Computing the Most Probable String with a Probabilistic Finite State Machine

Colin de la Higuera
Université de Nantes,
CNRS, LINA, UMR6241,
F-44000, France
`cdlh@univ-nantes.fr`

Jose Oncina
Dep. de Lenguajes y Sistemas Informáticos,
Universidad de Alicante,
Alicante, Spain
`oncina@ua.es`

Abstract

The problem of finding the consensus / most probable string for a distribution generated by a probabilistic finite automaton or a hidden Markov model arises in a number of natural language processing tasks: it has to be solved in several transducer related tasks like optimal decoding in speech, or finding the most probable translation of an input sentence. We provide an algorithm which solves these problems in time polynomial in the inverse of the probability of the most probable string, which in practise makes the computation tractable in many cases. We also show that this exact computation compares favourably with the traditional Viterbi computation.

1 Introduction

Probabilistic finite state machines are used to define distributions over sets of strings, to model languages, help with decoding or for translation tasks. These machines come under various names, with different characteristics: probabilistic (generating) finite state automata, weighted machines, hidden Markov models (HMMs) or finite state transducers...

An important and common problem in all the settings is that of computing the most probable event generated by the machine, possibly under a constraint over the input string or the length. The typical way of handling this question is by using the Viterbi algorithm, which extracts the most probable path/parse given the requirements.

If in certain cases finding the most probable parse is what is seeked, in others this is computed under the generally accepted belief that the computation of the most probable string, also called the consensus string, is untractable and that the Viterbi score is an acceptable approximation. But the probability of the string is obtained by summing over the different parses, so there is no strong reason that the string with the most probable parse is also the most probable one.

The problem of finding the most probable string was addressed by a number of authors, in computational linguistics, pattern recognition and bio-informatics [Sima'an, 2002, Goodman, 1998, Casacuberta and de la Higuera, 1999, 2000, Lyngsø and Pedersen, 2002]: the problem was proved to be $\mathcal{NP}$-hard; the associated decision problem is $\mathcal{NP}$-complete in limited cases only, because the most probable string can be exponentially long in the number of states of the finite state machine (a construction can be found in [de la Higuera and Oncina, 2013]). As a corollary, finding the most probable translation (or decoding) of some input string, when given a finite state transducer, is intractable: the set of possible transductions, with their conditional probabilities can be represented as a PFA.

Manning and Schütze [1999] argue that the Viterbi algorithm does not allow to solve the decoding problem in cases where there is not a one-to-one relationship between derivations and parses. In automatic translation Koehn [2010] proposes to compute the top n translations from word graphs, which is possible when these are deterministic. But when they are not, an alternative in statistical machine translation is to approximate these thanks to the Viterbi algorithm [Casacuberta and Vidal, 2004]. In speech recognition, the optimal decoding problem consists in finding the most probable sequence of utterances. Again, if the model is non-deterministic, this will usually be achieved by computing the most probable path instead.

In the before mentioned results the weight of each individual transition is between 0 and 1 and the score can be interpreted as a probability. An interesting variant, in the framework of multiplicity automata or of *accepting* probabilistic finite automata (also called Rabin automata), is the question, known as the *cut-point emptiness* problem, of the existence of a string whose weight is above a specific threshold; this problem is known to be undecidable [Blondel and Canterini, 2003].

In a recent analysis, de la Higuera and Oncina [2013] solved an associated decision problem: is there a string whose probability is above a given threshold? The condition required is that we are given an upper bound to the length of the most probable string and a lower bound to its probability. These encouraging results do not provide the means to actually compute the consensus string.

In this paper we provide three main results. The first (Section 3) relates the probability of a string with its length; as a corollary, given any fraction p, either all strings have probability less than p, or there is a string whose probability is at least p and is of length at most $\frac{(n+1)^2}{p}$ where n is the number of states of the corresponding PFA. The second result (Section 4) is an algorithm that can effectively compute the consensus string in time polynomial in the inverse of the probability of this string. Our third result (Section 5) is experimental: we show that our algorithm works well, and also that in highly ambiguous settings, the traditional approach, in which the Viterbi score is used to return the string with the most probable parse, will return sub-optimal results.

