Journal of Logic and Computation Advance Access published January 17, 2013

The most probable string: an algorithmic study

COLIN DE LA HIGUERA, *Université de Nantes, CNRS, LINA, UMR6241, F-44000, France*
E-mail: cdlh@univ-nantes.fr

JOSE ONCINA, *Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante, Alicante, Spain*
E-mail: oncina@dlsi.ua.es

Abstract

The problem of finding the consensus (most probable string) for a distribution generated by a weighted finite automaton or a probabilistic grammar is related to a number of important questions: computing the distance between two distributions or finding the best translation (the most probable one) given a probabilistic finite state transducer. The problem is undecidable with general weights and is $\mathcal{NP}$-hard if the automaton is probabilistic. We give a pseudo-polynomial algorithm that solves a decision problem directly associated with the consensus string and answers if there is a (reasonably short) string whose probability is larger than a given bound in time polynomial in the the size of this bound, both for probabilistic finite automata and probabilistic context-free grammars. We also study a randomized algorithm solving the same problem. Finally, we report links between the length of the consensus string and the probability of this string.

Keywords: Probabilistic automata, parsing.

1 Introduction

When using probabilistic machines to define distributions over sets of strings, the usual and best studied problems are those of parsing and of finding the most probable explanation of a given string (the most probable parse). These problems, when dealing with probabilistic (generating) finite state automata, hidden Markov models (HMMs) or probabilistic context-free grammars depend highly on the ambiguity of the machine: indeed, if there can be different parses for the same string, then the probability of the string is obtained by summing over the different parses.

A more difficult problem is that of finding the most probable string; this string is also known as the *consensus* string.

The problem of finding the most probable string was first addressed in the computational linguistics community by Sima'an [22]: he proved the problem to be $\mathcal{NP}$-hard if we consider tree grammars, and as a corollary he gave the same result for context-free grammars. Goodman [12] showed that, in the case of HMMs, the (decision) problem of finding whether the most probable string of a given length n has a probability at least p is $\mathcal{NP}$-Complete. Moreover, he points out that his technique cannot be applied to show the $\mathcal{NP}$-completeness of the problem when n is not prespecified because the most probable string can be exponentially long. Casacuberta and de la Higuera [5] proved the problem to be $\mathcal{NP}$-hard, using techniques developed for linguistic decoding [4]: their result holds for probabilistic finite state automata and for probabilistic transducers even when these are acyclic; in the transducer case the related (and possibly more important) question is that of finding the most probable translation. The problem was also addressed with motivations in bioinformatics by Lyngsø and Pedersen [15].

doi:10.1093/logcom/exs049

Their technique relies on reductions from maximal cliques. As an important corollary of their hardness results they prove that the L_1 and L_∞ distances between distributions represented by HMMs are also hard to compute: indeed, being able to compute such distances would enable to find (as a side product) the most probable string. This result was then applied on probabilistic finite automata in [6, 7] and the L_k distance, for each odd k was proved to be intractable.

An essential consequence of these results is that finding the most probable translation given some probabilistic (non-deterministic) finite state transducer is also at least as hard. Solving this problem consists in finding the most probable string inside the set of all acceptable translations, and this set is structured as a probabilistic finite automaton [4, 24]. Therefore, the most probable translation problem is also $\mathcal{NP}$-hard.

On the other hand, in the framework of multiplicity automata or of *accepting* probabilistic finite automata (also called Rabin automata), the problem of the existence of a string whose weight is above (or under) a specific threshold is known to be undecidable [2]: this is known as the cut-point emptiness problem. In the case where the weight of each individual edge is between 0 and 1, the score can be interpreted as a probability. The differences reside in the fact that in multiplicity automata the sum of the probabilities of all strings does not need to be bounded; this is also the case for Rabin automata, as each probability corresponds to the probability for a given string to belong to the language.

In this article, we attempt to better understand the status of the decision problem and provide algorithms that allow to find a string of probability higher than a given threshold in time polynomial in the inverse of this threshold. These algorithms give us some pragmatic answers to the consensus string problem, and provide the tools to hope to tackle that difficult and important problem in a systematic way.

We will first (Section 2) give the different definitions concerning automata theory, distributions over strings and complexity theory. In Section 3, we show that we can solve the most probable string problem in time polynomial in the inverse of the probability of this most probable string but in the bounded case, i.e. when we are looking for a string of length smaller than some given bound. In Section 4 we show how we can compute such bounds. In Section 5 the algorithms are experimentally compared and we conclude in Section 6. As the results proposed in this article can also be used to treat the problem of finding the most probable translation, when translations are defined by probabilistic transducers, we show the relevant constructions in the Appendix.

2 Definitions and notations

2.1 Languages and distributions

Let $[n]$ denote the set $\{1,\ldots,n\}$ for each $n\in\mathbb{N}$. An *alphabet* Σ is a finite non-empty set of symbols called *letters*. A *string* w over Σ is a finite sequence $w=a_1\ldots a_n$ of letters. Letters will be indicated by $a,b,c,\ldots$, and strings by $u,v,\ldots,z$. Let $|w|$ denote the length of w. In this case we have $|w|=|a_1\ldots a_n|=n$. The *empty string* is denoted by λ.

We denote by $\Sigma^\star$ the set of all strings, by Σ^n the set of those of length n, by $\Sigma^{<n}$ (respectively $\Sigma^{\leq n}$, $\Sigma^{\geq n}$) the set of those of length less than n (respectively at most n, at least n). When decomposing a string into substrings, we will write $w=w_1\ldots w_n$ where $\forall i\in[n]\, w_i\in\Sigma^\star$.

A *probabilistic language* $\mathcal{D}$ is a probability distribution over $\Sigma^\star$. The probability of a string $x\in\Sigma^\star$ under the distribution $\mathcal{D}$ is denoted as $Pr_\mathcal{D}(x)$ and must verify $\sum_{x\in\Sigma^\star} Pr_\mathcal{D}(x)=1$. If L is a language (thus a set of strings, included in $\Sigma^\star$), and $\mathcal{D}$ a distribution over $\Sigma^\star$, $Pr_\mathcal{D}(L)=\sum_{x\in L} Pr_\mathcal{D}(x)$.

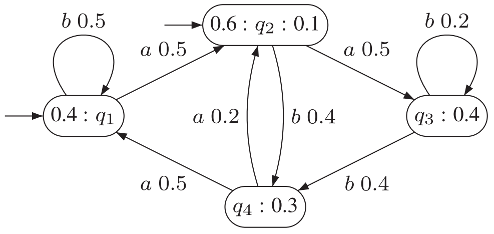

FIGURE 1. Graphical representation of a PFA.

If the distribution is modelled by some syntactic machine $\mathcal{M}$, the probability of x according to the probability distribution defined by $\mathcal{M}$ is denoted $Pr_{\mathcal{M}}(x)$. The distribution modelled by a machine $\mathcal{M}$ will be denoted by $\mathcal{D}_{\mathcal{M}}$ and simplified to $\mathcal{D}$ if the context is not ambiguous.

