Finding the Most Probable String and the Consensus String: an Algorithmic Study

Colin de la Higuera[*] and **Jose Oncina**[**]
[*]Université de Nantes, CNRS, LINA, UMR6241, F-44000, France
cdlh@univ-nantes.fr
[**]Departamento de Lenguajes y Sistemas Informaticos
Universidad de Alicante, Alicante, Spain
oncina@dlsi.ua.es

Abstract

The problem of finding the 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 which computes the most probable string in time polynomial in the inverse of the probability of the most probable string itself, both for probabilistic finite automata and probabilistic context-free grammars. We also give a randomised algorithm solving the same problem.

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 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 we study here 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 (1996): 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 (1998) showed that, in the case of HMMs, the problem of finding whether the most most probable string of a given length n is at least p is $\mathcal{NP}$-Complete. Moreover, he points 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 (2000) proved the problem to be $\mathcal{NP}$-hard, using techniques developed for linguistic decoding (Casacuberta and de la Higuera, 1999): 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 (2002). 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 (Cortes et al., 2006; Cortes et al., 2007) 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. It can be shown (Casacuberta and de la Higuera, 1999; Vidal et al., 2005) that 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. 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 (Blondel and Canterini, 2003). 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 paper we attempt to better understand the status of the problem and provide algorithms which find a string of probability higher than a given threshold in time polynomial in the inverse of this threshold. These algorithms give us pragmatic answers to the consensus string problem as it is possible to use the probabilistic machine to define a threshold and to use our algorithms to find, in this way, the most probable string.

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 compute the most probable string 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.

2 Definitions and Notations

2.1 Languages and Distributions

Let $[n]$ denote the set $\{1, \dots, n\}$ 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 $w = a_1 \dots a_n$ of letters. Let $|w|$ denote the length of w. In this case we have $|w| = |a_1 \dots a_n| = n$. The *empty string* is denoted by λ. When decomposing a string into substrings, we will write $w = w_1 \dots w_n$ where $\forall i \in [n]\ w_i \in \Sigma^\star$.

Letters will be indicated by $a, b, c, \dots,$ and strings by $u, v, \dots, z$.

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).

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 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.

If L is a language (thus a set of strings, included in $\Sigma^\star$), and $\mathcal{D}$ a distribution over $\Sigma^\star$, $Pr_{\mathcal{D}}(L) = \sum_{x \in L} Pr_{\mathcal{D}}(x)$.

2.2 Probabilistic Finite Automata

The probabilistic finite automata (PFA) (Paz, 1971) are generative devices:

Definition 1. *A* Probabilistic Finite Automaton (PFA) *is a tuple* $\mathcal{A} = \langle \Sigma, Q, S, F, \delta \rangle$, *where:*

- Σ *is the alphabet;*
- $Q = \{q_1, \dots, q_{|Q|}\}$ *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);*
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times Q \to \mathbb{R}^+$ *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 $\forall q \in Q$,

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

Let $x \in \Sigma^\star$. $\Pi_{\mathcal{A}}(x)$ is the set of all paths accepting x: a path is a sequence $\pi = q_{i_0} x_1 q_{i_1} x_2 \dots x_n q_{i_n}$ where $x = x_1 \cdots x_n$, $x_i \in \Sigma \cup \{\lambda\}$, and $\forall j \leq n, \exists p_j \neq 0$ such that $\delta(q_{i_{j-1}}, x_j, q_{i_j}) = p_j$. 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 $\Pi_{\mathcal{A}}(x)$. Note that this may result in an infinite sum because of λ-transitions (and more problematically λ-cycles.

An effective computation can be done by means of the Forward (or Backward) algorithm (Vidal et al., 2005).

