



Universiteit
Leiden
The Netherlands

Learning probabilistic automata using residuals

Chu, W.; Chen, S.; Bonsangue, M.M.; Cerone, A.; Olveczky, P.C.

Citation

Chu, W., Chen, S., & Bonsangue, M. M. (2021). Learning probabilistic automata using residuals. *Proceedings Of The 18Th International Colloquium Theoretical Aspects Of Computing - {Ictac} 2021 - , Virtual Event, Nur-Sultan, Kazakhstan, September 8-10, 2021*, (12819), 295-313. doi:10.1007/978-3-030-85315-0_17

Version: Publisher's Version

License: [Licensed under Article 25fa Copyright Act/Law \(Amendment Taverne\)](#)

Downloaded from: <https://hdl.handle.net/1887/3257031>

Note: To cite this publication please use the final published version (if applicable).



Learning Probabilistic Automata Using Residuals

Wenjing Chu¹✉, Shuo Chen², and Marcello Bonsangue¹ 

¹ Leiden Institute of Advanced Computer Science, Leiden University,
Leiden, The Netherlands

`chuw@liacs.leidenuniv.nl`

² Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
`s.chen3@uva.nl`

Abstract. A probabilistic automaton is a non-deterministic finite automaton with probabilities assigned to transitions and states that define a distribution on the set of all strings. In general, there are distributions generated by automata with a non-deterministic structure that cannot be generated by a deterministic one. There exist several methods in machine learning that can be used to approximate the probabilities of an automaton given its structure and a finite number of strings independently drawn with respect to an unknown distribution. In this paper, we efficiently construct a probabilistic automaton from a sample by first learning its non-deterministic structure using residual languages and then assigning appropriate probabilities to the transitions and states. We show that our method learns the structure of the automaton precisely for a class of probabilistic automata strictly including deterministic one and give some experimental results to compare the learned distribution with respect to other methods. To this end, we present a novel algorithm to compute the Euclidean distance between two weighted graphs effectively.

Keywords: Probabilistic automata · Residual finite state automata · Learning automata · L_2 distance between discrete distributions

1 Introduction

Probabilistic models like hidden Markov models and probabilistic finite automaton (PFA) are widely used in the field of machine learning, for example, in computational biology [2], speech recognition [1, 14, 15], and information extraction [20]. It has become increasingly clear that learning probabilistic models is essential to support these downstream tasks.

Passively learning a probabilistic automaton aims at constructing an approximation of a finite representation of an unknown distribution D through a finite number of strings independently drawn with respect to D . Many passive learning algorithms for probabilistic automata have been proposed. Still, most of them concentrate only on the restricted class of deterministic probabilistic finite

automata (DPFA). The most famous algorithm is ALERGIA [4] based on state merging and folding given a positive sample. ALERGIA has been extended to deal with deterministic probabilistic automata [8, 23], and at the limit, it characterizes the original distribution. However, because of the underlying determinism, the resulting automata are often very large (exponential on the size of the sample), so that it may easily become impractical.

In this paper, we propose a more efficient representation using non-determinism. We first learn from a finite sample the non-deterministic structure of the support of the distribution using residual languages and then add probabilities to the transitions solving non-determinism by a fair distribution of the probabilities. As such, the algorithm also approximates distributions generated by probabilistic automata that cannot be generated by deterministic ones [12].

There are not so many algorithms for learning general probabilistic automata. The most well known is the Baum-Welch algorithm [3] that constructs a fully connected graph on the estimated number of states needed and is therefore not very practical. Our work is based on the learning algorithm for residual automata introduced in [10]. The residual (also called derivative) of a language L with respect to a word u is the set of words v such that uv is in L . Residual automata are non-deterministic automata that can be used to learn efficiently any regular language. In the probabilistic setting, a learning algorithm using probabilistic residual distributions has been proposed in [13]. The starting point of their work is very similar to ours, but the resulting algorithm assumes, differently from ours, precise probabilities for each word in the sample.

To compare the goodness of our algorithm, we adapted the algorithm for computing the L_2 distance between two distributions presented in [18] in the context of weighted automata, i.e. automata transitions and states labeled with weights from a field (or more generally semirings) instead of probabilities. The novelty is in the computation of the shortest distance algorithm for weighted graphs using a weaker condition than the original one. This step was necessary in order to be able to apply it to classical probabilistic automata. The L_2 distance is used in few experiments to compare our algorithm with ALERGIA and with learning through k -testable languages [5]. The latter are language that can be accepted by an automaton that can see at most k many symbols. We also use other metrics in this comparison, such as accuracy, precision and sensitivity weighted with a confidence factor to recognize the probabilistic nature of the experiments.

2 Preliminaries

Let Σ be a finite alphabet and Σ^* be the set of all finite strings over Σ , with ε denoting the empty string. A language L is a subset of Σ^* . For any string u and any language L , we define $\text{Pref}(u) = \{v \in \Sigma^* \mid \exists w \in \Sigma^*, vw = u\}$ to be the set of prefixes of u and $\text{Pref}(L) = \bigcup_{u \in L} \text{Pref}(u)$ to be the prefix closure of L .

Definition 1. Non-deterministic finite automaton. A non-deterministic finite automaton (NFA) is a 5-tuple $A = \langle \Sigma, Q, I, F, \delta \rangle$, where

- Σ is a finite alphabet,
- Q is a finite set of states,
- $I : Q \rightarrow 2$ is characterizing the set of initial states,
- $F : Q \rightarrow 2$ is characterizing the set of final states,
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function.

The transition function δ can be naturally extended from symbol in Σ to arbitrary strings by defining the extended transition function $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ inductively as follows:

- For every $q \in Q$, $\delta^*(q, \epsilon) = q$,
- For every $q \in Q$, $x \in \Sigma^*$, and $a \in \Sigma$, $\delta^*(q, xa) = \bigcup \{\delta(p, a) \mid p \in \delta^*(q, x)\}$.

A string $x \in \Sigma^*$ is accepted by a NFA A from a state $q \in Q$ if $\delta^*(q, x) \cap F \neq \emptyset$. We denote by $L(A, q)$ the set of all those strings. The language $L(A)$ accepted by A is the set of all strings accepted by A from some $q_0 \in I$. A language L is called regular if there is a NFA A that accepts exactly the language L [17].

For a NFA A , an accepting path π for a string $x = a_1 \dots a_n$ is a sequence of states $q_0 \dots q_n$ such that $q_{i+1} \in \delta(q_i, a_{i+1})$ for all $0 \leq i \leq n-1$, starting from an initial state, i.e. $I(q_0) = 1$, and ending in a final state, i.e. $F(q_n) = 1$. We denote by $Paths(x)$ the set of all accepting paths for a given string x . Note that the set $Paths(x)$ is finite. An accepting path contains a cycle if there is a repeating state. That is, there exists different i and j such that $q_i = q_j$.