2.2 *Probabilistic finite automata*

Probabilistic finite automata (PFA) are generative devices for which there are a number of possible definitions [19, 24]. We will principally use PFA without empty (λ) transitions, and this restriction is without loss of generality, as algorithms exist allowing to transform, in polynomial time, more general PFA into PFA respecting the following definition [9, 18]:

DEFINITION 1
A PFA is a tuple $\mathcal{A}=\langle \Sigma, Q, S, F, \delta\rangle$, where:

- Σ is the alphabet;
- $Q = \{q_1, \ldots, q_{|Q|}\}$ is a finite set of *states*;
- $S : Q \to \mathbb{R}\cap[0,1]$ (initial probabilities);
- $F : Q \to \mathbb{R}\cap[0,1]$ (final probabilities);
- $\delta : Q\times\Sigma\times Q \to \mathbb{R}\cap[0,1]$ is a transition function; the function is complete: $\delta(q,a,q')=0$ can be interpreted as 'no transition from q to q' labelled with a'.
  S, δ and F are functions such that:

$$\sum_{q\in Q} S(q)=1, \tag{1}$$

and $\forall q\in Q$,

$$F(q)+\sum_{a\in\Sigma,\ q'\in Q}\delta(q,a,q')=1. \tag{2}$$

An example of PFA is shown in Figure 1.

Let $x\in\Sigma^\star$, $\Pi_{\mathcal{A}}(x)$ is the set of all paths accepting x: an accepting x-path is a sequence $\pi = q_{i_0}a_1q_{i_1}a_2\ldots a_nq_{i_n}$ where $x=a_1\cdots a_n$, $a_i\in\Sigma$, and $\forall j\le n, \exists p_j\neq 0$ such that $\delta(q_{i_{j-1}},a_j,q_{i_j})=p_j$.

The probability of the path π is

$$S(q_{i_0})\cdot\prod_{j\in[n]} p_j\cdot F(q_{i_n})$$

and the probability of the string x is obtained by summing over all the paths in $\Pi_{\mathcal{A}}(x)$. An effective computation can be done by means of the Forward (or Backward) algorithm [24].

Alternatively, a PFA (with d states) is given when the following is known:

- a vector $\mathbf{S} \in \mathbb{R}^{1\times d}$ representing the probabilities of starting at each state. $\mathbf{S}[i]{=}S(q_i)$.
- a map $\mathbf{M}: \Sigma \to \mathbb{R}^{d\times d}$ representing the transition probabilities. $\mathbf{M}_a[i,j] = \delta(q_i, a, q_j)$;
- a vector $\mathbf{F} \in \mathbb{R}^{d\times 1}$ representing the probabilities of ending in each state. $\mathbf{F}[i]{=}F(q_i)$.

Given a string $x = a_1 \cdots a_{|x|}$, with $a_i \in \Sigma$, we compute $Pr_{\mathcal{A}}(x)$ as:

$$Pr_{\mathcal{A}}(x) = \mathbf{S}\prod_{i=1}^{|x|}\mathbf{M}_{a_i}\mathbf{F} \tag{3}$$

Now, Equations 1 and 2 can be written as:

$$\mathbf{S1} = 1$$
$$\sum_{a\in\Sigma}\mathbf{M}_a\mathbf{1} + \mathbf{F} = \mathbf{1}$$

where $\mathbf{1} \in \mathbb{R}^{d\times 1}$ is such that $\forall i\, \mathbf{1}[i] = 1$.

Moreover,

$$\sum_{x\in\Sigma^*} Pr_{\mathcal{A}}(x) = \mathbf{S}\left(\sum_{i=0}^{\infty}\mathbf{M}_\Sigma^i\right)\mathbf{F}$$

where $\mathbf{M}_\Sigma = \sum_{a\in\Sigma}\mathbf{M}_a$.

This geometric series converges to $(I - \mathbf{M}_\Sigma)^{-1}$ if and only if

$$\lim_{i\to\infty}\mathbf{M}_\Sigma^i = \mathbf{0}$$

then, if there are no useless states this condition holds in order for $\mathcal{A}$ to define a probability distribution over $\Sigma^\star$, that is:

$$\sum_{x\in\Sigma^*} Pr_{\mathcal{A}}(x) = 1$$

We will, in the sequel, always suppose that this is the case.

2.3 Complexity classes and decision problems

We only give here some basic definitions and results from complexity theory. A *decision problem* is one whose answer is **true** or **false**. A decision problem is *decidable* if there is an algorithm which, given any specific instance, computes correctly the answer and halts. It is *undecidable* otherwise. A decision problem is in $\mathcal{P}$ if there is a polynomial time deterministic algorithm that solves it.

A decision problem is $\mathcal{NP}$*-complete* if it is both $\mathcal{NP}$-hard and in the class $\mathcal{NP}$: in this case a polynomial time non-deterministic algorithm exists that solves this problem. Alternatively, a problem is in $\mathcal{NP}$ if there exists a *polynomial certificate* for it. A polynomial certificate for an instance I is a short (polynomial length) string which when associated to instance I can be checked in polynomial time to confirm that the instance is indeed positive. A problem is $\mathcal{NP}$*-hard* if it is at least as hard as the satisfiability problem (SAT), or either of the other $\mathcal{NP}$-complete problems [11].

A randomized algorithm makes use of random bits to solve a problem. It *solves a decision problem with one-sided error* if given any value δ and any instance, the algorithm:

- makes no error on a negative instance of a problem (it always answers no);
- makes an error with probability at most δ when working on a positive instance.

A randomized algorithm that solves a decision problem under the conditions above is called a *Monte Carlo algorithm*. If such an algorithm exists and runs in time polynomial in $1/\delta$ and the size of the instance, then the problem is said to belong to the class $\mathcal{RP}$. It should be noted that by running such a randomized algorithm n times the error decreases exponentially with n: if a positive answer is obtained, then the instance had to be positive, and the probability of not obtaining a positive answer (for a positive instance) in n tries is less than δ^n.

When a decision problem depends on an instance containing integer numbers, the fair (and logical) encoding is in base 2. If the problem admits a polynomial algorithm whenever the integers are encoded in base 1, the problem (and the algorithm) are said to be *pseudo-polynomial*.

2.4 About sampling

One advantage of using PFA or similar devices is that they can be effectively used to develop randomized algorithms. But when generating random strings, the fact that the length of these is unbounded is a complex issue: the termination of the sampling algorithm might only be true *with probability 1*; this means that the probability of an infinite run, even if it cannot be discarded, is of null measure.

In the work of Ben-David et al. [1] which extends the definitions from Levin [14], a distribution is considered *samplable* if it is generated by a randomized algorithm that runs in time polynomial in the length of its output.

We will require a stronger condition to be met:

DEFINITION 2
A distribution represented by some machine $\mathcal{M}$ is *samplable in a bounded way* if there is a randomized algorithm which, when given a bound b, will either return any string w in $\Sigma^{\leq b}$ with probability $Pr_{\mathcal{M}}(w)$ or return fail with probability $Pr_{\mathcal{M}}(\Sigma^{>b})$. Furthermore, the algorithm should run in time polynomial in b.

As we also need parsing to take place in polynomial time, we will say that:

DEFINITION 3
A machine $\mathcal{M}$ is strongly *samplable* if

- one can parse an input string x by $\mathcal{M}$ and return $Pr_{\mathcal{M}}(x)$ in time polynomial in $|x|$;
- $\mathcal{D}_{\mathcal{M}}$ is samplable in a bounded way.

For example, if $\mathcal{A}$ is a PFA then $\mathcal{D}_{\mathcal{A}}$ is strongly samplable: parsing takes place, using algorithm Forward, in polynomial time, and $\mathcal{D}_{\mathcal{A}}$ can be sampled in a bounded way.