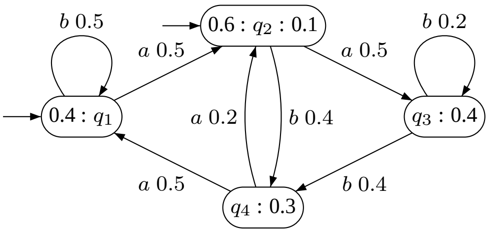

Figure 1: Graphical representation of a PFA.

Alternatively, a PFA (with n states) is given when the following matrices are known:

- $\mathbf{S} \in \mathbb{R}^{1\times n}$ represents the probabilities of starting at each state. $\mathbf{S}[i]$=$S(q_i)$;
- $\mathbf{M} = \{\mathbf{M}_a \in \mathbb{R}^{n\times n} | a \in \Sigma \cup \{\lambda\}\}$ represents the transition probabilities. $\mathbf{M}_a[i,j] = \delta(q_i, a, q_j)$;
- $\mathbf{F} \in \mathbb{R}^{n\times 1}$ represents the probabilities of ending in each state. $\mathbf{F}[i]$=$F(q_i)$.

Given a string $x = a_1 \cdots a_k$ we compute $Pr_{\mathcal{A}}(x)$ as:

$$Pr_{\mathcal{A}}(x) = \mathbf{S} \prod_{i=1}^{|x|} [\mathbf{M}_\lambda^* \mathbf{M}_{a_i}] \, \mathbf{M}_\lambda^* \mathbf{F} \qquad (3)$$

where

$$\mathbf{M}_\lambda^* = \sum_{i=0}^{\infty} M_\lambda^i = (I - \mathbf{M}_\lambda)^{-1}$$

Then, equations 1 and 2 can be written as:

$$\mathbf{S1} = 1 \qquad (4)$$

$$\sum_{a \in \Sigma \cup \{\lambda\}} \mathbf{M}_a \mathbf{1} + \mathbf{F} = \mathbf{1} \qquad (5)$$

where $\mathbf{1} \in \mathbb{R}^n$ is such that $\forall i \; \mathbf{1}[i] = 1$.
Note that

$$Pr_{\mathcal{A}}(\lambda) = \mathbf{S}\mathbf{M}_\lambda^* \mathbf{F} \in [0,1] \qquad (6)$$

This implies that $\mathbf{M}_\lambda^*$ should be a non singular matrix.

Moreover, in order for $Pr_{\mathcal{A}}$ to define a distribution probability over $\Sigma^\star$ it is required that:

$$\sum_{x \in \Sigma^*} Pr_{\mathcal{A}}(x) = \sum_{i=0}^{\infty} \mathbf{S}\mathbf{M}_\lambda^* \mathbf{M}_\Sigma^i \mathbf{M}_\lambda^* \mathbf{F}$$
$$= \mathbf{S}\mathbf{M}_\lambda^* (I - \mathbf{M}_\Sigma)^{-1} \mathbf{M}_\lambda^* \mathbf{F} = 1$$

where I is the identity matrix and $\mathbf{M}_\Sigma = \sum_{a \in \Sigma} \mathbf{M}_a$. Note that as a consequence of that, $(I - \mathbf{M}_\Sigma)$ is a non singular matrix.

2.3 Hidden Markov Models

Hidden Markov models (HMMs) (Rabiner, 1989; Jelinek, 1998) 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.

An HMM generates a string by visiting (in a hidden way) states and outputting values when in those states. Typical problems include finding the most probable path corresponding to a particular output (usually solved by the Viterbi algorithm). Here the question of finding the most probable output has been addressed by Lyngsø and Pedersen (2002). In this paper the authors prove that the hardness of this problem implies that it is also hard to compute certain distances between two distributions given by HMMs.

Note that to obtain a distribution over $\Sigma^\star$ and not each Σ^n the authors introduce a unique final state in which, once reached, the machine halts. An alternative often used is to introduce a special symbol ($\sharp$) and to only consider the strings terminating with $\sharp$: the distribution is then over $\Sigma^\star \, \sharp$.