For any language L and for any string $u \in \Sigma^*$, the residual language of L associated with u is defined by the u -derivative $L_u = \{x \in \Sigma^* \mid ux \in L\}$, and we call u a characterizing word for L_u . A language $L' \subseteq \Sigma^*$ is a residual language of L if there exists a string $u \in \Sigma^*$ such that $L' = L_u$. The number of residual languages of a language L is finite if and only if L is regular [11]. This implies that there exists a finite set of strings $\mathcal{B}(L)$ such that $x \in \mathcal{B}(L)$ if L_x is a residual language of a regular language L . The set $\mathcal{B}(L)$ can be constructed depending on the representation of the language L . For example, if L is the language accepted by a trimmed NFA A (i.e., minimal and with all states reachable from an initial state), then $\mathcal{B}(L)$ can be constructed as a finite set of minimal length strings reaching all states of A from some initial state.

Definition 2. Residual finite state automaton [7]. A residual finite state automaton (RFSFA) is a NFA $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ such that, for each state $q \in Q$, $L(A, q)$ is a residual language of $L(A)$.

In other words, a RFSFA A is a non-deterministic automaton whose states correspond exactly to the residual languages of the language recognized by A .

Non-deterministic automata can be generalized to frequency and probabilistic automata. Frequency finite automata associate a positive rational number to each transition, initial states and final ones representing the ‘number of occurrences’ of a transition or state.

Definition 3. Frequency finite automaton. A frequency finite automaton (FFA) is a 5-tuple $A = \langle \Sigma, Q, I_f, F_f, \delta_f \rangle$, where:

- Σ is a finite alphabet,
- Q is a finite set of states,
- $I_f : Q \rightarrow \mathbb{Q}^+$,
- $F_f : Q \rightarrow \mathbb{Q}^+$,
- $\delta_f : Q \times \Sigma \rightarrow \mathbb{Q}^{+Q}$

such that for every state $q \in Q$ the weight of the incoming transitions is equal to the weight of the outgoing transitions:

$$I_f(q) + \sum_{q' \in Q, a \in \Sigma} \delta_f(q', a)(q) = F_f(q) + \sum_{q' \in Q, a \in \Sigma} \delta_f(q, a)(q').$$

Intuitively, the above condition says that frequency is preserved by passing through states. Note that we allowed weights to be positive rational numbers instead of positive integers. This is for technical convenience, but has no effect on the definition. Frequency automata are strictly related to probabilistic automata. Recall that a probabilistic language over Σ^* is a function $D : \Sigma^* \rightarrow [0, 1]$ that is also a discrete distribution, that is:

$$\sum_{x \in \Sigma^*} D(x) = 1.$$

An interesting class of probabilistic languages can be described by a generalization of non-deterministic automata with probabilities as weight on states and transitions.

Definition 4. Probabilistic finite automaton. A probabilistic finite automaton (PFA) is a 5-tuple $A = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$, where:

- Σ is a finite alphabet,
- Q is a finite set of states,
- $I_p : Q \rightarrow (\mathbb{Q} \cap [0, 1])$ is the initial probability such that $\sum_{q \in Q} I_p(q) = 1$,
- $F_p : Q \rightarrow (\mathbb{Q} \cap [0, 1])$,
- $\delta_p : Q \times \Sigma \rightarrow (\mathbb{Q} \cap [0, 1])^Q$ is the transition function such that $\forall q \in Q$,

$$F_p(q) + \sum_{a \in \Sigma, q' \in Q} \delta_p(q, a)(q') = 1.$$

We define the support of a PFA $A = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ is the NFA $\text{supp}(A) = \langle \Sigma, Q, I, F, \delta \rangle$, where $I = \{q \mid I_p(q) > 0\}$, $F = \{q \mid F_p(q) > 0\}$, and $\delta(q, x)(q') = 1$ iff $\delta_p(q, x)(q') > 0$.

Given a string $x = a_1 \cdots a_n \in \Sigma^*$ of length n , an accepting (or valid) path π for x is a sequence of states $q_0 \cdots q_n$ such that:

- $I_p(q_0) > 0$,
- $\delta_p(q_i, a_{i+1})(q_{i+1}) > 0$ for all $0 \leq i < n$, and
- $F_p(q_n) > 0$.

We denote by $\text{Paths}_p(x)$ the set of all accepting paths for a string x . Note that this set is necessarily finite. A probabilistic automaton is said to be unambiguous if for any string $x \in \Sigma^*$ there is at most one path for x . Examples of unambiguous probabilistic automata are the deterministic ones, restricting the initial probability and the transition function to have a support of at most one state:

Definition 5. *Deterministic probabilistic finite automaton.* A PFA $A = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ is called *deterministic probabilistic finite automaton (DPFA)* if

- $|\{q \mid I_p(q) > 0\}| \leq 1$ (at most one single initial state),
- $\forall q \in Q, \forall a \in \Sigma, |\{q' \mid \delta_p(q, a)(q') > 0\}| \leq 1$ ((at most one next state)).

Basically, a DPFA is deterministic if its support is a DFA. All deterministic probabilistic automata are unambiguous, but not all unambiguous automata are deterministic because they can have more than one next state leading to a non-accepting path.

Given a path $\pi = q_0 \cdots q_n$ for a string $x = a_1 \cdots a_n$, we denote by $i_p(\pi)$ the probability $I_p(q_0)$ of its initial state q_0 , by $e_p(\pi)$ the probability $F_p(q_n)$ of the last state q_n of π , and by $\delta_p(\pi)$ the product of all probabilities along the transitions in the path, that is $\delta_p(\pi) = 1$ if x is the empty string and otherwise

$$\delta_p(\pi) = \prod_{i=0}^{n-1} \delta_p(q_i, a_{i+1})(q_{i+1}).$$

Note that $i_p(\pi)$, $e_p(\pi)$ and $\delta_p(\pi)$ are always strictly positive for an accepting path π . Given a probabilistic automaton A , the probability of a path $\pi \in \text{Paths}_p(x)$ is given by $i_p(\pi) \cdot \delta_p(\pi) \cdot e_p(\pi)$, while the probability of a string $x \in \Sigma^*$ is defined by:

$$\llbracket A \rrbracket(x) = \sum_{\pi \in \text{Paths}_p(x)} i_p(\pi) \cdot \delta_p(\pi) \cdot e_p(\pi). \quad (1)$$

A PFA is said to be consistent if all its states appear into at least one accepting path. If a PFA A is consistent then it is easy to show [12] that $\llbracket A \rrbracket$ gives a distribution on Σ^* , that is $\sum_{x \in \Sigma^*} \llbracket A \rrbracket(x) = 1$. A distribution D is called regular if it is generated by a PFA A , that is $D = \llbracket A \rrbracket$.