2.5 The problem

The most important question we pose is to find the most probable string in a probabilistic language. This string is named the *consensus* string.

Name: Consensus string (Cs)
Instance: A probabilistic machine $\mathcal{M}$
Question: Find in $\Sigma^\star$ a string x such that $\forall y \in \Sigma^\star, Pr_{\mathcal{M}}(x) \geq Pr_{\mathcal{M}}(y)$.

For example, for the PFA from Figure 1, the consensus string is a.

With the above problem, the associated decision problem would be to decide, given a probabilistic machine and a string, whether this string is the most probable string in the language or not. We introduce a slightly harder decision problem:

Name: Most probable string (MPS)
Instance: A probabilistic machine $\mathcal{M}$ and $p \geq 0$
Question: Is there in $\Sigma^\star$ a string x such that $Pr_{\mathcal{M}}(x) > p$?

It should be noted that there is a simple reduction to MPS from the natural decision problem: one could use an algorithm for MPS on the probability of the candidate most probable string in order to get confirmation that this string is the consensus string or not. On the other hand, there is no simple reduction to MPS from Cs.

As the problems are defined over probabilities, one should be careful: when mathematical notations are used, probabilities will be numbers from $\mathbb{R}$. But when complexity issues are at stake, the probabilities will be supposed to be encoded as fractions and therefore the complexity of the algorithms depends on the size of the encodings, hence, for a probability p, of $\log \frac{1}{p}$.

MPS is known to be $\mathcal{NP}$-hard [5]. In their proof the reduction is from SAT and uses only acyclic PFA. One problem with MPS is that there is no bound, in general, on the length of the most probable string. Indeed, even for regular languages, this string can be very long. In Section 4.4, we present a construction proving that the most probable string can be of exponential length.

Of interest, therefore, is to study the case where the consensus string can be bounded, with a bound given as a separate argument to the problem:

Name: Bounded most probable string (BMPS)
Instance: A probabilistic machine $\mathcal{M}$, an integer $p \geq 0$, an integer b
Question: Is there in $\Sigma^{\leq b}$ a string x such that $Pr_{\mathcal{M}}(x) > p$?

In BMPS, since strings of length up to b will have to be built, it is necessary, for the problem not to be trivially unsolvable, to accept a unary encoding of b.

3 Solving the bounded most probable string problem

We suppose that the bound b is somehow known. The question is therefore to solve BMPS.

We first solve it in a randomized way, then propose an algorithm that will work if the prefix probabilities can be computed. This is the case for PFA and for probabilistic context-free grammars.

3.1 Solving by sampling

If the machines are **strongly samplable** (following Definition 3), then the problem can be solved:

THEOREM 1
If a machine $\mathcal{M}$ is strongly samplable, then BMPS can be solved by a polynomial-time Monte Carlo algorithm.

```
Data: a machine $\mathcal{M}$, $p \geq 0$, $b \geq 0$
Result: $w \in \Sigma^{\leq b}$ such that $Pr_{\mathcal{M}}(w) > p$,
    false if there is no such $w$
begin
    map f;
    $m = \frac{8}{p} \ln \frac{2}{\delta}$;
    repeat $m$ times
        $w$=bounded_sample($\mathcal{M}$, $b$);
        f [w]++;
    foreach $w$: f [$w$] > $\frac{pm}{2}$ do
        if $\mathrm{Pr}_{\mathcal{M}}(w) > p$ then
            return $w$;
    return false
```

Algorithm 1. `BMPS_sampling`: solving BMPS by sampling.

PROOF. The idea is that any string s whose probability is at least p should appear (with high probability, at least $1-\delta$) in a sufficiently large randomly drawn sample (of size m), and have a relative frequency f/m of at least $p/2$.

Algorithm 1 therefore draws this large enough sample in a bounded way and checks if any of the more frequent strings (relative frequency f/m of at least $p/2$) has real probability at least p. In Algorithm 1 we call function `bounded_sample` that generates a string of length at most b.

We use multiplicative Chernov bounds to compute the probability that an arbitrary string of probability at least p has relative frequency f/m of at least $p/2$:

$$Pr\left(\frac{f}{m} < \frac{p}{2}\right) \leq 2e^{-mp/8}$$

So for a value of $\delta \leq 2e^{-mp/8}$ it is sufficient to randomly draw a sample of size $m \geq (8/p)\ln(2/\delta)$ in order to be certain (with error δ) that in a sample of size m any probable string is in the sample with relative frequency f/m of at least $p/2$.

We then only have to parse each string in the sample which has relative frequency at least $p/2$ to be sure (within error δ) that s is in the sample.

If there is no string with probability at least p, the algorithm will return **false**. ■

The complexity of the algorithm depends on that of bounded sampling and of parsing. One can check that in the case of PFA, the generation is in $O(b \cdot \log|\Sigma|)$ and the parsing (of a string of length at most b) is in $O(b \cdot |Q|^2)$.

3.2 A direct computation in the case of PFA

When the machine is a probabilistic finite automaton, we can do a bit better by making use of simple properties concerning probabilistic languages.

We are given a $p > 0$ and a PFA $\mathcal{A}$. Then we have the following properties:

PROPERTY 1
$\forall u \in \Sigma^\star$, $Pr_{\mathcal{A}}(u\Sigma^\star) \geq Pr_{\mathcal{A}}(u)$.

```
Data: a PFA: $\mathcal{A}=\langle\Sigma,\mathbf{S},\mathbf{M},\mathbf{F}\rangle$, $p\geq 0$, $b\geq 0$
Result: $w\in\Sigma^{\leq b}$ such that $Pr_{\mathcal{A}}(w)>p$,
        false if there is no such $w$
begin
    queue Q ;
    $p_\lambda=\mathbf{SF}$;
    if $p_\lambda>p$ then
        return $\lambda$;
    push(Q, ($\lambda$, S)) ;
    while not empty(Q) do
        ($w$, V) = pop (Q);
        foreach $a\in\Sigma$ do
            $\mathbf{V}'=\mathbf{VM}_a$;
            if $\mathbf{V}'\mathbf{F}>p$ then
                return $wa$;
            if $|w|<b$ and $\mathbf{V}'\mathbf{1}>p$ then
                push(Q, ($wa$, V')) ;
    return false
```

Algorithm 2. `BMPS_exact`: solving BMPS for PFA.

Property 2
For each $n\geq 0$ there are at most $1/p$ strings u in Σ^n such that $Pr_{\mathcal{A}}(u\Sigma^\star)\geq p$.

Both proofs are straightforward and hold not only for PFA but for all distributions. Notice that a stronger version of Property 2 is:

Property 3
There are at most $1/p$ pairwise prefix-incomparable strings u in $\Sigma^\star$ such that $Pr_{\mathcal{A}}(u\Sigma^\star)>p$.

A short analysis of the complexity of Algorithm (`BMPS_exact`) is as follows: for each length n we compute the set of viable prefixes of length n, and keep those whose probability is at least p. The process goes on until either there are no more viable prefixes or a valid string has been found. We use the fact that $Pr_{\mathcal{A}}(ua\Sigma^\star)$ and $Pr_{\mathcal{A}}(u)$ can be computed from $Pr_{\mathcal{A}}(u\Sigma^\star)$ provided we memorize the value in each state (by a standard dynamic programming technique). Property 2 ensures that at every moment at most $1/p$ valid prefixes are open.

