10^5$ examples to converge. Our algorithm identifies the structure of the PDFAs and achieves low error with about 200-500 examples, i.e., a reduction by a factor of 1000 w.r.t. [7]. An additional example taken from [2] produces similar results.

We perform an additional experiment on a large dataset: a weblog of a high-traffic Spanish online travel agent, recording about 120,000 user sessions. Each session can be modelled as a string over an alphabet of size about 90, and average length about 12. On this dataset, our algorithm is able to identify some nontrivial structure: it extracts PDFA with 30-50 states that are certainly quite different from trees. We are currently in touch with the company to assess whether the patterns embodied in the PDFA make sense to local experts.

To finish this section, let us remark an important difference of our algorithm with C-T, of a high-level rather than purely technical nature: The C-T algorithm receives a number of parameters of the target as input, computes the worst-case number of examples required for those parameters, and asks for the full sample of that size upfront. Rather, we place ourselves in the more common situation where we have a given, fixed sample and we have to extract as much information as possible from it. Our main theorem then says that the algorithm PAC-learns provided this sample is large enough with respect to the target's parameters (some of which, such as μ, are unknown; we are currently working on removing the need to have the other parameters as inputs).

2 Preliminaries

We essentially follow notation in [3]. A PDFA A is a tuple $\langle Q, \Sigma, \tau, \gamma, \xi, q_0 \rangle$ where Q is a finite set of states, Σ is the alphabet, $\tau : Q \times \Sigma \longrightarrow Q$ is the transition function, $\gamma : Q \times (\Sigma \cup \{\xi\}) \longrightarrow [0, 1]$ defines the probability of emitting each symbol from each state ($\gamma(q, \sigma) = 0$ when $\sigma \in \Sigma$ and $\tau(q, \sigma)$ is not defined), ξ is a special symbol not in Σ reserved to mark the end of a string, and $q_0 \in Q$ is the initial state. Transition function τ is extended to $Q \times \Sigma^\star$ in the usual way.

Given an observation string $x\xi = \sigma_0 \dots \sigma_k \xi$ emitted by a known PDFA A, the state at each step can be tracked by starting from the initial state q_0 and following the labelled transitions according to x until reaching last symbol ξ. Also, the probability of generating a given string $x\xi$ from state q can be calculated recursively as follows: if x is the empty word λ the probability is $\gamma(q, \xi)$, otherwise x is a string $\sigma_0\sigma_1 \dots \sigma_k$ with $k \geq 0$ and $\gamma(q, \sigma_0\sigma_1 \dots \sigma_k\xi) = \gamma(q, \sigma_0)\gamma(\tau(q, \sigma_0), \sigma_1 \dots \sigma_k\xi)$.

The probability of state q in PDFA A is defined as the sum of values $\gamma(q_0, x\xi)$, where x ranges over the set of strings in $\Sigma^\star$ that traverse q.

Assuming every state of A has non-zero probability of generating some string, one can define for each state q a probability distribution D_q^A on $\Sigma^\star$: For each x, probability $D_q^A(x)$ is $\gamma(q, x\xi)$. The one corresponding to the initial state $D_{q_0}^A$ is called the distribution defined by A, written D^A in short. When there is no ambiguity, we will omit superindex A.

Given a multiset S of strings from $\Sigma^\star$ we denote by $S(x)$ the multiplicity of x in S, write $|S| = \sum_{x \in \Sigma^\star} S(x)$ and for every $\sigma \in \Sigma$ define $S(\sigma) = \sum_{x \in \Sigma^\star} S(\sigma x)$. To resolve the ambiguity of this notation on strings of length 1, we will always use greek letters to mean elements of Σ, and latin letters for strings. We also denote by $S(\xi)$ the multiplicity

of the empty word, $S(\lambda)$. To each multiset S corresponds an empirical distribution $\hat{S}$ defined in the usual way, $\hat{S}(x) = S(x)/|S|$. Finally, prefixes(S) denotes the multiset of prefixes of strings in S.