The language $L(A)$ accepted by a probabilistic automaton A is the support of its distribution and is given by all strings x with a strictly positive probability $\llbracket A \rrbracket(x)$. In other words, $L(A)$ is the language of the support of A . A language is regular if and only if it is accepted by a (deterministic) probabilistic finite automaton. However, differently, than for ordinary automata, the class of distributions characterized by DPFA's is a proper subclass of the regular ones, characterized by PFAs [12].

The following lemma will be useful later stating that if an accepting path contains a cycle then we can pump that cycle to obtain infinitely many other accepting paths.

Lemma 1. *For a probabilistic automaton A , the probability of an accepting path π with a cycle is strictly smaller than 1.*

A useful tool for proving that a regular distribution generated by a PFA A cannot be expressed by a DPFA, is given by the function $\rho_A : \Sigma^* \rightarrow [0, 1]$ defined by

$$\rho_A(x) = \begin{cases} \frac{\llbracket A \rrbracket(x)}{\overline{\llbracket A \rrbracket}(x)} & \text{if } \overline{\llbracket A \rrbracket}(x) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

where $\overline{[A]}(x)$ is the probability of generating in the automaton A a (possibly infinite) string with finite prefix $x \in \Sigma^*$:

$$\overline{[A]}(x) = \sum_{\pi \in \text{Paths}_p(x)} i_p(\pi) \cdot \delta_p(\pi)$$

Note that the above definition does not make use of the final probability F of the automaton A , and as such can be considered as a generator of prefixes of finite and infinite strings. Important here is that if A is a DPFA, the set $\{\rho(x) | x \in \Sigma^*\}$ is necessarily finite and bound by the number of states q with $F_p(q) > 0$ [12].

3 Learning Probabilistic Languages Using Residuals

A *sample* (S, f) consists of a finite set of strings $S \subseteq \Sigma^*$ together with a frequency function $f : S \rightarrow \mathbb{N}$ assigning the number of occurrences of each string in the sample. The frequency function f partitions the strings in S into positive samples and negative ones. We denote by $S_+ = \{x \mid f(x) > 0\}$ the set of positive samples and by $S_- = \{x \mid f(x) = 0\}$ the set of negative samples. A *simple sample* is a sample (S, f) such that $f(x) \leq 1$ for every $x \in S$. In other words, a simple sample consists only of a set of strings that must be accepted together with a set of strings that should not be accepted.

A NFA $A = \langle \Sigma, Q, I, F, \delta \rangle$ is *consistent* with respect to a sample (S, f) , if every positive sample is accepted by A and every negative sample is not, i.e. $S_+ \subseteq L(A)$ and $S_- \cap L(A) = \emptyset$.

A sample (S, f) is *complete* with respect to a regular language L if there exists a finite characteristic set $\mathcal{B}(L) \in \Sigma^*$ such that

- the positive samples cover the language, that is, both x and xa are in $\text{Pref}(S_+)$ for every $x \in \mathcal{B}(L)$ and $a \in \Sigma$,
- the positive samples contain enough strings of L , that is, $\text{Pref}(S_+) \cap L \subseteq S_+$,
- distinguishable strings in the language are distinguishable in the sample too, that is, for every $u, v \in \text{Pref}(S_+)$, if $L_u \not\subseteq L_v$ then there exists $x \in \Sigma^*$ such that $ux \in S_+$ but $vx \in S_-$.

The first condition guarantees that prefixes of strings in S_+ are enough to reach all residual languages of L and to cover all possible transitions from it. The second condition is about requiring all characteristic strings of the residual languages to be in S_+ . And the third condition ensures that S_- is large enough to distinguish different residual languages.

Learning a regular language L from a simple sample (S, f) means building a non-deterministic finite automaton A consistent with the sample and such that if the sample is complete with respect to L , then $L(A) = L$. Of course, one should consider time and space complexity bounded on the two steps above, which are typically required to be polynomial on the number of strings in the sample and of the model representing the language L [7].

Learning a regular distribution D from a sample (S, f) of finite strings independently drawn with a frequency f according to the distribution D means

building a probabilistic finite automaton A with a support learning the language of the support of D and with a distribution associated with A that gets arbitrarily closer to D when the size of the sample (S, f) increases. In general, we cannot realistically expect to get exact information on the learned distribution with respect to the target one.

Next, we present our algorithm to learn an unknown regular distribution D from a sample (S, f) . The idea is to first learn the non-deterministic structure of the automaton underlying D using residual languages, and then labelling the transitions consistently with the frequency of the sample using a fair distribution when needed.

In our first step, we use Algorithm 1 below to build a RFSA from a simple sample (S, f) . The algorithm is similar to that presented in [9] but approximates the inclusion relation between residual languages by calculating on the fly the transitivity and right-invariant (with respect to concatenation) closure $\prec^{tr}$ of the following relation. For $u, v \in Pref(S_+)$, we define:

- $u \prec v$ if there is no string x such that $ux \in S_+$ and $vx \in S_-$,
- $u \simeq v$ if $u \prec v$ and $v \prec u$.

The idea is to characterize all distinguishable states (seen as prefixes of the positive samples). Intuitively, $u \prec^{tr} v$ is an estimate for the inclusion between the residuals $L_u \subseteq L_v$, and if the sample is complete with respect to the unknown language L , this is indeed the case.

Initially, the set of states Q of the automaton is empty. All prefixes of S_+ are explored, and only those which are distinguishable are added to the Q . States below ε with respect to $\prec$ are set to be initial states, while states that belong to S_+ are final ones. Finally, a transition $\delta(u, a) = v$ is added when $v \prec ua$, where $a \in \Sigma$. The algorithm ends either when u is the last string in $Pref$ or when the learned automaton is consistent with the sample.

Example 1. Given a sample (S, f) with $f(\varepsilon) = 3, f(aa) = f(ba) = 2, f(bb) = f(abb) = f(bab) = 1$ and $f(a) = f(b) = f(ab) = f(abb) = 0$ we have $S_+ = \{\varepsilon, aa, ba, bb, abb, bab\}$ and $S_- = \{a, b, ab, aab\}$. The Algorithm 1 terminates in three iterations:

- First, the state ε is added. Since $\varepsilon \prec^{tr} \varepsilon$, the state ε is an initial state, and it is also an accepting state because $\varepsilon \in S_+$. No transitions will be added yet, since a and b are not in S_- and thus distinguishable from ε
- In the next iteration, a is added to the states as $a \not\prec^{tr} \varepsilon$. Clearly, a is neither an initial state nor an accepting one. However, $a \prec \varepsilon a, \varepsilon \prec aa$, so two transitions $\delta(\varepsilon, a) = a$ and $\delta(a, a) = \varepsilon$ are added. As the automaton is not consistent with the sample, another iteration is needed.
- Finally, the state b is added because $b \not\prec^{tr} \varepsilon$ and $b \not\prec^{tr} a$. Also, b is neither initial nor final state because $b \in S_-$. Six transitions are added to the automaton, as $a \prec \varepsilon b, b \prec \varepsilon b, \varepsilon \prec ba, \varepsilon \prec bb, b \prec ab$ and $b \prec ba$. These transitions are $\delta(\varepsilon, b) = a, \delta(\varepsilon, b) = b, \delta(b, a) = \varepsilon, \delta(b, b) = \varepsilon, \delta(a, b) = b$ and $\delta(b, a) = b$. Since the automaton constructed so far is consistent with the sample, the algorithm terminates.