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 and by $\Sigma^{\leq n}$ the set of those of length at most n. When decomposing a string into sub-strings, 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 $\Sigma^\star$, $Pr_{\mathcal{D}}(L) = \sum_{x\in L} Pr_{\mathcal{D}}(x)$.

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 by $Pr_{\mathcal{M}}(x)$. 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 [Paz, 1971, Vidal et al., 2005]. In the sequel we will use λ-free PFA: these do not have empty (λ) transitions: this restriction is without loss of generality, as algorithms [Mohri, 2002, de la Higuera, 2010] exist allowing to transform, in polynomial time, more general PFA into PFA respecting the following definition:

Definition 1. *A λ-free* Probabilistic Finite Automaton (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 the complete transition function; $\delta(q,a,q') = 0$ 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 $\forall q \in Q$,

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

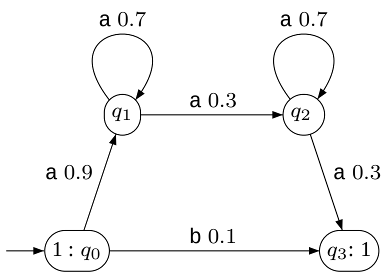

Figure 1: Graphical representation of a PFA.

An example of a PFA is shown in Fig. 1.

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

The probability of the path π is defined as $Pr_{\mathcal{M}}(\pi) = S(q_{i_0}) \cdot \prod_{j \in [n]} \delta(q_{i_{j-1}}, a_j, q_{i_j}) \cdot F(q_{i_n})$ and the probability of the string x is obtained by summing over the probabilities of all the paths in $\Pi_{\mathcal{A}}(x)$. An effective computation can be done by means of the Forward (or Backward) algorithm [Vidal et al., 2005].

We denote by $|\mathcal{A}|$ the size of $\mathcal{A}$ as one more than the number of its states. Therefore, for $\mathcal{A}$ as represented in Figure 1, we have $|\mathcal{A}| = 5$.

We do not recall here the definitions for hidden Markov models. It is known that one can transform an HMM into a PFA and vice-versa in polynomial time [Vidal et al., 2005].

2.3 The problem

The goal is to find the *most probable string* in a probabilistic language. This string is also named the *consensus* string.

Name: Consensus string (CS)
Instance: A probabilistic machine $\mathcal{M}$
Question: Find $x \in \Sigma^\star$ 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 aaaaa. Note that the string having the most probable single parse is b.

2.4 Associated decision problem

In [de la Higuera and Oncina, 2013] the following decision problem is studied:

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

BMPS is known to be $\mathcal{NP}$-hard [Casacuberta and de la Higuera, 2000]. De la Higuera and Oncina [2013] present a construction proving that the most probable string can be of exponential length: this makes the bound issue crucial in order to hope to solve CS. The proposed algorithm takes p and b as arguments and solves BMPS in time complexity $O(\frac{b|\Sigma|\cdot|Q|^2}{p})$. It is assumed that all arithmetic operations are in constant time.

The construction relies on the following simple properties, which ensure that only a reasonable amount of incomparable prefixes have to be scrutinized.

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

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

3 Probable strings are short

Is there a relation between the lengths of the strings and their probabilities? In other words, can we show that a string of probability at least p must be reasonably short? If so, the b parameter is not required: one can compute the consensus string without having to guess the bound b. Let us prove the following:

Proposition 1. *Let $\mathcal{A}$ be a λ-free* PFA *with n states and w a string.*
Then $|w| \leq \frac{(n+1)^2}{Pr_{\mathcal{A}}(w)}$.

As a corollary,

Corollary 1. *Let $\mathcal{A}$ be a λ-free* PFA *with n states. If there is a string with probability at least p, its length is at most $b = \frac{(n+1)^2}{p}$.*

Proof. Let w be a string of length len and $Pr_{\mathcal{A}}(w) = p$. A path is a sequence $\pi = q_{i_0} a_1 q_{i_1} a_2 \dots a_{\texttt{len}} q_{i_{\texttt{len}}}$, with $a_i \in \Sigma$.