If all arithmetic operations are in constant time, the complexity of the algorithm is in $O\big((b|\Sigma|\cdot|Q|^2)/(p)\big)$.

3.3 Sampling vs exact computing

BMPS can be solved with a randomized algorithm (and with error at most δ) or by the direct Algorithm `BMPS_exact`. If we compare costs, and assuming that bounded sampling of a string can be done in time linear in b, and that all arithmetic operations take constant time we have:

- Complexity of (randomized) Algorithm `BMPS_sampling` for PFA is in $O(((8b)/(p))\ln((2)/(\delta))\cdot\log|\Sigma|)$ to build the sample and $O\big(((2b)/(p))\cdot|Q|^2\big)$ to check the $2/p$ most frequent strings.

```
Data: a machine $\mathcal{M}$, $p \geq 0$, $b \geq 0$
Result: $w \in \Sigma^{\leq b}$ such that $Pr_{\mathcal{M}}(w) > p$,
    false if there is no such $w$
begin
    queue Q;
    $p_w = Pr_{\mathcal{M}}(\lambda)$;
    if $p_w > p$ then
        return $\lambda$;
    push(Q, $\lambda$);
    while not empty(Q) do
        $w$ = pop (Q);
        foreach $a \in \Sigma$ do
            if $Pr_{\mathcal{M}}(wa) > p$ then
                return $wa$;
            if $|w| < b$ and $Pr_{\mathcal{M}}(wa\Sigma^*) > p$ then
                push(Q, $wa$);
    return false
```

Algorithm 3. `BMPS_general`: exactly solving BMPS for general machines and grammars.

- Complexity of Algorithm `BMPS_exact` is in $O\big((b|\Sigma|\cdot|Q|^2)/(p)\big)$.

Therefore, for the randomized algorithm to be faster, the alphabet has to be very large. Experiments (see Section 5) show that this is rarely the case.

3.4 Generalizing to other machines

In Algorithm `BMPS_exact` the key is that the different $Pr_{\mathcal{M}}(u\Sigma^\star)$ can be computed. The algorithm can therefore be generalized to any distribution for which there is a machine with which $Pr_{\mathcal{M}}(u\Sigma^\star)$ can be computed.

The more general Algorithm 3 (`BMPS_general`) can be used in such a case (the distributions have also got to be strongly samplable). The complexity can of course be higher, as, in Algorithm `BMPS_exact`, we make use of dynamic programming techniques to avoid computing the prefix probabilities from scratch.

Stolcke [23] proposes an algorithm computing the prefix probabilities for context-free grammars: Algorithm `BMPS_general` can therefore be used for probabilistic context-free grammars.

4 Can the threshold be computed?

The question we now have to answer is: how do we choose the bound, or the threshold passed as a parameter to either of the previous algorithms? In other words, we are given some machine $\mathcal{M}$ and a number $p \geq 0$; we are looking for a value n_p which would be the smallest integer such that $Pr_{\mathcal{M}}(x) \geq p \implies |x| \leq n_p$. If we can compute this bound we can run one of the algorithms from the previous section.

4.1 Analytically computing n_p

If given the machine $\mathcal{M}$ we can compute the mean μ and the variance σ of the length of strings in $\mathcal{D}_{\mathcal{M}}$, we can use Chebychev's inequality:

$$Pr_{\mathcal{M}}\Big(\big||x|-\mu\big|>k\sigma\Big)<\frac{1}{k^2}$$

We now choose $k=1/\sqrt{p}$ and rewrite:

$$Pr_{\mathcal{M}}\Big(|x|>\mu+\frac{\sigma}{\sqrt{p}}\Big)<p$$

This means that, if we are looking for strings with a probability bigger than p, it is not necessary to consider strings longer than $\mu+\big(\sigma/\sqrt{p}\big)$.

In other words, we can set $b=\lceil\mu+\big(\sigma/\sqrt{p}\big)\rceil$ and run an algorithm from Section 3 which solves BMPS.

4.2 Analytically computing n_p for PFA

We consider the special case where the probabilistic machine is a PFA $\mathcal{A}$. We are interested in computing the mean and the variance of the string length. We note that it is irrelevant whether the PFA is deterministic or not.

The mean string length of the strings generated by $\mathcal{A}$ can be computed as:

$$\begin{aligned}\mu&=\sum_{i=0}^{\infty} i\,Pr_{\mathcal{A}}(\Sigma^i)\\&=\sum_{i=0}^{\infty} i\,\mathbf{S}\mathbf{M}_{\Sigma}^{i}\mathbf{F}\\&=\mathbf{S}\mathbf{M}_{\Sigma}(I-\mathbf{M}_{\Sigma})^{-2}\mathbf{F}\end{aligned}$$

Moreover, taking into account that:

$$\begin{aligned}\sum_{i=0}^{\infty} i^2 Pr_{\mathcal{A}}(\Sigma^i)&=\sum_{i=0}^{\infty} i^2\mathbf{S}\mathbf{M}_{\Sigma}^{i}\mathbf{F}\\&=\mathbf{S}\mathbf{M}_{\Sigma}(I+\mathbf{M}_{\Sigma})(I-\mathbf{M}_{\Sigma})^{-3}\mathbf{F}\end{aligned}$$

The variance can be computed as:

$$\begin{aligned}\sigma^2&=\sum_{i=0}^{\infty}(i-\mu)^2 Pr_{\mathcal{A}}(\Sigma^i)\\&=\sum_{i=0}^{\infty} i^2 Pr_{\mathcal{A}}(\Sigma^i)-\mu^2\\&=\mathbf{S}\mathbf{M}_{\Sigma}(I+\mathbf{M}_{\Sigma})(I-\mathbf{M}_{\Sigma})^{-3}\mathbf{F}\\&\quad-\Big[\mathbf{S}\mathbf{M}_{\Sigma}(I-\mathbf{M}_{\Sigma})^{-2}\mathbf{F}\Big]^2\end{aligned}$$

Then, both values are finite since $(I-\mathbf{M}_\Sigma)$ is non-singular.

Moreover, to compute μ and σ^2 only a constant number of matrix sums, multiplications and inversions are required. As all these operations (over Rationals) can be computed in polynomial time, the bound $b = \lceil \mu + \frac{\sigma^2}{p} \rceil$ not only is always finite but can be computed in polynomial time.

4.3 *Computing $n_{p,\delta}$ via sampling*

In certain cases we cannot obtain an analytical value for the mean and the variance. This may be the case, for instance, for probabilistic context-free grammars. We have to resort to sampling in order to compute an estimation of n_p.

A sufficiently large sample is built and used by Lemma 1 to obtain our result. In that case we have the following:

- If the instance is negative, it is impossible to find a string with high enough probability, so the answer will always be **false**.
- If the instance is positive, then the bound returned by the sampling will be good in all but a small fraction (less than δ) of cases. When the sampling has gone correctly, then the algorithm when it halts has checked all the strings up to length n. And the total weight of the remaining strings is less than p.