We consider several measures of divergence between distributions. Let D_1 and D_2 be probability distributions on $\Sigma^\star$. The Kullback–Leibler divergence, KL for short, is defined as

$$\mathrm{KL}(D_1, D_2) = \sum_x D_1(x) \log \frac{D_1(x)}{D_2(x)}.$$

The L_∞ supremum distance is

$$L_\infty(D_1, D_2) = \max_{x \in \Sigma^\star} |D_1(x) - D_2(x)|.$$

Finally, we also use the supremum distance on prefixes (introduced here, as far as we know):

$$\mathrm{pref}L_\infty(D_1, D_2) = \max_{x \in \Sigma^\star} |D_1(x\Sigma^\star) - D_2(x\Sigma^\star)|.$$

Definition 1. *We say distributions D_1 and D_2 are* μ-distinguishable *when $\mu \leq \max\{L_\infty(D_1, D_2), \mathrm{pref}L_\infty(D_1, D_2)\}$. A PDFA A is μ-distinguishable when for each pair of states q_1 and q_2 their corresponding distributions D_{q_1} and D_{q_2} are μ-distinguishable.*

Observe that $\mathrm{pref}L_\infty(D_{q_1}, D_{q_2}) \geq \mu$ iff there is any $x \in \Sigma^\star$ such that $|\gamma(q_1, x) - \gamma(q_2, x)| \geq \mu$. By definition, our measure of distinguishability is never smaller than the usual L_∞-distinguishability in the literature [3, 12].

3 Description of the Algorithm

We show below a learning algorithm for PDFAs that has as input parameters the alphabet size $|\Sigma|$, an upper bound L on the expected length of strings emitted from any state of the target (alternatively, a bound on the expected length of strings from the initial state and a bound on the variance), an upper bound n on the number of states of the target, and the confidence (δ) and precision (ϵ) parameters. In contrast with the C-T algorithm, it does not need as input parameter the distinguishability μ of the target.

According to [3], a PAC learner for the class of PDFA can be easily obtained from a polynomial-time algorithm, so-called `Learner` from now on, satisfying the requirements listed below; we follow their notation.

1. `Learner` returns (with high probability) a graph G isomorphic to a subgraph of the target PDFA A. This means that there is a bijection Φ from a subset of states of A to all nodes of G such that 1) $\Phi(q_0) = v_0$ (where q_0, v_0 are the initial states of A and G, respectively) and 2) if $\tau_G(v, \sigma) = w$ then $\tau(\Phi^{-1}(v), \sigma) = \Phi^{-1}(w)$.
2. The states in A whose probability is greater than $\epsilon_2/(L+1)$, which we call *frequent states*, have a representative in G. That is Φ is defined on frequent states.
3. If q is a frequent state in A and $\sigma \in \Sigma$ is such that $\gamma(q, \sigma) > \epsilon_5$ (we say (q, σ) is a *frequent transition*) then $\tau_G(\Phi(q), \sigma)$ exists and it equals $\Phi(\tau(q, \sigma))$.
4. A multiset S_v is attached to every node v in the graph. If v represents a frequent target state q (i.e., $\Phi(q) = v$ where q is frequent), then for every $\sigma \in \Sigma \cup \{\xi\}$, it holds $|S_v(\sigma)/|S_v| - \gamma(q, \sigma)| < \epsilon_1$. A multiset holding this property is said to be ϵ_1-good.

Numbers ϵ_1, ϵ_2 and ϵ_5 above and auxiliar quantities ϵ_0 and δ_0 that we use later are defined as follows. Note that they do not depend on μ.

$$\epsilon_1 = \frac{\epsilon^2}{16(|\Sigma|+1)(L+1)^2}$$
$$\epsilon_2 = \frac{\epsilon}{4n(n+1)L(L+1)\log(4(L+1)(|\Sigma|+1)/\epsilon)}$$
$$\epsilon_5 = \frac{\epsilon}{4|\Sigma|(n+1)L(L+1)\log(4(L+1)(|\Sigma|+1)/\epsilon)}$$
$$\epsilon_0 = \frac{\epsilon_2\epsilon_5}{n|\Sigma|(L+1)}$$
$$\delta_0 = \frac{\delta}{n^2|\Sigma| + 3n|\Sigma| + n}$$

From a such graph G a PDFA hypothesis H can be easily built having a small KL divergence with respect to A. This is described in the paragraphs "Completing the Graph" and "Estimating Probabilities" in [3], page 480. Basically, it is enough to complete the graph when necessary by introducing a new node, the *ground node*, representing all the low frequency states and new transitions to the ground node. Finally, a smoothing scheme is performed in order to estimate the transition probabilities.