Let $\Pi^j_{\mathcal{A}}(w)$ be the subset of $\Pi_{\mathcal{A}}(w)$ of all paths π for which state q_j is the most used state in π.

If, for some path π, there are several values j such that q_j is the most used state in π, we arbitrarily add π to the $\Pi^j_{\mathcal{A}}(w)$ which has the smallest index j.

Then, because of a typical combinatorial argument, there exists at least one j such that

$Pr_{\mathcal{A}}(\Pi^j_{\mathcal{A}}(w)) \geq \frac{p}{n}$. Note that in any path in $\Pi^j_{\mathcal{A}}(w)$ state q_j appears at least $\frac{\texttt{len}+1}{n}$ times. Consider any of these paths π in $\Pi^j_{\mathcal{A}}(w)$. Let k be the smallest integer such that $q_{i_k} = q_j$ (ie the first time we visit state q_j in path π is after having read the first k characters of w). Then for each value k' such that $q_{i_{k'}} = q_j$, we can shorten the path π by removing the cycle between q_{i_k} and $q_{i_{k'}}$ and obtain in each case a path for a new string, and the probability of this path is at least that of π.

We have therefore at least $\frac{\texttt{len}+1}{n} - 1$ such alternative paths for π.

We call Alt(π, j) the set of alternative paths for π and q_j. Hence $|\text{Alt}(\pi, j)| \geq \frac{\texttt{len}+1}{n} - 1$.

And therefore

$$Pr_{\mathcal{A}}(\text{Alt}(\pi, j)) \geq \left(\frac{\texttt{len}+1}{n} - 1\right) Pr_{\mathcal{A}}(\pi).$$

We now want to sum this quantity over the different π in $\Pi^j_{\mathcal{A}}(w)$. Note that there could be a difficulty with the fact that two different paths may share an identical alternative that would be counted twice. The following lemma (proved later) tells us that this is not a problem.

Lemma 1. *Let π and π' be two different paths in $\Pi^j_{\mathcal{A}}(w)$, and π'' be a path belonging both to* Alt(π, j) *and to* Alt(π', j). *Then $Pr_{\mathcal{A}}(\pi'') \geq Pr_{\mathcal{A}}(\pi) + Pr_{\mathcal{A}}(\pi')$.*

Therefore, we can sum and

$$\sum_{\pi \in \Pi^j_{\mathcal{A}}(w)} Pr_{\mathcal{A}}(\text{Alt}(\pi, j)) \geq \left(\frac{\texttt{len}+1}{n} - 1\right) \frac{p}{n}.$$

The left hand side represents a mass of probabilities distributed by $\mathcal{A}$ to other strings than w. Summing with the probability of w, we obtain:

$$\begin{aligned}
\left(\frac{\texttt{len}+1}{n} - 1\right) \cdot \frac{p}{n} + p &\leq 1 \\
(\texttt{len}+1-n) \cdot p + pn^2 &\leq n^2 \\
(\texttt{len}+1-n) \leq \frac{n^2(1-p)}{p} & \\
\texttt{len} \leq \frac{n^2(1-p)}{p} + n - 1 &
\end{aligned}$$

It follows that $\texttt{len} \leq \frac{(n+1)^2}{p}$.

□

Proof of the lemma. $\pi'' = \pi_j$ and $\pi'' = \pi'_{j'}$. Necessarily we have $j = j'$.

Now $q_{i^1_k} w_k q_{i^1_{k+1}} w_{k+1} \ldots w_{k+t} q_{i^1_{k+t}}$ and $q_{i^2_k} w_k q_{i^2_{k+1}} w_{k+1} \ldots w_{k+t} q_{i^2_{k+t}}$ are the two fragments of the paths that have been removed from π and π'. These are necessarily different, but might coincide in part. Let h be the first index for which they are different, ie $\forall z < h, q_{i^1_z} = q_{i^2_z}$ and $q_{i^1_h} \neq q_{i^2_h}$.