Equivalence results between HMMs and PFA can be found in (Vidal et al., 2005).

2.4 Probabilistic Context-free Grammars

Definition 2. *A* probabilistic context-free grammar (PCFG) G *is a quintuple* $< \Sigma, V, R, P, N >$ *where* Σ *is a finite alphabet (of terminal symbols),* V *is a finite alphabet (of variables or nonterminals),* $R \subset V \times (\Sigma \cup V)^*$ *is a finite set of production rules, and* N $(\in V)$ *is the axiom.* $P : R \rightarrow \mathbb{R}^+$ *is the probability function.*

A PCFG is used to generate strings by rewriting iteratively the non terminals in the string, starting from the axiom. A string may be obtained by different derivations. In this case the problem is called ambiguity. Parsing with a PCFG is usually done by adapting the Earley or the CKY algorithms.

Particularly appealing is a very efficient extension of the Early algorithm due to Stolcke (1995) that can compute:

- the probability of a given string x generated by a PCFG G;
- the single most probable parse for x;
- the probability that x occurs as a prefix of some string generated by G, which we denote by $Pr_G(x\,\Sigma^\star)$.

2.5 Probabilistic Transducers

There can be different definitions of probabilistic transducers. We use the one from (Vidal et al., 2005):

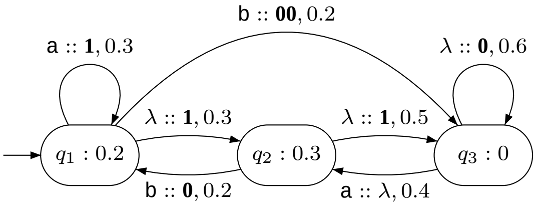

Figure 2: Transducer.

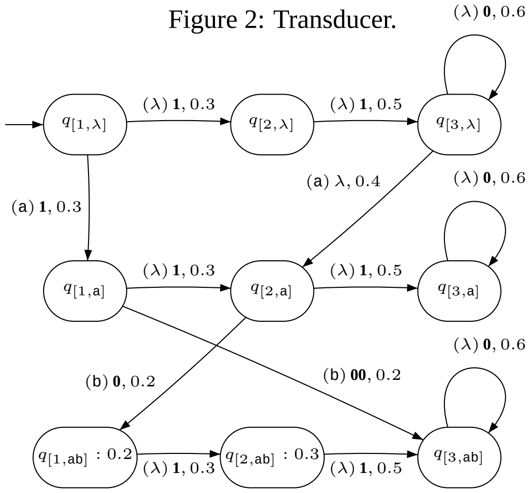

Figure 3: Corresponding non normalized PFA for the translations of ab. Each state indicates which input prefix has been read. Between the brackets, on the transitions, the input symbol justifying the transition.

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 string (possible empty string) of symbols from the target alphabet. PFSTs can be viewed as graphs, as for example in Figure 3.

Definition 3 (Probabilistic transducer)**.** *A* probabilistic *finite state* transducer *(*PFST*) is a 6-tuple* $\langle Q, \Sigma, \Gamma, S, E, F\rangle$ *such that:*

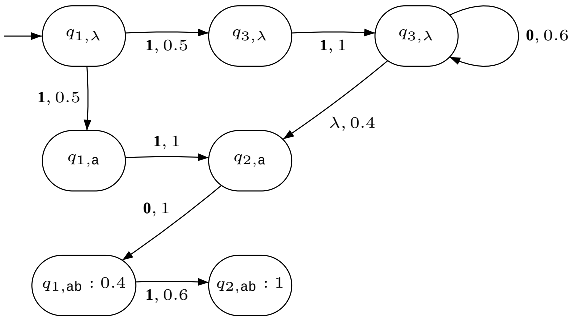

Figure 4: Corresponding normalized PFA for the translations of ab. The most probable string (**111**) has probability 0.54.