The general goal of this section is to compute, given a strongly samplable machine $\mathcal{M}$ capable of generating strings following distribution $\mathcal{D}_\mathcal{M}$ and a positive value p, an integer $n_{p,\delta}$ such that $Pr_{\mathcal{D}_\mathcal{M}}(\Sigma^{\geq n_{p,\delta}}) < p$. If we do this by sampling we will of course have the result depend also on the value δ covering the case where the sampling process went abnormally wrong.

Lemma 1
Let $\mathcal{D}$ be a distribution over $\Sigma^\star$. Then if we draw, following distribution $\mathcal{D}$, a sample S of size at least $(1/p)\ln(1/\delta)$, given any $p>0$ and any $\delta>0$, the following holds with probability at least 1-δ: the probability of sampling a string x longer than any string seen in S is less than p.

Alternatively, if we write $n_S = \max\{|y| : y \in S\}$, then, with probability at least $1-\delta$, $Pr_\mathcal{D}(|x| > n_S) < p$.

Proof. Denote by m_p the smallest integer such that the probability for a randomly drawn string to be longer than m_p is less than p: $Pr_\mathcal{D}(\Sigma^{>m_p}) < p$.

We need now to compute a large enough sample to be sure (with a possible error of at most δ) that $\max\{|y| : y \in S\} \geq m_p$. For $Pr_\mathcal{D}(|x| > m_p) < p$ to hold, a sufficient condition is that we take a sample large enough to be nearly sure (i.e. with probability at least $1-\delta$) to have at least one string as long as m_p. On the contrary, the probability of having all (k) strings in S of length less than m_p is at most $(1-p)^k$. Using the fact that $(1-p)^k > \delta$ implies that $k > (1/p)\ln(1/\delta)$, it follows that it is sufficient, once we have chosen δ, to take $n_{p,\delta} > (1/p)\ln(1/\delta)$ to have a correct value. ■

Note that in the above, all we ask is that we are able to sample. This is indeed the case with HMM, PFA and (well defined) probabilistic context-free grammars, provided these are not expansive. Lemma 1 therefore holds for any such machines.

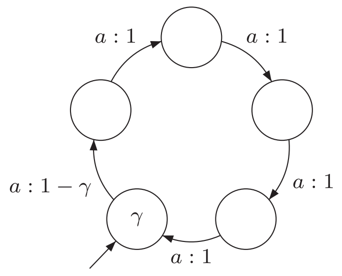

FIGURE 2. Automaton for $\psi = 5$.

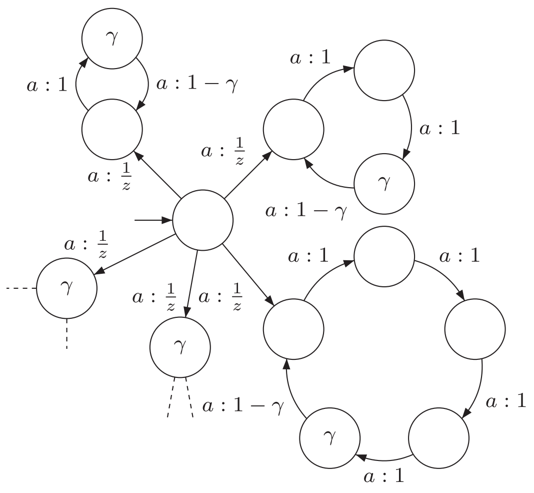

FIGURE 3. An automaton whose most probable string is of exponential length.

4.4 *The consensus string can be of exponential length*

If the consensus string can be very long, how long might it be? We show now an automaton for which the most probable string is of exponential length with the size of the automaton. The construction is based on one by de la Higuera [8]. Let us use a value $\gamma > 0$ whose exact value we will compute later.

We first note (Figure 2) how to build an automaton that only gives non-null probabilities to strings whose length are multiples of ψ for any value of ψ (and of particular interest are the prime numbers). Here, $Pr(a^{k\psi}) = \gamma(1-\gamma)^k$.

We now extend this construction by building for a set of prime numbers $\{\psi_1, \psi_2, \ldots, \psi_z\}$ the automaton for each ψ_i and adding an initial state. This PFA can be constructed as proposed in Figure 3, and has $1+\sum_{i=1}^{z}\psi_i$ states. When parsing a non-empty string, a sub-automaton will only contribute if the length of the string is a multiple of ψ_i.

The probability of string a^k with $k = \prod_{i=1}^{z} p_i$ is $\sum_{i=1}^{z}\frac{1}{z}\gamma(1-\gamma)^{\frac{k}{\psi_i}-1} = \frac{\gamma}{z}\sum_{i=1}^{z}(1-\gamma)^{\frac{k}{\psi_i}-1}$.

First consider a string of length less than k. This string is not accepted by at least one of the sub-automata so its probability is at most $\gamma\,((z-1)/z)$.

On the other hand we prove now that for a good value of γ, $Pr(a^k) > \gamma\,((z-1)/z)$.

We simplify by noticing that since $(k/\psi_i)-1 \leq k$, $(1-\gamma)^{\frac{k}{\psi_i}-1} \geq (1-\gamma)^k$.

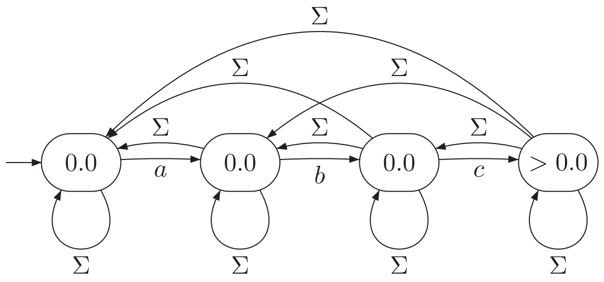

FIGURE 4. Topology of the automaton.

So $Pr(a^k) \geq \frac{\gamma}{z} \sum_{i=1}^{z} (1-\gamma)^k = \gamma(1-\gamma)^k$.

$$(1-\gamma)^k > \frac{z-1}{z}$$
$$1-\gamma > \sqrt[k]{\frac{z-1}{z}}$$
$$\gamma < 1 - \sqrt[k]{\frac{z-1}{z}}$$

No shorter string can have higher probability.

And since $\prod_{i=1}^{z} p_i$ increases exponentially with $1 + \sum_{i=1}^{i=z} p_i$ (the number of states of the PFA), we are done.

5 Experiments

We report here some experiments in which we compared the proposed algorithms over probabilistic automata.

In order to have languages where the consensus string is not very short, we generated a set of random automata with a linear topology, only one initial state and one final state, and where transitions were added leading from each state to all previous states labelled by all the symbols of the vocabulary. The topology of this PFA is illustrated in Figure 4.

The probabilities on the edges and the final state were set by assigning randomly (uniformly) distributed numbers in the range $[0,1]$ and then normalizing.

In order to have a cut bound for the sampling (Algorithm 1) and the exact (Algorithm 2) algorithm, a branch and bound algorithm to find the most probable string was implemented. This algorithm is very similar to the exact algorithm but finishes when the queue is empty. In order to have a measure, independent of the machine, of the computational resources needed by the algorithm the number of multiplications performed in each run of the algorithms were counted.

Table 1 shows the average, for 10 random 4-state automata with increasing vocabulary sizes (from 2 to 14), of the number of multiplications needed by the sampling and the exact algorithms alongside with the probability of the most probable string and the number of multiplications needed by the branch and bound algorithm to find it.

In all cases, the algorithms gave the correct answer to the MPS question. In our experiments, the exact algorithm (2) is systematically faster (by several orders of magnitude) than the one that uses sampling (1).