We have:

$P(q_{i^1_{h-1}}, w_h, q_{i^1_h}) + P(q_{i^2_{h-1}}, w_h, q_{i^2_h}) \leq 1$ and the result follows.

□

We use Proposition 1 to associate with a given string w an upper bound over the probability of any string having w as a prefix:

Definition 2. *The* Potential Probability $\mathcal{PP}$ *of a prefix string w is*

$$\mathcal{PP}(w) = \min(Pr_{\mathcal{A}}(w\,\Sigma^\star), \frac{|\mathcal{A}|^2}{|w|})$$

$\mathcal{PP}(w)$ is also an upper bound on the probability of any string having w as a prefix:

Property 3. $\forall u \in \Sigma^\star \;\; Pr_{\mathcal{A}}(wu) \leq \mathcal{PP}(w)$

Indeed, $Pr_{\mathcal{A}}(wu) \leq Pr_{\mathcal{A}}(w\,\Sigma^\star)$ and, because of Proposition 1, $Pr_{\mathcal{A}}(wu) \leq \frac{|\mathcal{A}|^2}{|wu|} \leq \frac{|\mathcal{A}|^2}{|w|}$.

This means that we can decide, for a given prefix w, and a probability to be improved, if w is *viable*, ie if the best string having w as a prefix can be better than the proposed probability. Furtermore, given a PFA $\mathcal{A}$ and a prefix w, computing $\mathcal{PP}(w)$ is simple.

4 Solving the consensus string problem

Algorithm 1 is given a PFA $\mathcal{A}$ and returns the most probable string. The bounds obtained in the previous section allow us to explore the viable prefixes, check if the corresponding string improves our current candidate, add an extra letter to the viable prefix and add this new prefix to a priority queue.

The priority queue (**Q**) is a structure in which the time complexity for insertion is $O(\log |\mathbf{Q}|)$ and the extraction (**Pop**) of the first element can take place in constant time. In this case the order for the queue will depend on the value $\mathcal{PP}(w)$ of the prefix w.

```
Data: a PFA $\mathcal{A}$
Result: $w$, the most probable string
1  $Current\_Prob = 0$;
2  $\mathbf{Q} = [\lambda]$;
3  $Continue =$ true;
4  while not(Empty(Q))and $Continue$ do
5      $w =$ Pop(Q);
6      if $\mathcal{PP}(w) > Current\_Prob$ then
7          $p = Pr_{\mathcal{A}}(w)$;
8          if $p > Current\_Prob$ then
9              $Current\_Prob = p$;
10             $Current\_Best = w$;
11         foreach $a \in \Sigma$ do
12             if $\mathcal{PP}(wa) > Current\_Prob$ then
13                 Insert($wa, \mathcal{PP}(wa)$, Q)
14     else
15         $Continue =$ false;
16 return $Current\_Best$
```
Algorithm 1: Finding the Consensus String

Analysis: Let p_{opt} be the probability of the consensus string. Let **Viable** be the set of all strings w such that $\mathcal{PP}(w) \geq p_{\text{opt}}$.

- **Fact 1.** Let w be the first element of **Q** at some iteration. If $\mathcal{PP}(w) < p_{\text{opt}}$, every other element of **Q** will also have smaller $\mathcal{PP}$ and the algorithm will halt. It follows that until the consensus string is found, the first element w is in **Viable**.

- **Fact 2.** If $w \in$**Viable**, $\mathcal{PP}(w) \geq p_{\text{opt}}$, therefore $\frac{|\mathcal{A}|^2}{|w|} \geq p_{\text{opt}}$ so $|w| \leq \frac{|\mathcal{A}|^2}{p_{\text{opt}}}$.

- **Fact 3.** There are at most $\frac{1}{p_{\text{opt}}}$ pairwise incomparable prefixes in **Viable**. Indeed, all elements of **Viable** have $\mathcal{PP}(w) = \min(Pr_{\mathcal{A}}(w\,\Sigma^\star), \frac{|\mathcal{A}|^2}{|w|}) \geq p_{\text{opt}}$ so also have $Pr_{\mathcal{A}}(w\,\Sigma^\star) \geq p_{\text{opt}}$ and by Property 2 we are done.