Algorithm 1: Building a RFSA from a simple sample

Input: A simple sample (S, f)
Output: A RFSA $\langle \Sigma, Q, I, F, \delta \rangle$

```

1:  $\text{Pref} := \text{Pref}(S_+)$  ordered by length-lexicographic order
2:  $Q := I := F := \delta := \emptyset$ 
3:  $u := \varepsilon$ 
4: loop
5:   if  $\exists u' \in Q$  such that  $u \simeq^{tr} u'$  then
6:      $\text{Pref} := \text{Pref} \setminus u\Sigma^*$ 
7:   else
8:      $Q := Q \cup \{u\}$ 
9:     if  $u \prec^{tr} \varepsilon$  then
10:       $I := I \cup \{u\}$ 
11:     if  $u \in S_+$  then
12:       $F := F \cup \{u\}$ 
13:     for  $u' \in Q$  and  $a \in \Sigma$  do
14:       if  $u'a \in \text{Pref}$  and  $u \prec^{tr} u'a$  then
15:          $\delta := \delta \cup \{\delta(u', a) = u\}$ 
16:       if  $ua \in \text{Pref}$  and  $u' \prec^{tr} ua$  then
17:          $\delta := \delta \cup \{\delta(u, a) = u'\}$ 
18:   if  $u$  is the last string of  $\text{Pref}$  or  $\langle \Sigma, Q, I, F, \delta \rangle$  is consistent with  $S$  then
19:     exit loop
20:   else
21:      $u := \text{next string in Pref}$ 
22: return  $\langle \Sigma, Q, I, F, \delta \rangle$ 

```

The resulting automaton is shown in Fig. 1a.

Once we have learned the structure of a RFSA from a sample (S, f) , the next step is adding frequencies to get a FFA based on the frequency information of the sample. This step will not change the structure of the automaton, so Σ and Q are the same as the ones resulting from Algorithm 1. Frequency is distributed fairly by dividing it among non-deterministic transitions.

Algorithm 2: Building a FFA from a RFSA

Input: A RFSA $\langle \Sigma, Q, I, F, \delta \rangle$ consistent with a sample (S, f)
Output: A FFA $\langle \Sigma, Q, I_f, F_f, \delta_f \rangle$

```

1:  $I_f(q) := 0$  for all  $q \in Q$ 
2:  $F_f(q) := 0$  for all  $q \in Q$ 
3:  $\delta_f(q, a) := 0$  for all  $q \in Q$  and  $a \in \Sigma$ .
4: for  $a_1 \dots a_n \in S_+$  do
5:   compute  $\text{Paths}(x)$ 
6:   for every  $\pi := q_0 \dots q_n \in \text{Paths}(x)$  do
7:      $I_f(q_0) := I_f(q_0) + \frac{f(x)}{|\text{Paths}(x)|}$ 
8:      $F_f(q_n) := F_f(q_n) + \frac{f(x)}{|\text{Paths}(x)|}$ 
9:     for  $i := 0, i := i + 1, i \leq n - 1$  do
10:       $\delta_f(q_i, a_{i+1})(q_{i+1}) := \delta_f(q_i, a_{i+1})(q_{i+1}) + \frac{f(x)}{|\text{Paths}(x)|}$ 
11: return  $\langle \Sigma, Q, I_f, F_f, \delta_f \rangle$ 

```

It is not hard to prove that the resulting automaton is indeed a FFA, satisfying the frequency preservation condition when passing through states.

Example 2. Continuing from the previous example, let us consider the case of $ba \in S_+$. Two paths are accepting this string, namely $\varepsilon a \varepsilon$ and $\varepsilon b \varepsilon$. As they both start from and end to the same state, $I_f(\varepsilon)$ and $F_f(\varepsilon)$ are incremented by 2, respectively. However, the frequency $f(ba) = 2$ is divided equally between the two b -transitions from state ε , incrementing each of them by 1. After all strings in S_+ are treated, we get the FFA shown in Fig. 1b.

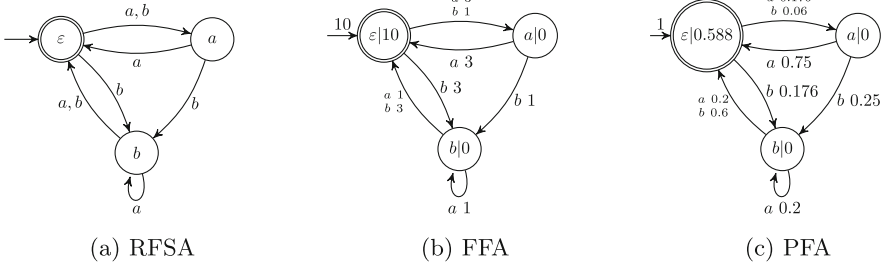


Fig. 1. Three automata learned from the sample (S, f) , with $f(\varepsilon) = 3$, $f(aa) = f(ba) = 2$, $f(bb) = f(abb) = 1 = f(bab) = 1$, and $f(a) = f(b) = f(ab) = f(abb) = 0$.

The last step is the standard for building a PFA from a given FFA. Again, the structure is not modified, but frequencies labelling the transitions and the states are used to calculate the probabilities. In the algorithm below, $FREQ(q)$ denotes the number both of strings either passing through a state q or ending in it, and SUM_I denotes the number of strings entering all initial states. For every state q in Q , the probability of being initial state is $\frac{I_f(q)}{SUM_I}$ and of being final state is $\frac{F_f(q)}{FREQ(q)}$, while the probability associated to each transition from q to q' with input a is $\frac{\delta_f(q,a)(q')}{FREQ(q)}$.

Algorithm 3: Building a PFA from a FFA

Input: A FFA $\langle \Sigma, Q, I_f, F_f, \delta_f \rangle$

Output: A PFA $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$

- 1: **for** $q \in Q$ **do**
 - 2: $FREQ(q) := F_f(q) + \sum_{a \in \Sigma, q' \in Q} \delta_f(q, a)(q')$
 - 3: $F_p(q) := \frac{F_f(q)}{FREQ(q)}$
 - 4: **for** $a \in \Sigma, q' \in Q$ **do**
 - 5: $\delta_p(q, a)(q') := \frac{\delta_f(q, a)(q')}{FREQ(q)}$
 - 6: $SUM_I := \sum_{q \in Q} I_f(q)$
 - 7: **for** $q \in Q$ **do**
 - 8: $I_p(q) := \frac{I_f(q)}{SUM_I}$
 - 9: **return** $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$
-

When the input is a FFA, the above algorithm returns a probabilistic automaton.