The proof that an algorithm `Learner` with these properties, plus this additional graph completion and probability estimation step, is a PAC-learner is essentially the contents of Sections 4.3, 4.4 and 5 in [3]. It does

not involve distinguishability at all, so we can apply it in our setting even if we have changed our measure of distinguishability.

Our learning algorithm takes as inputs the parameters listed above and a sample from the target machine containing N examples. `Learner` performs at most $n|\Sigma| + 1$ learning stages, each one making a pass over all training examples and guaranteed to add one transition to the graph G it is building.

At the beginning of each stage, `Learner` has a graph G that summarizes our current knowledge of the target A. Nodes and edges in G represent, respectively, states and transitions of the target A. We call *safe nodes* the nodes of G, as they are inductively guaranteed (with probability at least $1 - \delta_0$) as stand for distinct states of A, with transitions among them as in A. Safe nodes are denoted by strings in $\Sigma^\star$.

Attached to each safe node v there is a multiset S_v that keeps information about the distribution on the target state represented by v. The algorithm starts with a trivial graph G consisting of a single node $v_0 = \lambda$ representing the initial state q_0 of the target, whose attached multiset is formed by all the available examples.

When a new stage starts, the learner adds a candidate node $u = v_u\sigma$ for each (safe) node v_u of G and each $\sigma \in \Sigma$ such that $\tau_G(v_u, \sigma)$ is undefined. Candidate nodes gather information about transitions of A leading from states that have already a safe representative in G but not having yet an edge counterpart in G. Attached to each candidate node u there is also a multiset S_u, initially empty. The learner also keeps, for each candidate node u, a list L_u of safe nodes that have not been yet distinguished (proved different) from u. Initially L_u contains all nodes in G.

For each training example $x\xi = \sigma_0 \dots \sigma_{i-1}\sigma_i\sigma_{i+1} \dots \sigma_k\xi$ in the dataset, `Learner` traverses the graph matching each observation σ_i to a state until either (1) all observations in x have been exhausted or (2) a transition to a candidate node is reached. This occurs when all transitions up to σ_{i-1} are defined and lead to a safe node w, but there is no edge out of w labeled by σ_i. In this case, we add the suffix $\sigma_{i+1} \dots \sigma_k$ to the multiset S_u of candidate node $u = w\sigma_i$ and, before processing a new example, we consider all pairs (u, v) where safe node v belongs to L_u. For each such pair, we call function `Test_Distinct`(u, v) described in Figure 1. If the test returns "distinct", we assume that u and v reach distinct states in the target and we remove v from L_u.

A candidate node becomes *important* when $|S_u|$ exceeds $\epsilon_0 N/2$. Every time that there is an important candidate node $u = v_u\sigma$ whose associated

set L_u is empty, u is promoted to a new safe node (G gets a new node labelled with u), u is not anymore a candidate node, and an edge from v_u to u labeled by σ is added to G. The multiset S_u is attached to the new safe node, u is included in the list $L_{u'}$ of all remaining candidate nodes u', and the phase continues.

If all training examples are processed without the condition above occurring, the algorithm checks whether there are any important candidate nodes left. If none remains, `Learner` returns G and stops. Otherwise, it closes the phase as follows: It chooses the important candidate $u = v_u\sigma$ and the safe node $v \in L_u$ having smallest distinguishability on the empirical distributions (samples), and identifies them, by adding to G an edge from v_u to v labeled by σ.