- **Fact 4.** There are at most $\frac{1}{p_{\text{opt}}} \cdot \frac{|\mathcal{A}|^2}{p_{\text{opt}}}$ different prefixes in **Viable**, as a consequence of facts 2 and 3.

- **Fact 5.** At every iteration of the main loop at most $|\Sigma|$ new elements are added to the priority queue.

- **Fact 6.** Therefore, since only the first elements of the priority queue will cause (at most $|\Sigma|$) insertions, and these (fact 1) are necesarily viable, the total number of insertions is bounded by $|\Sigma| \cdot \frac{|\mathcal{A}|^2}{p_{\text{opt}}} \cdot \frac{1}{p_{\text{opt}}}$.

The time complexity of the algorithm is proportional to the number of insertions in the queue and is computed as follows:

- $|\mathbf{Q}\ |$ is at most $\frac{|\Sigma|\cdot|\mathcal{A}|^2}{p_{\text{opt}}^2}$;

- Insertion of an element into **Q** is in $O\left(\log\left(\frac{|\Sigma|\cdot|\mathcal{A}|^2}{p_{\text{opt}}^2}\right)\right)$.

5 Experiments

From the theoretical analysis it appears that the new algorithm will be able to compute the consensus string. The goal of the experiments is therefore to show how the algorithm scales up: there are domains in natural language processing where the probabilities are very small, and the alphabets very large. In others this is not the case. How well does the algorithm adapt to small probabilities? A second line of experiments consists in measuring the quality of the most probable parse for the consensus string. This could obviously not be measured up to now (because of the lack of an algorithm for the most probable string). Finding out how far (both in value and in rank) the string returned by the Viterbi algorithm is from the consensus string is of interest.

5.1 Further experiments

A more extensive experimentation may consist in building a collection of random PFA, in the line of what has been done in HMM/PFA learning competitions, for instance [Verwer et al., 2012]. A connected graph is built, the arcs are transformed into transitions by adding labels and weights. A normalisation phase takes place so as to end up with a distribution of probabilities.

The main drawback of this procedure is that, most often, the consensus string will end up by being the empty string (or a very short string) and any algorithm will find it easily.

Actually, the same will happen when testing on more realistic data: a language model built from n-grams will often have as most probable string the empty string.

An extensive experimentation should also compare this algorithm with alternative techniques which have been introduced:

A first extension of the Viterbi approximation is called *crunching* [May and Knight, 2006]: instead of just computing for a string the probability of the best path, with a bit more effort, the value associated to a string is the sum of the probabilities of the n best paths.

Another approach is *variational decoding* [Li et al., 2009]: in this method and alternatives like Minimum Risk Decoding, a best approximation by n-grams of the distribution is used, and the most probable string is taken as the one which maximizes the probability with respect to this approximation. These techniques are shown to give better results than the Viterbi decoding, but are not able to cope with long distance dependencies.

Coarse-to-fine parsing [Charniak et al., 2006] is a strategy that reduces the complexity of the search involved in finding the best parse. It defines a sequence of increasingly more complex Probabilistic Context-Free grammars (PCFG), and uses the parse forest produced by one PCFG to prune the search of the next more complex PCFG.

5.2 A tighter bound on the complexity

The complexity of the algorithm can be measured by counting the number of insertions into the priority queue. This number has been upper-bounded by $|\Sigma| \cdot \frac{|\mathcal{A}|^2}{p_{\text{opt}}^2}$. But it is of interest to get a better and tighter bound. In order to have a difficult set of test PFAs we built a family of models for which the consensus string has a parametrized length and where an exponentially large set of strings has the same length and slightly lower probabilities than the consensus string.

In order to achieve that, the states are organised in levels, and at each level there is a fixed number of states (multiplicity). One of the states of the first level is chosen as initial state. All the states have an ending probability of zero except for one unique state of the last level; there is a transition from each state of level n to all states of level $n+1$ but also to all the states of level $k \leq n$.