TABLE 1. Number of multiplications made by Algorithm 1 (`BMPS_sampling`), Algorithm 2 (`BMPS_exact`), and a branch and bound version for 4-state automata. **Prob** indicates the probability of the consensus string

Vocabulary size	**Prob**	`BMPS_sampling`	`BMPS_exact`	`B&B`
2	1.71e−03	5.37e+06	2.45e+02	9.99e+04
3	3.30e−04	2.72e+07	6.86e+02	4.59e+05
4	1.22e−04	1.70e+08	1.33e+03	2.86e+06
5	8.64e−05	9.43e+07	2.21e+03	1.81e+06
6	4.03e−05	1.42e+08	3.00e+03	2.84e+06
7	2.47e−05	2.44e+08	5.87e+03	5.37e+06
8	1.82e−05	2.79e+08	8.53e+03	6.69e+06
9	8.23e−06	9.47e+08	1.04e+04	2.36e+07
10	4.40e−06	1.25e+09	1.50e+04	2.97e+07
11	3.65e−06	1.51e+09	1.59e+04	3.76e+07
12	4.56e−06	1.05e+09	1.90e+04	3.27e+07
13	1.90e−06	4.37e+09	2.63e+04	1.17e+08
14	2.27e−06	1.71e+10	4.59e+04	1.10e+08

A number of questions arise following these experiments:

- Can we build alternative settings (possibly artificial and unlikely) which would be favourable to the randomized algorithm?
- If larger automata are used, in which the number of states (which is a key parameter) increases, can there be an advantage to use the randomized version?
- Can tighter statistical bounds be used, allowing the size of the randomly drawn sample to be smaller?

6 Conclusion

We have proved in this article the following:

1. If we can sample and parse (such a distribution is called *strongly samplable*), then we have a randomized algorithm that solves BMPS.
2. If furthermore we can analytically compute the mean and variance of the distribution, there is an exact algorithm for BMPS. This implies that the problem is decidable for a PFA or an HMM.
3. In the case of PFA the mean and the variance are polynomially computable, so BMPS can be solved in time polynomial in the size of the PFA and in $1/p$.
4. There exists a PFA whose most probable string is not of polynomial length (Section 4.4), so it is impossible to hope for a polynomial time algorithm depending only on the size of the PFA.

Many of these results can be extended or adapted to other probabilistic machines:

- Probabilistic context-free grammars also define distributions over strings. An extension of the Early algorithm due to Stolcke [23] computes the probability that x occurs as a prefix of some string generated by G. This allows us to use Algorithm 3 to solve BMPS for context-free grammars.
- Hidden markov Models (HMMs) [13, 21] are finite state machines defined by (1) a finite set of states, (2) a probabilistic transition function, (3) a distribution over initial states and (4) an output function. The question of finding the most probable output has been addressed by

Lyngsø and Pedersen [15]. Equivalence results between HMMs and PFA can be found in [24]; these can be used for the proposed constructions from this work to apply.

- Probabilistic transducers are used to compute probabilities for pairs of strings, or of a string being the translation of another. It can be shown (see Appendix) that finding the most probable translation of a given string can be reduced to finding the most probable string. The constructions presented in the Appendix therefore allow to use these results to that setting.

Practically, the crucial problem is Cs; solving the decision problem is of a more technical interest. If we are given the bounds (both on the probability and on the length of the consensus string) we have shown how to solve the problem. If not, we believe some of the techniques introduced here can help, by either sampling to obtain a lower bound for the probability of the most probable string and solving BMPS, or by some form of binary search.

Further experiments are needed to see in what cases the sampling algorithm works better, and also to check its robustness with more complex models (like probabilistic context-free grammars).

Finally, the relation between the probability of the most probable string and its length is a key question.

Acknowledgements

Discussions with Pascal Koiran and Khalil Sima'an during different stages of this work were of great help. This is an extended version of a work presented at IWPT 2011 [10]. The first author acknowledges partial support by the Région des Pays de la Loire. The second author thanks the Spanish CICyT for partial support of this work through projects TIN2009-14205-C04-C1, and the program Consolider Ingenio 2010 (CSD2007-00018). This work was supported in part by the IST Programme of the European Community, under the Pascal 2 Network of Excellence, IST-2007-216886.

References

[1] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, **44**, 193–219, 1992.

[2] V. D. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computer Systems*, **36**, 231–245, 2003.

[3] F. Casacuberta. Probabilistic estimation of stochastic regular syntax-directed translation schemes. In *VI Spanish Symposium on Pattern Recognition and Image Analysis*, pp. 201–297. Asociación Española de Reconocimiento de Formas y Análisis de Imagenes, 1995.

[4] F. Casacuberta and C. de la Higuera. Optimal linguistic decoding is a difficult computational problem. *Pattern Recognition Letters*, **20**, 813–821, 1999.

[5] F. Casacuberta and C. de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In *Proceedings of the International Colloquium on Grammatical Inference (ICGI) 2000*. Vol. 1891 of *Lecture Notes in Artificial Intelligence*, pp. 15–24. Springer, 2000.

[6] C. Cortes, M. Mohri, and A. Rastogi. On the computation of some standard distances between probabilistic automata. In *Proceedings of the Conference on Implementation and Application of Automata (CIAA) 2006*. Vol. 4094 of *Lecture Notes in Computer Science*, pp. 137–149. Springer, 2006.

[7] C. Cortes, M. Mohri, and A. Rastogi. l_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, **18**, 761–779, 2007.

[8] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning Journal*, **27**, 125–138, 1997.

[9] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[10] C. de la Higuera and J. Oncina. Finding the most probable string and the consensus string: an algorithmic study. In *Proceedings of the 12th International Conference on Parsing Technologies*, pp. 26–36. The Association for Computational Linguistics, 2011.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.

[12] J. T. Goodman. *Parsing Inside–Out*. Ph.D. Thesis. Harvard University, 1998.

[13] F. Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1998.

[14] L. Levin. Average case complete problems. *SIAM Journal on Computing*, **15**, 285–286, 1986.

[15] R. B. Lyngsø and C. N. S. Pedersen. The consensus string problem and the complexity of comparing hidden Markov models. *Journal of Computing and System Science*, **65**, 545–569, 2002.

[16] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, **23**, 269–311, 1997.

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

[18] M. Mohri. Generic E-removal and input E-normalization algorithms for weighted transducers. *International Journal on Foundations of Computer Science*, **13**, 129–143, 2002.

[19] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

[20] D. Picó and F. Casacuberta. Some statistical-estimation methods for stochastic finite-state transducers. *Machine Learning Journal*, **44**, 121–141, 2001.

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

[22] K. Sima'an. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, Vol. cmp-lg/9606019, pp. 1175–1180, 1996.

[23] A. Stolcke. An efficient probablistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, **21**, 165–201, 1995.

[24] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite state automata—part I and II. *Pattern Analysis and Machine Intelligence*, **27**, 1013–1039, 2005.

Received 10 February 2012

Appendix: probabilistic transducers

There are different definitions of probabilistic transducers. We follow here [24]: *Probabilistic finite-state transducers* (PFST) are similar to PFA, but in this case two different alphabets (source Σ and target Γ) are involved. Each transition in a PFST has attached a symbol from the source alphabet (or λ) and a (possible empty) string of symbols from the target alphabet. PFSTs can be viewed as graphs, as for example in Figure A1.