Example 3. The probabilistic automaton A resulting from the FFA in Fig. 1b is shown in Fig. 1c. The support automaton is consistent with the sample (S, f) .

4 Metrics for Probabilistic Automata

In the previous section, we have presented an algorithm to learn a distribution presented via a PFA. The support of the learned automaton learns the support language of the original distribution. Precise learning of the distribution itself is not realistic, so next, we consider the problem of computing how close the resulting distribution is to the original one. We consider two methods: one when the original distribution is presented via a PFA itself and another to compute easily understandable metrics such as accuracy, precision, or recall when comparing the learned automaton against a sample.

4.1 The L_2 Distance Between Probabilistic Automata

There are many standard distances that can be used to compare regular distributions by means of their representations as probabilistic automata. Here we will concentrate on L_p metrics using a variation of the algorithm presented in [6] for stochastic weighted automata. The L_p distance between two distributions D_1 and D_2 on Σ^* is defined as

$$L_p(D_1, D_2) = \left(\sum_{x \in \Sigma^*} |D_1(x) - D_2(x)|^p \right)^{\frac{1}{p}}.$$

Examples include the Euclidean distance L_2 and the ‘Manhattan’ distance L_1 . Another useful distance is the L_∞ , adapted from the L_1 by substituting the sum with the supremum. In general, the problem of computing L_{2p+1} and L_∞ given two probabilistic automata is shown to be NP-hard [6, 16], even for automata without cycles.

In this paper we restrict to L_2 using an adaptation of the algorithm to compute it for probabilistic automata by [6]. The basic idea is that

$$\begin{aligned} (L_2(A_1, A_2)) &= \left(\sum_{x \in \Sigma^*} |\llbracket A_1 \rrbracket(x) - \llbracket A_2 \rrbracket(x)|^2 \right)^{\frac{1}{2}} \\ &= \left(\sum_{x \in \Sigma^*} (\llbracket A_1 \rrbracket(x) - \llbracket A_2 \rrbracket(x))^2 \right)^{\frac{1}{2}} \\ &= \left(\sum_{x \in \Sigma^*} \llbracket A_1 \rrbracket(x)^2 - 2\llbracket A_1 \rrbracket(x)\llbracket A_2 \rrbracket(x) + \llbracket A_2 \rrbracket(x)^2 \right)^{\frac{1}{2}} \\ &= \left(\sum_{x \in \Sigma^*} \llbracket A_1 \rrbracket(x)^2 - 2 \sum_{x \in \Sigma^*} \llbracket A_1 \rrbracket(x)\llbracket A_2 \rrbracket(x) + \sum_{x \in \Sigma^*} \llbracket A_2 \rrbracket(x)^2 \right)^{\frac{1}{2}}. \end{aligned} \tag{2}$$

In the second equality, the absolute values can be removed since they are squared. The last three summations can be computed separately via a shortest distance

algorithm for weighted graphs (see below). In general, we consider three different situations.

First, when A_1 and A_2 are acyclic, those summations are finite and can be computed directly.

Second, when A_1, A_2 are deterministic probabilistic automata, we compute their intersection automaton A using the product construction. To avoid computing three intersections, we can keep the label of each transition

$$\delta_p((q_1, q_2), a)(q'_1, q'_2)$$

as a pair $(\delta_{p1}(q_1, a)(q'_1), \delta_{p2}(q_2, a)(q'_2))$, where δ_{p1} is the transition function of A_1 and δ_{p2} is the one of A_2 . When calculating $\llbracket A_i \rrbracket(x)^2$, we only need to square the i -th component of the pair, while we will multiply the two components to calculate $\llbracket A_1 \rrbracket(x)\llbracket A_2 \rrbracket(x)$. This is possible because, for any string $x \in \Sigma^*$, there is at most one accepting path in A_1 and A_2 . In the end, we use the shortest distance algorithm over the intersection automaton with weight modified as described above to compute $\sum_{x \in \Sigma^*} (\llbracket A_1 \rrbracket(x))^i (\llbracket A_2 \rrbracket(x))^{2-i}$ for $i = 0, 1$ and 2 .

Third, when A_1 and A_2 are arbitrary automata, there may be multiple paths with the same label, which means we cannot avoid performing three different intersection automata: one of A_1 with itself, another of A_1 with A_2 , and the last of A_2 with itself. As before, we use the shortest distance algorithm over the intersection automaton to compute $\sum_{x \in \Sigma^*} (\llbracket A_1 \rrbracket(x))^i (\llbracket A_2 \rrbracket(x))^{2-i}$ for $i = 0, 1$ and 2 .

A Shortest Distance Algorithm for Weighted Graphs. Classical shortest paths problems compute the shortest paths from one set of source vertices to all other vertices in a weighted graph. The classical shortest paths problem has been generalized to the weighted graph [18]: The shortest distance from a set of vertices I to a vertex F is the sum of the weights of all paths from nodes in I to nodes in F [18] presented a generic algorithm to compute single-source shortest distances for a directed graph with weight in a semiring. Termination of the algorithm depends on the graph being k -closed, a condition that unfortunately is not satisfied by our probabilistic automata (or their intersection). Therefore we have to adapt the algorithm so as to work with a weaker condition, boundness.

A weighted graph $\langle \Sigma, Q, I, F, \delta \rangle$ consists of a finite alphabet Σ , a finite set of states, an initial weight $I : Q \rightarrow \mathbb{Q}$, a final weight $F : Q \rightarrow \mathbb{Q}$, and a transition function $\delta : Q \times \Sigma \rightarrow \mathbb{Q}^Q$. It is similar to a probabilistic automaton, but it does not need to satisfy its restriction. In fact every probabilistic automaton is a weighted graph, and also the intersection of two probabilistic automata as defined in the previous section is a weighted graph (but, in general, not a probabilistic automaton).

Definition 6. A weighted graph $\langle \Sigma, Q, I, F, \delta \rangle$ is bounded, if for any cycle π there exists a $k \in \mathbb{Q}$ such that:

$$\sum_{n=1}^{\infty} \delta(\pi)^n = k$$

For example, every probabilistic automaton $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ is bounded because the probability of a path with a cycle is always strictly less than 1. It follows that $\sum_{n=1}^{\infty} \delta_p(\pi) = \frac{r}{1-r}$, where $\delta_p(\pi) = r < 1$. Also, the intersection of two probabilistic automata is a bounded weighted graph, but not necessarily a probabilistic automaton because weights need to be normalized.

Next, we provide a shortest distances algorithm for bounded weighted graphs. The pseudo-code is given in Algorithm 4.