Finally, the phase ends by erasing all the candidate nodes and another phase starts.

function `Test_Distinct`(u, v)
//u is a candidate node; v is safe
$m_u \leftarrow |S_u|$; $s_u \leftarrow |\operatorname{prefixes}(S_u)|$;
$m_v \leftarrow |S_v|$; $s_u \leftarrow |\operatorname{prefixes}(S_v)|$;
$t_{u,v} \leftarrow \left(\frac{2}{\min(m_u,m_v)} \ln \frac{4m_u^2(s_u+s_v)\pi^2}{3\delta_0}\right)^{1/2}$
$d \leftarrow \max\left(L_\infty(\hat{S}_u, \hat{S}_v), \operatorname{pref}L_\infty(\hat{S}_u, \hat{S}_v)\right)$
if $d > t_{u,v}$ **then return** "distinct"
else return "not clear"

Fig. 1. The state-distinctness test

4 Analysis

In this section we show that algorithm `Learner` satisfies conditions (1)-(4) before, and therefore can be turned into a PAC-learner for PDFA.

The following two lemmas describe the behavior of `Test_Distinct`. Here D_u (respectively, D_v) denotes the target distribution on state $q = \tau(q_0, u)$ ($q = \tau(q_0, v)$).

Lemma 2. *If $D_u = D_v$ function* `Test_Distinct`(u, v) *returns "not clear" with probability* $1 - 6\delta_0/(\pi^2 m_u^2)$.

Proof. Let $D = D_u (= D_v)$. Function `Test_Distinct` returns "different" when there exists a string x such that $|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| > t_{u,v}$ or $|\hat{S}_u(x) - \hat{S}_v(x)| > t_{u,v}$. First, we bound the probability of the event $\mathrm{pref}L_\infty(\hat{S}_u, \hat{S}_v) > t_{u,v}$. To avoid summing over infinitely many x, consider $S_u \cup S_v$ ordered, say lexicographically. Then the event above is equivalently to saying "there is an index i in this ordering such that some prefix of the ith string in $S_u \cup S_v$ in this ordering, call it x, satisfies the condition above". (This is another way of saying that only x's appearing in prefixes$(S_u \cup S_v)$ can make the inequality true, since all others have $\hat{S}_u(x\Sigma^\star) = \hat{S}_v(x\Sigma^\star) = 0$.) Therefore, its probability is bounded above by the maximum of $(s_u + s_v)\Pr[\,|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| > t_{u,v}]$ over all strings x. By the triangle inequality, this is at most

$$(s_u+s_v)\left(\Pr[\,|\hat{S}_u(x\Sigma^\star) - D(x\Sigma^\star)| > t_{u,v}/2] + \Pr[|\hat{S}_v(x) - D(x\Sigma^\star)| > t_{u,v}/2]\right).$$

Since $E[\hat{S}_u(x\Sigma^\star)] = E[\hat{S}_v(x\Sigma^\star)] = D(x\Sigma^\star)$, by Hoeffding's inequality this is at most

$$\begin{aligned}&(s_u + s_v)(2\,\exp(-2(t_{u,v}^2/4)\; m_u) + 2\,\exp(-2(t_{u,v}^2/4)\; m_u))\\ \le\; &4(s_u + s_v)\exp(-(t_{u,v}^2/2)\;\min(m_u, m_v)),\end{aligned}$$

which is $3\delta_0/(\pi^2 m_u^2)$ by definition of $t_{u,v}$.

A similar reasoning also shows that the probability of the event $L_\infty(\hat{S}_u, \hat{S}_v) > t_{u,v}$ is at most $3\delta_0/(\pi^2 m_u^2)$ and we are done.

Lemma 3. *If D_u and D_v are μ-distinguishable and $\min(m_u, m_v) \ge \frac{8}{\mu^2}\ln\frac{8(m_u+m_v)m_u^2 L\pi^2}{3\delta_0^2}$ then* `Test_Distinct`(u, v) *returns "different" with probability $1 - \delta_0$.*

Proof. We first bound the size of prefixes$(S_u \cup S_v)$. Clearly, its expected size is at most $L|S_u \cup S_v|$. Then, by Markov's inequality, $\Pr[|\,\text{prefixes}(S_u \cup S_v)| \ge \frac{2}{\delta_0}\cdot L|S_u \cup S_v|]$ is less than $\delta_0/2$. Therefore, we have with probability at least $1 - \delta_0/2$ that $s_u + s_v \le 2(m_u + m_v)L/\delta_0$.