Definition 4 (Probabilistic transducer)
A *probabilistic finite state transducer* (PFST) is a 6-tuple $\langle Q, \Sigma, \Gamma, S, E, F \rangle$ such that:

- Q is a finite set of *states*; these will be labelled $q_1, \ldots, q_{|Q|}$;
- $S : Q \to \mathbb{R} \cap [0,1]$ (initial probabilities);

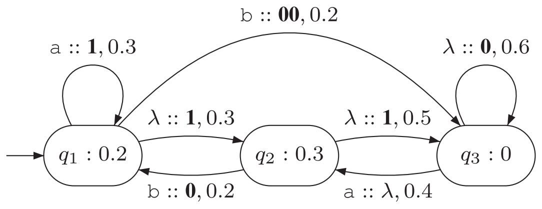

FIGURE A1. Transducer.

- $F: Q \to \mathbb{R} \cap [0,1]$ (halting probabilities);
- $E \subset Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^\star \times Q \times \mathbb{R}$ is the finite set of transitions;

S and F are functions such that:

$$\sum_{q \in Q} S(q) = 1,$$

and $\forall q \in Q$,

$$F(q) + \sum_{(q,a,w,q',p) \in E} p = 1.$$

Let $x \in \Sigma^\star$ and $y \in \Gamma^\star$. Given a probabilistic transducer $\mathcal{T}$, let $\Pi_{\mathcal{T}}(x,y)$ be the set of all paths accepting (x,y): a path is a sequence $\pi = q_{i_0}(x_1,y_1)q_{i_1}(x_2,y_2)\ldots(x_n,y_n)q_{i_n}$ where $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_n$, with $\forall j \in [n], x_j \in \Sigma \cup \{\lambda\}$ and $y_j \in \Gamma^\star$, and $\forall j \in [n], \exists p_{i_j}$ such that $(q_{i_{j-1}}, x_j, y_j, q_{i_j}, p_{i_j}) \in E$. The probability of the path is

$$S(q_{i_0}) \cdot \prod_{j \in [n]} p_{i_j} \cdot F(q_{i_n})$$

And the probability of the translation pair (x,y) is obtained by summing over all the paths in $\Pi_{\mathcal{T}}(x,y)$.

Note that the probability of y given x (the probability of y as a translation of x, denoted as $Pr_{\mathcal{T}}(y|x)$) is $\frac{Pr_{\mathcal{T}}(x,y)}{\sum_{z \in \Sigma^\star} Pr_{\mathcal{T}}(x,z)}$.

Probabilistic finite state transducers are used as models for the *stochastic translation problem* of a source sentence $x \in \Sigma^\star$; it can be defined as the search for a target string y that:

$$\operatorname*{argmax}_y Pr(y \mid x) = \operatorname*{argmax}_y Pr(y,x).$$

The problem of finding this optimal translation is proved to be $\mathcal{NP}$-hard [5]. An approximate solution can be computed in polynomial time using an algorithm similar to the Viterbi algorithm for probabilistic finite-state automata [3, 20].

The stochastic translation problem is computationally tractable in particular cases. If the PFST $\mathcal{T}$ is *non-ambiguous in the translation sense* (no input string has two non-null translations: $Pr_{\mathcal{T}}(x,y) > 0$ and $Pr_{\mathcal{T}}(x,y') > 0 \implies y = y'$), the translation problem is polynomial. If the PFST $\mathcal{T}$ is *simply non-ambiguous* (no string pair (x,y) has two different non-null parses), then the translation problem is also polynomial. In both cases, the computation can be carried out using an adequate version of the Viterbi algorithm [24].

Alternative types of PFSTs have been introduced and applied with success in machine translation. These have sometimes been called *weighted finite-state transducers* [16, 17].

The stochastic translation problem can be solved as a most probable string problem. Indeed, the set of all translations of a given string can be represented by a PFA. We describe in this section the

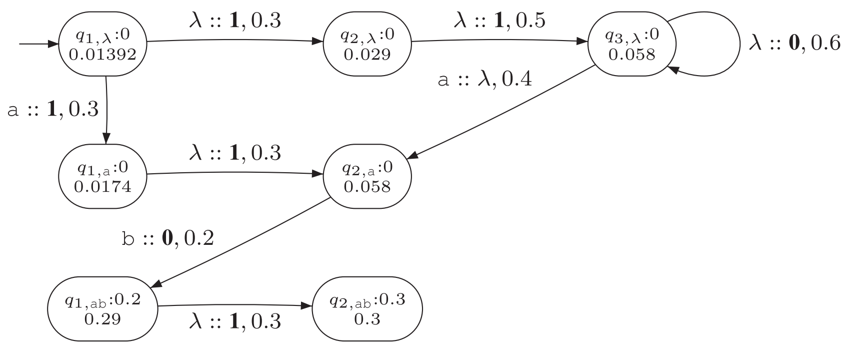

FIGURE A2. Corresponding non-normalized PFST for the translations of ab. Each state indicates which input prefix has been read. The second line in the states gives the values $\mathbf{w}_q$.

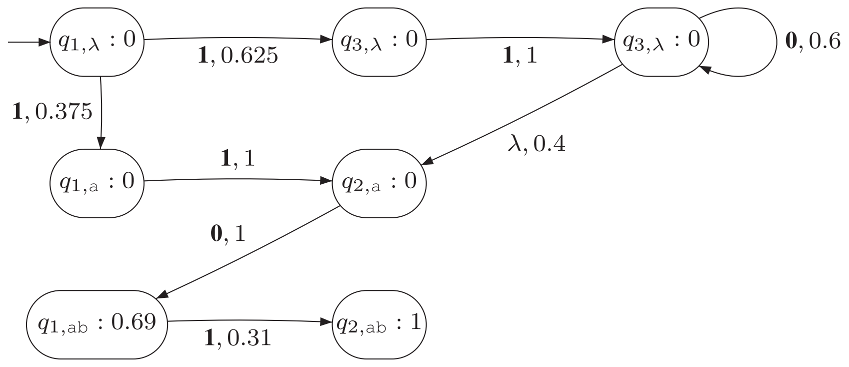

FIGURE A3. Corresponding normalized PFA for the translations of ab. The most probable string (**110**) has probability 0.43125.

construction for this: this construction can then be used to compute effectively the most probable translation of a given string.

The transformation is illustrated in Figures A1, A2, and A3.

Let us first note that, for a fixed x, $Pr(x,y)$ is not a distribution of probability, and that it is unclear whether a unique finite state machine can be used to represent all the conditional probabilities $Pr(y|x)$.

The approach that we follow here is to build, for a given x, the PFA $\mathcal{A}_x$ that describes $Pr_{\mathcal{T}}(y|x)$:

$$Pr_{\mathcal{A}_x}(y) = Pr_{\mathcal{T}}(y|x) = \frac{Pr_{\mathcal{T}}(x,y)}{\sum_z Pr_{\mathcal{T}}(x,z)}$$

Let $\mathcal{T}_x$ be the following restriction[1] of the transducer $\mathcal{T}$:

$$\mathcal{T}_x(z,y) = \begin{cases} \mathcal{T}(z,y) & \text{if } z = x \\ 0 & \text{otherwise} \end{cases}$$

[1]Note that the construction requires to manipulate transducers and PFA that can read strings and not just symbols over the transitions.