Algorithm 4: A shortest distance algorithm for weighted graphs

Input: A bounded weighted graph $\langle \Sigma, Q, I, F, \delta \rangle$
Output: A rational number d , the shortest distance between I and F

```

1: Let  $S$  and  $M$  be an empty set
2: for  $q \in Q$  do
3:   if  $I_p(q) \neq 0$  then
4:      $d[q] := I_p(q)$ 
5:      $r[q] := I_p(q)$ 
6:      $M[q] := \{q\}$ 
7:     add state  $q$  to  $S$ 
8:   else
9:      $d[q] := 0$ 
10:     $r[q] := 0$ 
11: while  $S \neq \emptyset$  do
12:    $q := S[0]$ 
13:   remove  $q$  from  $S$ 
14:   add  $q$  to  $P$ 
15:    $r' := r[q]$ 
16:    $r[q] := 0$ 
17:   for all  $a \in \Sigma, q' \in Q$  do
18:     if  $\delta_p(q, a)(q') \neq 0$  then
19:       if  $q'$  is not in  $M[q]$  then
20:          $M[q'] := M[q] + aq'$ 
21:          $d[q'] := d[q] + (r' \times \delta_p(q, a)(q'))$ 
22:          $r[q'] := r[q'] + (r' \times \delta_p(q, a)(q'))$ 
23:         if  $q' \notin S$  then
24:           add  $q'$  to  $S$ 
25:       else
26:         find cyclic subsequence  $q'xq'$  in  $M[q]$  and store it  $Re$ 
27:         remove alphabet symbols from  $q'xq'$  and store the resulting path in  $\pi$ 
28:         if  $Re \notin M[q']$  then
29:            $l := \delta_p(\pi)$ 
30:            $k := \frac{l}{1-l}$ 
31:            $d[q'] := d[q'] + (r' \times k)$ 
32:            $r[q'] := r[q'] + (r' \times k)$ 
33: for  $q \in Q$  do
34:    $d[q] := d[q] \times F_p[q]$ 
35: return  $d$ 

```

The algorithm uses a set S to maintain the set of next states after transitions and M to store the sequence of transitions visited. S is initialized as a set of initial states. $d[q]$ is the total weight from an initial state to the current state q , $r[q]$ is the weight of the current transition from an initial state to state q .

In the while loop from line 11 to 31, each time we extract a state q from set S , then store the value of $r[q]$ in r' and set $r[q]$ to 0. Lines 17–31 is calculating distances. First, for all transitions starting from state q , if next state q' does not

exist in $M[q]$, update $M[q']$ and the value of $d[q']$ and $r[q']$. If next state q' is not in S , add q' into S . If next state q' exists in $M[q]$, find path π of repetition part, then update $d[q]$. When q is the last state in set S , and there are no more transitions, the while loop ends. In the end, for each state q , $d[q]$ is multiplied by the final weight of the state.

4.2 Metrics Using the Sample

In practice, we usually don't know the target distribution of its PFA representation. So we often metric such as Accuracy, Precision, or Sensitivity when testing a PFA against a sample. To measure the similarity or dissimilarity of strings from the sample and ones from the learned automaton, the learned strings are categorized in terms of a confusion matrix [21], as shown in Table 1.

Table 1. Confusion matrix

Classification by sample	Classification by learned automaton	
	$\omega \in L(A)$	$\omega \notin L(A)$
$\omega \in S_+$	True Positive (TP)	False Negative (FN)
$\omega \in S_-$	False Positive (FP)	True Negative (TN)

Since the confusion matrix only takes into account the support of a probabilistic language, we propose a generalization of true positive and false negative weighted by a confidence measure, based on the L_1 distance between the sample and the distribution of the learned automaton. This leads to a new definition of precision, sensitivity and accuracy for probabilistic automata:

$$Precision = \frac{cTP}{|TP| + cFP}, \quad Sensitivity = \frac{cTP}{|TP| + cFN},$$

$$Accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|}.$$

where $cTP = \sum_{x \in TP} 1 - |P_s(x) - \llbracket A \rrbracket(x)|$, and $cFN = \sum_{x \in FN} P_s(x)$. Here $P_s(x) = \frac{f(x)}{\sum_{y \in S} f(y)}$, is the probability of the string x given the sample S . Similarly, we could define the confidence false positive $cFP = \sum_{x \in FP} \llbracket A \rrbracket(x)$. We do not weight TN with a confidence value, as the probability of not belonging to the sample and to the language of A is both 0, and therefore have 0 distance. Also, note the asymmetry between $|TP|$ and cFP in the denominator of Precision and Sensitivity (TP does not use the confidence extensions). This is because $|TP|$ simply refers to the total number of samples and is needed to average cTP .

When the distribution of the learned automaton coincides with that of the sample, $cTP = |TP| = |S_+|$, $|TN| = |S_-|$, and $|FP| = |FN| = 0$. In this case, precision, sensitivity and accuracy will be all 1. On the other opposite,

when there are no true positive but only false positive and false negative, then $cTP = |TP| = |TN| = 0$, $|FP| = |S_-|$, $|FN| = |S_+|$ and $cFP = cFN = 1$ meaning that the precision, sensitivity and accuracy will be 0.

5 Experimental Results

We used the metrics introduced above to study the performance of our algorithm for learning probabilistic languages. We used different sizes of samples independently draw according to a distribution presented by four different probabilistic automata depicted in Fig. 2: one DPFA, one PFA, one RFSA and one PFA that cannot be expressed by a DPFA. First, we generate a set S of size n of strings from the alphabet by length-lexicographic order and assign the probability of each strings according to the target automaton. Given a fixed number of total occurrences m , we then calculate the frequency of each string in the sample based on its assigned probability. Note that samples generated in this way need not to be complete. All target automata we consider have 3 to 5 states, for which we generate a sample set of size $n < 50$ and total number of occurrences m varying between 10 to 200.