- Q *is a finite set of* states*; these will be labelled* q_1,*. . .* , $q_{|Q|}$*;*
- $S : Q \to \mathbb{R}^+ \cap [0, 1]$ *(initial probabilities);*
- $F : Q \to \mathbb{R}^+ \cap [0, 1]$ *(halting probabilities);*
- $E \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^\star \times Q \times \mathbb{R}^+$ *is the set of transitions;*

S, δ and F are functions such that:

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

and $\forall q \in Q$*,*

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

Let $x \in \Sigma^\star$ and $y \in \Gamma^\star$. 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 the *stochastic translation problem* of a source sentence $x \in \Sigma^\star$ that 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 a $\mathcal{NP}$-hard by Casacuberta and de la Higuera (2000).

An approximate solution to the stochastic translation can be computed in polynomial time by using an algorithm similar to the Viterbi algorithm for probabilistic finite-state automata (Casacuberta, 1995; Picó and Casacuberta, 2001).

The stochastic translation problem is computationally tractable in particular cases. If the PFST $\mathcal{T}$ is *non-ambiguous in the translation sense* ($\forall x \in \Sigma^\star$ there are not two target sentences $y, y' \in \Gamma^\star$, $y \neq y'$, such that $Pr_\mathcal{T}(x, y) > 0$ and $Pr_\mathcal{T}(x, y') > 0$), the translation problem is polynomial. If the PFST $\mathcal{T}$ is simply *non-ambiguous* ($\forall x \in \Sigma^\star$ there are not two different paths that deal with (x, y) and with probability different to zero), the translation problem is also polynomial. In both cases, the computation can be carried out using an adequate version of the Viterbi algorithm (Vidal et al., 2005).

Alternative types of PFSTs have been introduced and applied with success in different areas of machine translation. In (Mohri, 1997; Mohri et al., 2000), *weighted finite-state transducers* are studied.

2.6 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* if not. A decision problem is in $\mathcal{P}$ if there is a polynomial time 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 always 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 (Garey and Johnson, 1979).

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 in at most δ cases when working on a positive instance.

If such an algorithm exists, the problem is said to belong to the class $\mathcal{RP}$. It should be noticed 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. A randomized algorithm which solve a decision problem in the conditions above is called a *Monte Carlo algorithm*.

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*.

2.7 About Sampling

One advantage of using PFA or similar devices is that they can be effectively used to develop randomised algorithms. But when generating random strings, the fact that the length of these is unbounded is an issue. Therefore the termination of the 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. (1992) which extends Levin's original definitions from (Levin, 1986), a distribution over $\{0, 1\}^*$ 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. We want a distribution represented by some machine $\mathcal{M}$ to be *sampable in a bounded way*, ie, we require that 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 a machine $\mathcal{M}$ is *stronly* *sampable* if

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

2.8 The Problem

The question is to find the most probable string in a probabilistic language. An alternative name to this string is the *consensus* string.

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

With the above problem we associate the following decision problem:

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

For example, if we consider the PFA from Figure 1, the most probable string is a.

Note that p is typically encoded as a fraction and that the complexity of our algorithms is to depend on the size of the encodings, hence of $\log \frac{1}{p}$.

The problem MPS is known to be $\mathcal{NP}$-hard (Casacuberta and de la Higuera, 2000). In their proof the reduction is from SAT and uses only acyclic PFA. There is a problem with MPS: there is no bound, in general, over the length of the most probable string. Indeed, even for regular languages, this string can be very long. In Section 4.4 such a construction is presented.

Of interest, therefore, is to study the case where the longest 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 $\mathcal{M}$, a $p \geq 0$, an integer b
Question: Is there in $\Sigma^{\leq b}$ a string x such that $Pr_{\mathcal{M}}(x) \geq p$?

In complexity theory, numbers are to be encoded in base 2. In BMPS, it is necessary, for the problem not to be trivially unsolvable, to consider a unary encoding of b, as strings of length up to b will have to be built.