Now assume there is a string x witnessing that $\mathrm{pref}L_\infty(D_u, D_v) > \mu$ (otherwise some x is a witness for $L_\infty(D_u, D_v) > \mu$ and we argue in a similar way), i.e. a string such that $|D_u(x\Sigma^\star) - D_v(x\Sigma^\star)| > \mu$. If $\min(m_u, m_v) \ge \frac{8}{\mu^2}\ln\frac{8(m_u+m_v)m_u^2 L\pi^2}{3\delta_0^2}$, by the argument above with high probability we have $t_{u,v} \le \mu/2$ and the probability of returning "different" is at least the probability of the event $|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| > \mu/2$. The hypothesis on x and the triangle inequality shows that probability of the complementary event $|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| \le \mu/2$ is at most

$\Pr[\,|\hat{S}_u(x\Sigma^\star) - D_u(x\Sigma^\star)| > \mu/4] + \Pr[\,|\hat{S}_v(x\Sigma^\star) - D_v(x\Sigma^\star)| > \mu/4]$. By the Hoeffding bound, this sum is less than $\delta_0/2$, and we are done.

Lemmas 4–8 below share the hypothesis the current G is isomorphic to a subgraph of A and deal with one fixed stage of the learning algorithm. Probabilities are taken over samples.

Lemma 4. *Let u be a candidate node. If u is promoted to safe (in this stage) then, with probability $1 - \delta_0$, node u corresponds to a new target state, i.e., one not represented in the current graph G.*

Proof. We show that a candidate node u representing the same target state than a safe state v has very small probability of being promoted. By Lemma 2 at any specific call to `Test_Distinct`(u, v), the function returns the wrong value "different" with probability at most $6\delta_0/(\pi^2 m_u^2)$. The test is called once for every example included in S_u, so the value of m_u increases by 1 at each consecutive call within the stage. Therefore, the probability that safe node v is ruled out from L_u is at most

$$\sum_{m_u \geq 1} 6\delta_0/(\pi^2\, m_u^2) = \delta_0$$

Lemma 5. *Let u be a candidate node, and let μ be the distinguishability of the target. If N is greater than $\frac{16}{\epsilon_0\mu^2}(3e \ln \frac{48}{\epsilon_0\mu^2} + \ln \frac{16L\pi^2}{3\delta_0^2})$ with probability $1 - n\delta_0$, if candidate node u is merged with a safe node v then, strings u and v end in the same state in the target.*

Proof. Assume candidate u is merged with safe node v. Necessarily u is important and $N \geq m_u > \epsilon_0 N/2$. As v is safe it also holds $m_v > \epsilon_0 N/2$. It is also clear that $m_u + m_v \leq 2N$. From these values of m_u, m_v and the hypothesis on N, it can be checked that $\min(m_u, m_v)$ satisfies requirement in Lemma 3. So, if they were representatives of different target states, safe v would remain in L_u with probability at most δ_0.

Lemma 6. *Assume that, after processing all examples, graph G has no safe node v representing a frequent state q of A. If $N > \frac{8(L+1)}{\epsilon_2} \ln \frac{1}{\delta_0}$ then, with probability $1 - \delta_0$, the learner will not finish yet.*

Lemma 7. *Assume that, after processing all examples, graph G has no edge representing a frequent transition (q, σ) in A. If $N > \frac{8(L+1)}{\epsilon_2\epsilon_5} \ln \frac{1}{\delta_0}$ then with probability $1 - \delta_0$, some candidate is important and the learner will not finish yet.*