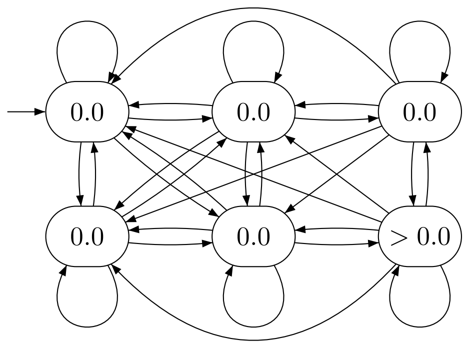

Figure 2: The topology of an automaton with 3 levels and multiplicity 2

An example of this structure, with 3 levels and multiplicity 2 is represented in Fig. 2. The shortest string with non-null probability will be of length $n-1$ where n is the number of levels.

A collection of random PFA was built with vocabulary size varying from 2 to 6, number of levels from 3 to 5, and the multiplicity from 2 to 3. 16 different PFA were generated for each vocabulary size, number of levels and multiplicity. Therefore, a total of 480 automata were built. From those 16 were discarded because the low probability of the consensus string made it impossible to deal with the size of the priority queue. 464 automata were therefore considered for the experiments.

In the first set of experiments the goal was to test how strong is the theoretical bound on the number of insertions in the priority queue.

Fig. 3 plots the inverse of the probability of the consensus string versus the number of insertions in the priority queue. The bound from Section 4 is

$$|\mathbf{Q}| \leq \frac{|\Sigma| \cdot |\mathcal{A}|^2}{p_{\text{opt}}^2}$$

Our data indicates that

$$|\mathbf{Q}| \leq \frac{1}{p_{\text{opt}}^2} \leq \frac{|\Sigma| \cdot |\mathcal{A}|^2}{p_{\text{opt}}^2}$$

Furthermore, $\frac{2}{p_{\text{opt}}}$ seems empirically to be a better bound: a more detailed mathematical analysis could lead to prove this.

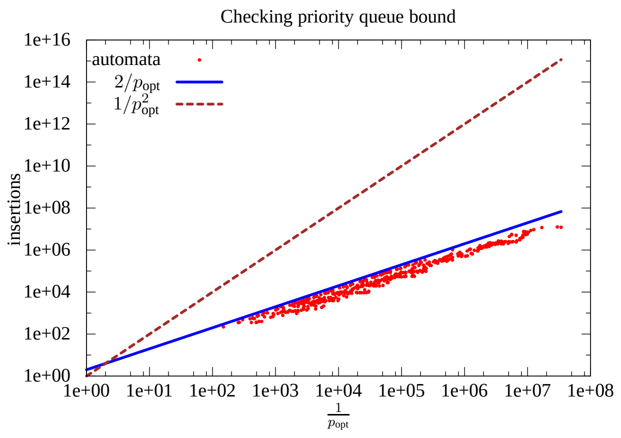

Figure 3: Number of insertions made in the priority queue

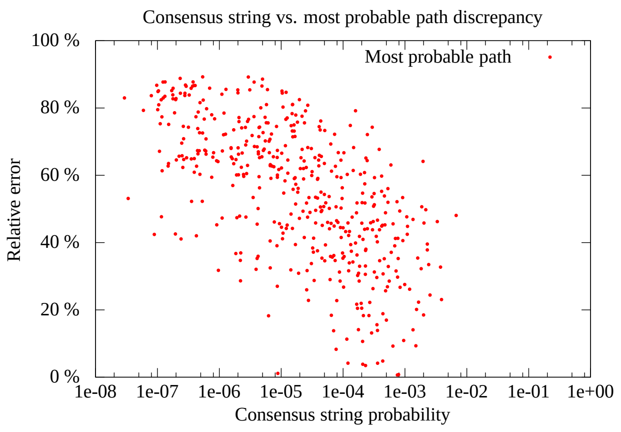

Figure 4: Most probable path accuracy

5.3 How good is the Viterbi approximation?

In the literature the string with the most probable path is often used as an approximation to the consensus string. Using the same automata, we attempted to measure the distance between the probability of the string returned by the Viterbi algorithm, which computes the most probable parse, and the probability of the consensus string.