3 Solving BMPS

In this section we attempt to solve the bounded case. We first solve it in a randomised way, then propose an algorithm that will work each time the prefix probabilities can be computed. This is the case for PFA and for probabilistic context free grammars.

3.1 Solving by Sampling

Let us consider a class of **strongly sampable** machines.

Then BMPS, for this class, belongs to $\mathcal{RP}$:

Theorem 1. *If a machine $\mathcal{M}$ is strongly sampable,* BMPS *can be solved by a Monte Carlo algorithm.*

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 $\frac{f}{m}$ of at least $\frac{p}{2}$.

Algorithm 1 therefore draws this large enough sample in a bounded way and then checks if any of the more frequent strings (relative frequency $\frac{f}{m}$ of at least $\frac{p}{2}$) has real probability at least p.

We use multiplicative Chernov bounds to compute the probability that an arbitrary string whose probability is at least p has relative frequency $\frac{f}{m}$ of at least $\frac{p}{2}$:

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

So for a value of $\delta \leq 2e^{-mp/8}$ it is sufficient to draw a sample of size $m \geq \frac{8}{p} \ln \frac{2}{\delta}$ 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 $\frac{f}{m}$ of at least $\frac{p}{2}$.

We then only have to parse each string in the sample which has relative frequency at least $\frac{p}{2}$ 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 $\mathcal{O}(b \cdot \log |\Sigma|)$ and the parsing (of a string of length at most b) is in $\mathcal{O}(b \cdot |Q|^2)$.

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.

```
Data: a machine $\mathcal{M}$, $p \geq 0, b \geq 0$
Result: $w \in \Sigma^{\leq b}$ such that $Pr_{\mathcal{M}}(w) \geq p$,
    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 $[w] \geq \frac{pm}{2}$ do
        if $\mathrm{Pr}_{\mathcal{M}}(w) \geq p$ then
            return $w$;
    return false
end
```

Algorithm 1: Solving BMPS in the general case

```
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 $Pr_{\mathcal{A}}(u) \geq p$,
false if there is no such $w$
begin
    Queue Q;
    $p_\lambda = \mathbf{SF}$;
    if $p_\lambda \geq p$ then
        return $p_\lambda$;
    push( Q, $(\lambda, \mathbf{F})$);
    while not empty(Q) do
        $(w, \mathbf{V})$ = pop (Q);
        foreach $a \in \Sigma$ do
            $\mathbf{V}' = \mathbf{V}\mathbf{M}_a$;
            if $\mathbf{V}'\mathbf{F} \geq p$ then
                return $\mathbf{V}'\mathbf{F}$;
            if $|w| < b$ and $\mathbf{V}'\mathbf{1} \geq p$ then
                push( Q, $(wa, \mathbf{V}')$);
    return false
end
```

Algorithm 2: Solving BMPS for automata

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

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

Property 2. *For each $n \geq 0$ there are at most $\frac{1}{p}$ 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:

Property 3. *If X is a set of strings such that (1) $\forall u \in X, Pr_{\mathcal{A}}(u\,\Sigma^\star) \geq p$ and (2) no string in X is a prefix of another different string in X, then $|X| \leq \frac{1}{p}$.*

Analysis and complexity of Algorithm 2. The idea of the algorithm is as follows. For each length n 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 $\frac{1}{p}$ valid prefixes are open.

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

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 2. If we compare costs, and assuming that bounded sampling a string can be done in time linear in b, and that all arithmetic operations take constant time we have:

- Complexity of (randomized) Algorithm 1 for PFA is in $\mathcal{O}(\frac{8b}{p} \ln \frac{2}{\delta} \cdot \log |\Sigma|)$ to build the sample and $\mathcal{O}(\frac{2b}{p} \cdot |Q|^2)$ to check the $\frac{2}{p}$ most frequent strings.