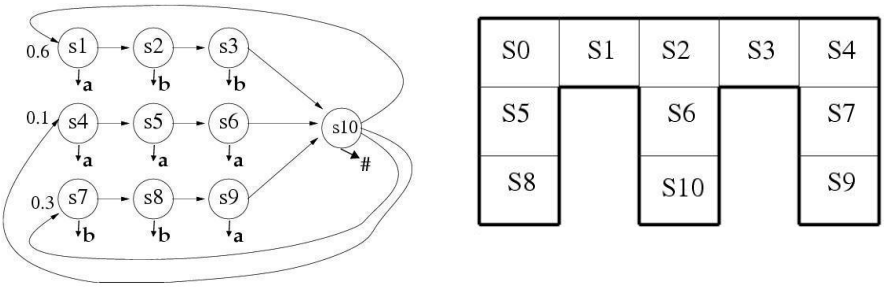

Fig. 2. Example PFAs from [7]

Lemma 8. *Let u be a candidate node. If the number N of training examples is greater than $\frac{1}{\epsilon_0 \epsilon_1^2} \ln \frac{2(|\Sigma|+1)}{\delta_0}$ and u is promoted to safe then, with probability $1-\delta_0$, multiset S_u is ϵ_1-good.*

Let N_0 be $\max\left(\frac{16}{\epsilon_0 \mu^2}(3e \ln \frac{48}{\epsilon_0 \mu^2} + \ln \frac{16 L \pi^2}{3\delta_0^2}), \frac{8}{\epsilon_0 \epsilon_1^2} \ln \frac{2(|\Sigma|+1)}{\delta_0}\right)$. A straightforward induction shows the main theorem:

Theorem 9. *If $N > N_0$, with probability $1-\delta$* `Learner` *returns a graph G satisfying requirements (1)–(4) listed above.*

As explained already, the second phase of the learning algorithm takes the graph G, completes it if necessary, and sets the transition probabilities according to their empirical distribution. An additional smoothing is performed, as described in [3]. We do not describe it here, but state that the resulting PDFA will approximate the target in the KL distance, as can be deduced from the proof in [3].

5 Experiments

5.1 Small Targets

Our first experiments used the two automata tested by Gavaldà *et al.* [7], shown in Figure 2. The one on the left is a (nondeterministic) HMM repeatedly generating strings in $\{abb, aaa, bba\}$ with different probabilities. The one on the right is the "cheese maze": at each state (or square), an observation (a letter in $\{1, 2, 3\}$) indicates the number of walls around that state, with the exception of $s10$ where the automaton terminates. They have thus 10 states each. We have additionally used the

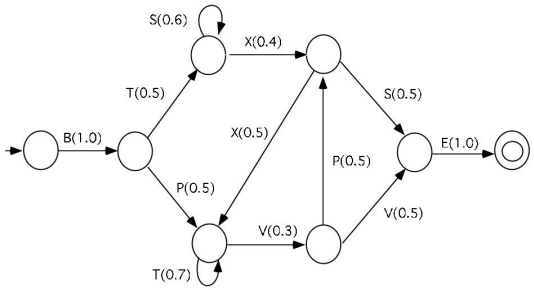

Fig. 3. The Reber grammar; plot taken from [2]

Reber grammar automaton, with 8 states, discussed in [2] and shown in Figure 3.

For each of these automata, and different values of N, we generated 10 examples of size N from the target, and run our algorithm on these examples. In all experiments we used $\delta = 0.05$ and the (known) number of target states for n. For the Reber grammar, the full structure of the automaton was identified about half the times with $N = 100$, but systematically identified when $N = 200$, at which point transition probabilities were correct within (absolute) 5%. For $N = 1000$, transition probabilities were correct within 1%. For the cheese maze automaton, at $N = 300$ the structure was found 9 out of 10 times, with transition probabilities correct up to 2%. For $N = 1000$, the structure was correctly found in all trials. Interestingly, when the program was changed to use only L_∞-, rather than prefL_∞-distinguishability, the point at which the structure is identified more than 50% of the times was around $N = 1300$. That is, using prefL_∞ did help in this case. Results were similar for the HMM on the left of Figure 2.