For each automaton, we have measured the probability of the most probable string (p_{opt}) and the probability of the most probable path (p_p).

In 63% of the 464 PFA the consensus string and the string with the most probable path were different, and in no case does the probability of the consensus string coincide with the probability of the most probable path.

Figure 4 shows the relative error when using the probability of the most probable path instead of the probability of the consensus string. In other words we measure and plot $\frac{p_{\text{opt}}-p_p}{p_{\text{opt}}}$. It can be observed that the relative error can be very large and increases as the probability of the consensus string decreases. Furthermore, we ran an experiment to compute the rank of the string with most probable path, when ordered by the actual probability. Over the proposed benchmark, the average rank is 9.2 and the maximum is 277.

6 Conclusion

The algorithm provided in this work allows to compute the most probable string with its exact probability. Experimentally it works well in settings where the number of paths involved in the sums leading to the computation of the probability is large: in artificial experiments, it allowed to show that the best string for the Viterbi score could be outranked by more that 10 alternative strings.

Further experiments in natural language processing tasks are still required in order to understand in which particular settings the algorithm can be of use. In preliminary language modelling tasks two difficulties arose: the most probable string is going to be extremely short and of little interest. Furthermore, language models use very large alphabets, so the most probable string of length 10 will typically have a probability of the order of 10^{-30}. In our experiments the algorithm was capable of dealing with figures of the order of 10^{-6}. But the difference is clearly impossible to deal with.

The proposed algorithm can be used in cases where the number of possible translations and paths may be very large, but where at least one string (or translation) has a reasonable probability. Other future research directions concern in lifting the λ-freeness condition by taking into acount λ-transitions on the fly: in many cases, the transformation is cumbersome. Extending this work to probabilistic context-free grammars is also an issue.

Acknowledgement

Discussions with Khalil Sima'an proved important in a previous part of this work, as well as later help from Thomas Hanneforth. We also wish to thank the anonymous reviewer who pointed out a certain number of alternative heuristics that should be compared more extensively.

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).

References

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

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

F. Casacuberta and C. de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In A. L. de Oliveira, editor, *Grammatical Inference: Algorithms and Applications, Proceedings of* ICGI *'00*, volume 1891 of LNAI, pages 15–24. Springer-Verlag, 2000.

F. Casacuberta and E. Vidal. Machine translation with inferred stochastic finite-state transducers. *Computational Linguistics*, 30(2):205–225, 2004.

E. Charniak, M. Johnson, M. Elsner, J. L. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, and T. Vu. Multilevel coarse-to-fine PCFG parsing. In *Proceedings of* HLT-NAACL *2006*. The Association for Computer Linguistics, 2006.

C. de la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

C. de la Higuera and J. Oncina. The most probable string: an algorithmic study. *Journal of Logic and Computation, doi: 10.1093/logcom/exs049*, 2013.

J. T. Goodman. *Parsing Inside–Out*. PhD thesis, Harvard University, 1998.

P. Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

Z. Li, J. Eisner, and S. Khudanpur. Variational decoding for statistical machine translation. In *Proceedings of* ACL/IJCNLP *2009*, pages 593–601. The Association for Computer Linguistics, 2009.

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(3):545–569, 2002.

C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

J. May and K. Knight. A better n-best list: Practical determinization of weighted finite tree automata. In *Proceedings of* HLT-NAACL *2006*. The Association for Computer Linguistics, 2006.

M. Mohri. Generic e-removal and input e-normalization algorithms for weighted transducers. *International Journal on Foundations of Computer Science*, 13(1):129–143, 2002.

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

K. Sima'an. Computational complexity of probabilistic disambiguation: NP-completeness results for language and speech processing. *Grammars*, 5(2):125–151, 2002.

S. Verwer, R. Eyraud, and C. de la Higuera. Results of the pautomac probabilistic automaton learning competition. In *Journal of Machine Learning Research - Proceedings Track*, volume 21, pages 243–248, 2012.

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(7):1013–1039, 2005.