- Complexity of Algorithm 2 is in $\mathcal{O}(\frac{b|\Sigma|\cdot|Q|^2}{p})$.

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 Generalising to Other Machines

What is really important in Algorithm 2 is that the different $Pr_{\mathcal{M}}(u\,\Sigma^\star)$ can be computed. If this is a case, the algorithm can be generalized and will work with other types of machines. This is the case for context-free grammars (Stolcke, 1995).

For classes which are strongly sampable, we propose the more general Algorithm 3.

4 More about the Bounds

The question we now have to answer is: how do we choose the bound? We are given some machine $\mathcal{M}$ and a number $p \geq 0$. We are looking for a value n_p which is 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.

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

Algorithm 3: Solving BMPS for general machines

4.1 Computing Analytically 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 $k = \frac{1}{\sqrt{p}}$ 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 $\mu + \frac{\sigma}{\sqrt{p}}$.

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

4.2 Computing Analytically 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. It can be noted that the fact the PFA is deterministic or not is not a problem.

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

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

Moreover, taking into account that:

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

The variance can be computed as:

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

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

4.3 Computing $n_{p,\delta}$ via Sampling

In certain cases we cannot draw an analytically obtained value for the mean and the variance. 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 anyhow impossible to find a string with high enough probability, so the answer will always be **false**.
- If the instance is positive, 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 strogly sampable 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 $Pr_{\mathcal{D}_{\mathcal{M}}}(\Sigma^{n_{p,\delta}}) < p$. 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 $\frac{1}{p}\ln\frac{1}{\delta}$, 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 $k > \frac{1}{p}\ln\frac{1}{\delta}$, it follows that it is sufficient, once we have chosen δ, to take $n_{p,\delta} > \frac{1}{p}\ln\frac{1}{\delta}$ 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) probablistic context-free grammars, provided these are not expansive. Lemma 1 therefore holds for any of such machines.

4.4 The Most Probable String Can Be of Exponential Length

If the most probable 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 (de la Higuera, 1997). Let us use a value $\gamma > 0$ whose exact value we will compute later.

We first note (Figure 5) 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, $Pr(a^{k\psi}) = \gamma(1-\gamma)^k$.

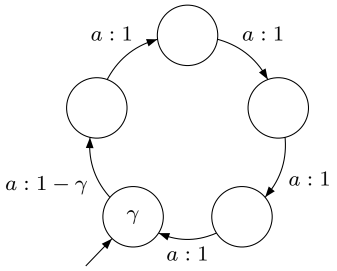

Figure 5: Automaton for $(a^5)^*$.

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. When parsing a non empty string, a sub-automaton will only add to the mass of probabilities if the string is of length multiple of ψ_i. This PFA can be constructed as proposed in Figure 6, and has $1 + \sum_{i=1}^{i=z}\psi_i$ states.

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

First consider a string of length less than k. This string is not accepted by at least one of the sub-automata so it's probability is at most $\gamma\frac{z-1}{z}$.

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

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

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

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

no shorter string can have higher probability.

5 Experiments

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

In order to have languages where the most probable 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

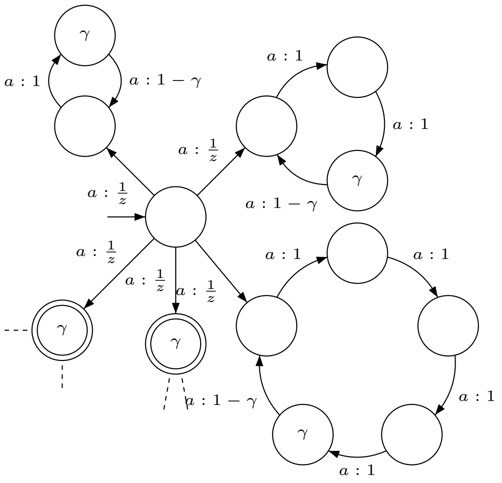

Figure 6: An automaton whose smallest 'interesting string' is of exponential length.

previous states labelled by all the symbols of the vocabulary.