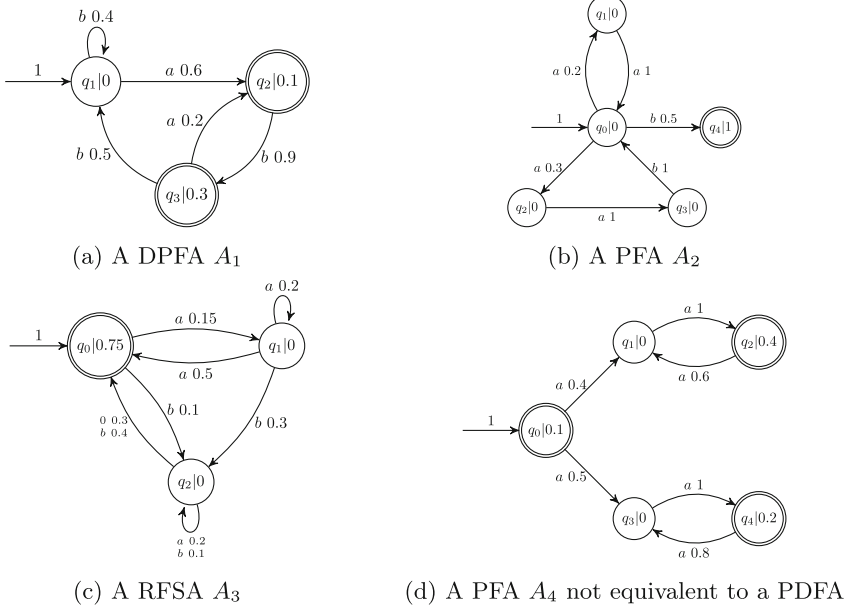


Fig. 2. The four target automata for our experiments

We compare our algorithm to ALERGIA [4] and k-testable algorithms [5]. Contrary to our algorithm presented here, the performance of these other algorithms may be impacted by a parameter setup. For ALERGIA we choose two

different parameters $\alpha = 0.9$ and $\alpha = 0.1$. For k -testable algorithms, we set k to be 2, 3, 4 and 5.

For the case of the DPFA A_1 , the distribution found by all algorithms converges with respect to the L_2 distance rather quickly towards the original one. The 5-testable algorithm has the highest precision and sensitivity and the smallest L_2 distance, but it needs 19 states to learn an automaton of 3. Our algorithm has the best accuracy and is the only one learning the same structure as the original automata (Fig. 3).

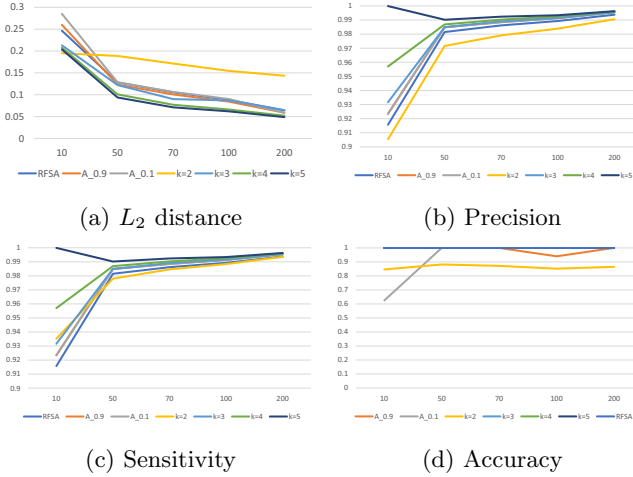


Fig. 3. Results of learning A_1

A similar situation happens when learning the RFSA A_3 . In this case, our algorithm learns a distribution that cannot be described by any DPFA (see appendix for proof that the distribution generated by A_3 cannot be generated by any DFPA). We omit the tables here because of a lack of space.

When considering the PFA A_2 , our algorithm, ALERGIA and 5-testable algorithm outperform all the others, see Fig. 4. Only our algorithm can learn the same number of states but with few more transitions. Accuracy is 1 again. Some errors are introduced because of the fair distribution among non-deterministic transitions.

Finally, we considered the PFA A_4 that cannot be expressed by any DPFA, and that does not have an equivalent RFSA as support, either. All algorithms cannot learn the same structure as the target automaton. Nevertheless, our algorithm achieves the best performance. The L_2 distance is smallest, precision is highest, sensitivity is second highest, and accuracy is always 1 (something not true for all other algorithms). Even if we perform better because the RFSA we learn has the same structure as the support of target distribution, our algorithms will never be able to identify it. We omit the tables here because of a lack of space.

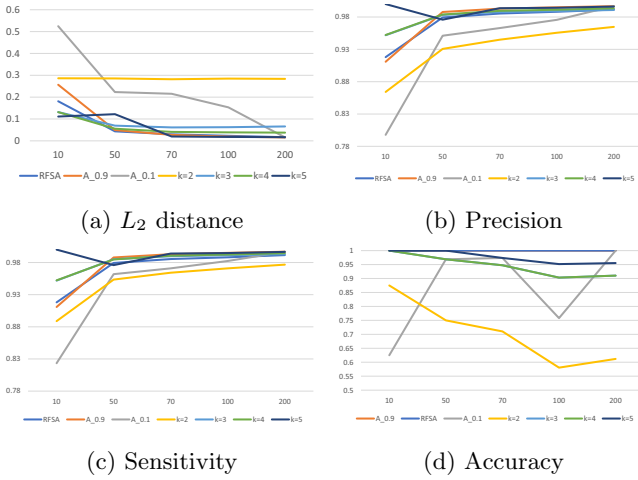


Fig. 4. Results of learning A_2

6 Conclusion

We proposed a new algorithm to learn regular distributions using residual language and adapted existing metrics to evaluate its performance. Our experimental results show that our algorithm can learn the structure of the target automaton efficiently, but distributing probabilities fairly among non-deterministic transitions can cause problems in learning the target distribution at the limit. Other techniques could be used to alleviate this problem and finding a better approximating solution. For example, we will investigate the use of evolutionary computing, and machine learning to better distribute probabilities among non-deterministic transition. Alternatively, we will investigate the use of iterative methods for polynomial constraint solving. Contrary to most existing algorithms, we have shown that our method can learn some PFA with a RFSA support that does not generate a deterministic regular distribution. Furthermore, it would be interesting in having larger samples so to experiment, for example, the impact of passive learning of probabilistic automata in model checking [19, 22]. It would also be interesting to have a deeper analysis of the distance algorithm and the new metrics we introduced. We leave both these points for future work.

A Appendix

In this appendix we prove that the probabilistic language described automaton A_3 cannot generated by any deterministic automata.

Proof. For A_3 , we have

$$\begin{aligned}
 \rho_A(a^{2n}) &= \frac{\llbracket A \rrbracket(a^{2n})}{\llbracket A \rrbracket(a^{2n})} \\
 &= \frac{\sum_{\pi \in \text{Path}_p(a^{2n})} \delta_p(\pi) \cdot e_p(\pi)}{\sum_{\pi \in \text{Path}_p(a^{2n})} \delta_p(\pi)} \\
 &= \frac{\sum_{\pi_0 \in \text{Path}_p(a^{2n})} \delta_p(\pi_0) 0.5 + \sum_{\pi_1 \in \text{Path}_p(a^{2n})} \delta_p(\pi_1) 0}{\sum_{\pi_0 \in \text{Path}_p(a^{2n})} \delta_p(\pi_0) + \sum_{\pi_1 \in \text{Path}_p(a^{2n})} \delta_p(\pi_1)} \\
 &= \frac{\sum_{\pi_0 \in \text{Path}_p(a^{2n})} \delta_p(\pi_0)}{2[\sum_{\pi_0 \in \text{Path}_p(a^{2n})} \delta_p(\pi_0) + \sum_{\pi_1 \in \text{Path}_p(a^{2n})} \delta_p(\pi_1)]}
 \end{aligned} \tag{3}$$