5.2 An Experiment with a Real, Large Dataset

As a larger test, we used a web logfile recording sessions from a high-traffic Spanish online travel agency, selling flights, hotel stays, car rentals, and theater tickets. Each entry in the logfile records a user request to the company's web to initiate some action. The local experts distinguish 91 types of requests or tags; some examples could be "search_flight", "search_hotel", "book_flight", "credit_card_info", "home", "register", "help", etc. We preprocessed the logfile transforming each request into a tag identifier and grouping clicks from the same user into

sessions. Therefore each session can be viewed as a string over a 91-letter alphabet. The median and average of session length are 4 and 11.9, excluding 1-click sessions, and we had 120,000 sessions to work with.

We ran our algorithm on subsets of several sizes N of this dataset. Since human web users cannot be perfectly modelled by PDFA, one should not expect high accuracy. Still, there are certainly patterns that users tend to follow, so it was worth checking whether any structure is found.

We tried $N = 40,000$ to $N = 100,000$ in multiples of $10,000$. Figure 4 presents, for each N, the size of the resulting PDFA and the L_1-distance from the dataset to a randomly generated sample of size $100,000$ from the output machine.

Sample	# states	L_1 distance
40k	35	.582
50k	36	.546
60k	39	.518
70k	42	.516
80k	45	.480
100k	54	.439

Fig. 4. Results on the online travel agency dataset

Note that the L_1 distance can have value up to 2. (In fact, we tried generating several independent samples of size 100k from the PDFA obtained with the 100k sample and computing their L_1 mutual distance. The results were around 0.39 even though they came from the same machine, so this value is really the baseline once sample size has been fixed to 100k.) The table shows that convergence to a fixed machine did not occur, which is no surprise. On the other hand, the resulting machines were not at all tree-like PDFAs that occur when no states can be merged: most safe states did absorb candidate states at some point. Given the alphabet size (91) and number of states (≥ 30), depicting and understanding them is not immediate.

Note that we have not discussed the values of ϵ and L used in the experiments. In fact, our implementation does not use them: they are used only to determine when a state is important. In particular, observe that ϵ and L are not used in the state distinctness test. The implementation keeps merging candidate states as long as there is any left at the end of the stage, hence every candidate state is eventually merged. We believe that it is possible to prove that this variant is still a PAC learner, since

non-important states, after smoothing, by definition do not contribute much to the KL distance. We believe it is also possible to remove the need for an upper bound on n without significantly increasing sample size in practice; this would give a PAC-learning whose only parameter is the confidence δ.

6 Conclusions

We believe that these first experiments, as preliminary as they are, show that maybe one cannot rule out the existence of a provably-PAC learner that has reasonable sample sizes in practice. More systematic experimentation, as well as improving of our slow, quick-and-dirty implementation, is work in progress.

References

[1] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *ICGI*, pages 139–152, 1994.

[2] R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *ITA*, 33(1):1–20, 1999.

[3] A. Clark and F. Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, 2004.

[4] C. de la Higuera, J. Oncina, and E. Vidal. Identification of DFA: data-dependent vs data-independent algorithms. In *ICGI*, pages 313–325, 1996.

[5] F. Denis, Y. Esposito, and A. Habrard. Learning rational stochastic languages. In *COLT*, pages 274–288, 2006.

[6] P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38:1349–1371, 2005.

[7] R. Gavaldà, P. W. Keller, J. Pineau, and D. Precup. PAC-learning of Markov models with hidden state. In *ECML*, pages 150–161, 2006.

[8] O. Guttman. *Probabilistic Automata Distributions over Sequences*. Ph.D. thesis, The Australian National University, September 2006.

[9] O. Guttman, S. V. N. Vishwanathan, and R. C. Williamson. Learnability of probabilistic automata via oracles. In *ALT*, pages 171–182, 2005.

[10] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *STOC*, pages 273–282, 1994.

[11] C. Kermorvant and P. Dupont. Stochastic grammatical inference with multinomial tests. In *ICGI*, pages 149–160, 2002.

[12] N. Palmer and P. W. Goldberg. PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. In *ALT*, pages 157–170, 2005.

[13] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *J. Comput. Syst. Sci.*, 56(2):133–152, 1998.

[14] F. Thollard and A. Clark. Learning stochastic deterministic regular languages. In *ICGI*, pages 248–259, 2004.

[15] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *ICML*, pages 975–982, 2000.