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

Voc size	Sampling (s)	Exact (s)
2	0.34	0.00
4	13.26	0.00
6	13.80	0.01
8	31.85	0.02
10	169.21	0.09
12	156.58	0.10

Table 1: Execution time of Algorithm 1 (sampling) and Algorithm 2 (exact) for 4 state automata

In our experiments, the exact algorithm is systematically faster than the one that uses sampling.

Alternative settings which would be favourable to the randomized algorithm are still to be found.

6 Conclusion

We have proved the following:

1. There exists a PFA whose most probable string is not of polynomial length.

2. If we can sample and parse (strongly sampable distribution), then we have a randomised algorithm which solves MPS.

3. If furthermore we can analytically compute the mean and variance of the distribution, there is an exact algorithm for MPS. This means that the problem is decidable for a PFA or HMMs.

4. In the case of PFA the mean and the variance are polynomially computable, so MPS can be solved in time polynomial in the size of the PFA and in $\frac{1}{p}$.

5. In the case of PFA, we can use practical algorithms:

   (a) randomly draw a sample S of n strings following distribution $\mathcal{D}_{\mathcal{A}}$;

   (b) let $p = \max\{p(u) : u \in S\}$ and $b = \max\{|u| : u \in S\}$;

   (c) run Algorithm 2 using p and b.

Practically, the crucial problem may be CS; A consensus string can be found by either sampling to obtain a lower bound to the probability of the most probable string and solving MPS, 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, in Section 4.4 we showed that the length of the most probable string could be exponential, but it is unclear if a higher bound to the length can be obtained.

Acknowledgement

Discussions with Pascal Koiran during earlier stages of this work were of great help towards the understanding of the nature of the problem. The first author also 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).

References

S. Ben-David, B. Chor, O. Goldreich, and M. Luby. 1992. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193—-219.

V. D. Blondel and V. Canterini. 2003. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computer Systems*, 36(3):231–245.

F. Casacuberta and C. de la Higuera. 1999. Optimal linguistic decoding is a difficult computational problem. *Pattern Recognition Letters*, 20(8):813–821.

F. Casacuberta and C. de la Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In *Proceedings of* ICGI *2000*, volume 1891 of LNAI, pages 15–24. Springer-Verlag.

F. Casacuberta. 1995. Probabilistic estimation of stochastic regular syntax-directed translation schemes. In R. Moreno, editor, *VI Spanish Symposium on Pattern Recognition and Image Analysis*, pages 201–297. AERFAI.

C. Cortes, M. Mohri, and A. Rastogi. 2006. On the computation of some standard distances between probabilistic automata. In *Proceedings of* CIAA *2006*, volume 4094 of LNCS, pages 137–149. Springer-Verlag.

C. Cortes, M. Mohri, and A. Rastogi. 2007. l_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(4):761–779.

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

M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability*. Freeman.

Joshua T. Goodman. 1998. *Parsing Inside–Out*. Ph.D. thesis, Harvard University.

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

L. Levin. 1986. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286.

R. B. Lyngsø and C. N. S. Pedersen. 2002. The consensus string problem and the complexity of comparing hidden markov models. *Journal of Computing and System Science*, 65(3):545–569.

M. Mohri, F. C. N. Pereira, and M. Riley. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32.

M. Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(3):269–311.

A. Paz. 1971. *Introduction to probabilistic automata*. Academic Press, New York.

D. Picó and F. Casacuberta. 2001. Some statistical-estimation methods for stochastic finite-state transducers. *Machine Learning Journal*, 44(1):121–141.

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

K. Sima'an. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In COLING, pages 1175–1180.

A. Stolcke. 1995. An efficient probablistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. 2005. Probabilistic finite state automata – part I and II. *Pattern Analysis and Machine Intelligence*, 27(7):1013–1039.