where π_0 is the path for the string x ending at state q_0 , and π_1 is a path for the string x ending at state q_1 . Let r_{2n} denote $\sum_{\pi_0 \in \text{Path}_p(a^{2n})} \delta_p(\pi_0)$, and s_{2n} denote $\sum_{\pi_1 \in \text{Path}_p(a^{2n})} \delta_p(\pi_1)$. Then $\frac{\llbracket A \rrbracket(a^{2n})}{\llbracket A \rrbracket(a^{2n})} = \frac{3r_{2n}}{4(r_{2n} + s_{2n})}$. Suppose $\frac{\llbracket A \rrbracket(a^{2n})}{\llbracket A \rrbracket(a^{2n})} = \frac{\llbracket A \rrbracket(a^{2(n+1)})}{\llbracket A \rrbracket(a^{2(n+1)})}$, we can get:

$$\begin{aligned}
 \frac{\llbracket A \rrbracket(a^{2n})}{\llbracket A \rrbracket(a^{2n})} &= \frac{\llbracket A \rrbracket(a^{2(n+1)})}{\llbracket A \rrbracket(a^{2(n+1)})} \\
 \frac{3r_{2n}}{4(r_{2n} + s_{2n})} &= \frac{3}{4} \frac{r_{2n} \cdot 0.15 \cdot 0.5 + s_{2n} \cdot 0.5 \cdot 0.2}{r_{2n}(0.15 \cdot 0.5 + 0.15 \cdot 0.2) + s_{2n}(0.2 \cdot 0.5 + 0.2 \cdot 0.2 + 0.5 \cdot 0.15)} \\
 \frac{r_{2n}}{r_{2n} + s_{2n}} &= \frac{0.075r_{2n} + 0.1s_{2n}}{0.105r_{2n} + 0.215s_{2n}} \\
 \frac{\frac{r_{2n}}{s_{2n}}}{\frac{r_{2n}}{s_{2n}} + 1} &= \frac{0.075\frac{r_{2n}}{s_{2n}} + 0.1}{0.105\frac{r_{2n}}{s_{2n}} + 0.215}
 \end{aligned} \tag{4}$$

Since $\frac{r_{2n}}{s_{2n}}$ is greater than 0, we get $\frac{r_{2n}}{s_{2n}} = 29.6125$.

$$\frac{r_{2n}}{s_{2n}} = \frac{0.075r_{2(n-1)} + 0.1s_{2(n-1)}}{0.03r_{2(n-1)} + 0.04s_{2(n-1)}} \tag{5}$$

It is easy to find that $\frac{r_{2(n-1)}}{s_{2(n-1)}}$ is strictly smaller than 29.6125, so the set $\{\rho(a^{2n}) \mid n > 0\}$ cannot be finite. Therefore, the automaton show as Fig. 2c cannot be expressed as deterministic probabilistic automaton. $\square$

References

1. Bahl, L.R., Brown, P.F., de Souza, P.V., Mercer, R.L.: Estimating hidden Markov model parameters so as to maximize speech recognition accuracy. *IEEE Trans. Speech Audio Process.* **1**(1), 77–83 (1993)
2. Baldi, P., Brunak, S., Bach, F.: *Bioinformatics: The Machine Learning Approach*. MIT Press, Cambridge (2001)

3. Baum, L.E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Stat.* **41**(1), 164–171 (1970)
4. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Carrasco, R.C., Oncina, J. (eds.) *ICGI 1994*. LNCS, vol. 862, pp. 139–152. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58473-0_144
5. Chu, W., Bonsangue, M.: Learning probabilistic languages by k-testable machines. In: *2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 129–136. IEEE (2020)
6. Cortes, C., Mohri, M., Rastogi, A.: LP distance and equivalence of probabilistic automata. *Int. J. Found. Comput. Sci.* **18**(04), 761–779 (2007)
7. De La Higuera, C.: Characteristic sets for polynomial grammatical inference. *Mach. Learn.* **27**(2), 125–138 (1997)
8. De La Higuera, C., Oncina, J.: Identification with probability one of stochastic deterministic linear languages. In: Gavaldá, R., Jantke, K.P., Takimoto, E. (eds.) *ALT 2003*. LNCS (LNAI), vol. 2842, pp. 247–258. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39624-6_20
9. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using RFSA. In: Abe, N., Khardon, R., Zeugmann, T. (eds.) *ALT 2001*. LNCS, vol. 2225, pp. 348–363. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45583-3_26
10. Denis, F., Lemay, A., Terlutte, A.: Residual finite state automata. *Fund. Inform.* **51**(4), 339–368 (2002)
11. Denis, F., Lemay, A., Terlutte, A.: Some classes of regular languages identifiable in the limit from positive data. In: Adriaans, P., Fernau, H., van Zaanen, M. (eds.) *ICGI 2002*. LNCS (LNAI), vol. 2484, pp. 63–76. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45790-9_6
12. Dupont, P., Denis, F., Esposito, Y.: Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recogn.* **38**(9), 1349–1371 (2005)
13. Esposito, Y., Lemay, A., Denis, F., Dupont, P.: Learning probabilistic residual finite state automata. In: Adriaans, P., Fernau, H., van Zaanen, M. (eds.) *ICGI 2002*. LNCS (LNAI), vol. 2484, pp. 77–91. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45790-9_7
14. Jelinek, F.: *Statistical Methods for Speech Recognition*. MIT Press, Cambridge (1997)
15. Lee, K.F.: *Large-vocabulary speaker-independent continuous speech recognition: the SPHINX system*. Carnegie Mellon University (1988)
16. Lyngsø, R.B., Pedersen, C.N.: The consensus string problem and the complexity of comparing hidden Markov models. *J. Comput. Syst. Sci.* **65**(3), 545–569 (2002)
17. Martin, J.C.: *Introduction to Languages and the Theory of Computation*, vol. 4. McGraw-Hill, New York (1991)
18. Mohri, M.: Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.* **7**(3), 321–350 (2002)
19. Nouri, A., Raman, B., Bozga, M., Legay, A., Bensalem, S.: Faster statistical model checking by means of abstraction and learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) *RV 2014*. LNCS, vol. 8734, pp. 340–355. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_28
20. Seymore, K., McCallum, A., Rosenfeld, R., et al.: Learning hidden Markov model structure for information extraction. In: *AAAI-99 Workshop on Machine Learning for Information Extraction*, pp. 37–42 (1999)

21. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage.* **45**(4), 427–437 (2009)
22. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L*-based learning of Markov decision processes (extended version). arXiv preprint [arXiv:1906.12239](https://arxiv.org/abs/1906.12239) (2019)
23. Thollard, F., Dupont, P., De La Higuera, C., et al.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: *ICML*, pp. 975–982 (2000)