Note that this transducer is in general not normalized: $\sum_{z,y} Pr_{\mathcal{T}_x}(z,y) = \sum_y Pr_{\mathcal{T}_x}(x,y) \neq 1$. We will address this issue afterwards.

This transducer can be built from $\mathcal{T}$ by making $|x|+1$ copies of the states of $\mathcal{T}$ and ensuring that the only possible input string is x:

$$\begin{aligned}
Q_{\mathcal{T}_x} &= \{q_{r,z} : r \in Q_{\mathcal{T}}, z \in \mathrm{Pref}(x)\} \\
S_{\mathcal{T}_x}(q_{r,z}) &= \begin{cases} S_{\mathcal{T}_x}(r) & \text{if } z = \lambda \\ 0 & \text{otherwise} \end{cases} \\
F_{\mathcal{T}_x}(q_{r,z}) &= \begin{cases} F_{\mathcal{T}_x}(r) & \text{if } z = x \\ 0 & \text{otherwise} \end{cases} \\
(r,a,w,r',p) \in E_{\mathcal{T}} &\Rightarrow (q_{r,z},a,w,q_{r',za},p) \in E_{\mathcal{T}_x} \forall za \in \mathrm{Pref}(x), |a| \leq 1
\end{aligned}$$

For $x = \mathtt{ab}$, the transducer $\mathcal{T}_{\mathtt{ab}}$ corresponding to the transducer from Figure A1 is represented in Figure A2. It is easy to check that the restricted transducer $\mathcal{T}_{\mathtt{ab}}$ contains all the eligible paths that allow to parse ab in the initial transducer.

The transducer has to be trimmed by removing all unreachable and absorbing states. In the running example, this trimming leads to removing states $q_{3,\mathtt{a}}$ and $q_{3,\mathtt{ab}}$ (not represented in Figure A2).

For each state q of $\mathcal{T}_x$ let $\mathbf{w}_q$ be the sum of the probabilities of all the paths that start in q. These quantities can be obtained by parsing in $\mathcal{T}_x$ or by solving the linear system of equations given by:

$$\mathbf{w}_q = \sum_{(q,a,w,q',p) \in E_{\mathcal{T}_x}} p\mathbf{w}_{q'} + F_{\mathcal{T}_x}(q)$$

Then,

$$\sum_y Pr_{\mathcal{T}}(x,y) = \sum_y Pr_{\mathcal{T}_x}(x,y) = \sum_{q \in Q_{\mathcal{T}_x}} S_{\mathcal{T}_x}(q)\mathbf{w}_q.$$

In a similar way we introduce the notation

$$\mathbf{w}_x = \sum_z Pr_{\mathcal{T}}(x,z)$$

Now, we build the PFA $\mathcal{A}_x$ from $\mathcal{T}_x$ as:

$$\begin{aligned}
Q_{\mathcal{A}_x} &= Q_{\mathcal{T}_x} \\
S_{\mathcal{A}_x}(q) &= \frac{\mathbf{w}_q}{\mathbf{w}_x} S_{\mathcal{T}_x}(q) \\
F_{\mathcal{A}_x}(q) &= \frac{1}{\mathbf{w}_q} F_{\mathcal{T}_x}(q) \\
(q,w,q',\frac{\mathbf{w}_{q'}}{\mathbf{w}_q}p) \in E_{\mathcal{A}_x} &\iff (q,a,w,q',p) \in E_{\mathcal{T}_x} \quad \forall a \in \Sigma \cup \{\lambda\}.
\end{aligned}$$

Note that, by construction, given q and q' there is only one transition in $E_{\mathcal{T}_x}$ that goes from q to q' so there is no ambiguity in the last line of the construction.

$\mathcal{A}_x$ is well defined.

That is,

$$\sum_q S_{\mathcal{A}_x}(q) = \sum_q \frac{\mathbf{w}_q}{\mathbf{w}_x} S_{\mathcal{T}_x}(q) = \frac{1}{\mathbf{w}_x} \sum_q \mathbf{w}_q S_{\mathcal{T}_x}(q) = \frac{\mathbf{w}_x}{\mathbf{w}_x} = 1$$

and

$$\begin{aligned} F(q) + \sum_{(q,w,q',p)\in E_{\mathcal{A}_x}} p &= \frac{F_{\mathcal{T}_x}(q)}{\mathbf{w}_q} + \sum_{(q,a,w,q',p)\in E_{\mathcal{T}_x}} p\frac{\mathbf{w}_{q'}}{\mathbf{w}_q} \\ &= \frac{1}{\mathbf{w}_q}\left(F_{\mathcal{T}_x}(q) + \sum_{(q,a,w,q',p)\in E_{\mathcal{T}_x}} p\mathbf{w}_{q'}\right) \\ &= \frac{\mathbf{w}_q}{\mathbf{w}_q} = 1 \end{aligned}$$

Now, let $\pi_{\mathcal{T}_x} = q_{i_0}(x_1,y_1)q_{i_1}(x_2,y_2)\ldots(x_n,y_n)q_{i_n}$ be a path for the output string y in $\mathcal{T}_x$, then there exists a path $\pi_{\mathcal{A}_x} = q_{i_0}y_1q_{i_1}y_2\ldots y_nq_{i_n}$ in $\mathcal{A}_x$.

So,

$$\begin{aligned} Pr_{\mathcal{A}_x}(\pi_{\mathcal{A}_x}) &= S_{\mathcal{T}_x}(q_{i_0})\frac{\mathbf{w}_{q_0}}{\mathbf{w}_x}\left(\prod_{j\in[n]} p_{i_j}\frac{\mathbf{w}_{q_{i_j}}}{\mathbf{w}_{q_{i_{j-1}}}}\right)\frac{1}{\mathbf{w}_{q_{i_n}}}F(q_{i_n}) \\ &= \frac{1}{\mathbf{w}_x} S_{\mathcal{T}_x}(q_{i_0}) \prod_{j\in[n]} p_{i_j} F(q_{i_n}) \\ &= \frac{1}{\mathbf{w}_x} Pr_{\mathcal{T}_x}(\pi_{\mathcal{T}_x}) \end{aligned}$$

and,

$$\begin{aligned} Pr_{\mathcal{A}_x}(y) &= \sum_{\pi\in\Pi(y)} Pr_{\mathcal{A}_x}(\pi) \\ &= \sum_{\pi\in\Pi(y)} \frac{1}{\mathbf{w}_x} Pr_{\mathcal{T}_x}(\pi) \\ &= \frac{Pr_{\mathcal{T}_x}(x,y)}{\mathbf{w}_x} \\ &= Pr_{\mathcal{T}}(y|x). \end{aligned}$$

This construction was applied to the non-normalized transducer $\mathcal{T}_{\mathrm{ab}}$ from Figure A2: the result is represented in Figure A3.

Since $((Pr_{\mathcal{T}}(x,y))/\mathbf{w}_x) = Pr_{\mathcal{T}}(y|x)$, the above construction can be used to compute both joint and conditional probabilities. Therefore, an algorithm computing the consensus string for PFA could be used to solve the stochastic translation problem and both the related decision problems: is there a string y such that $Pr_{\mathcal{T}}(x,y) > c$ or such that $Pr_{\mathcal{T}}(y|x) > c$, given a constant c?