# Computation of Infix Probabilities for Probabilistic Context-Free Grammars

**Mark-Jan Nederhof**
School of Computer Science
University of St Andrews
United Kingdom
markjan.nederhof@gmail.com

**Giorgio Satta**
Dept. of Information Engineering
University of Padua
Italy
satta@dei.unipd.it

**Abstract**

The notion of infix probability has been introduced in the literature as a generalization of the notion of prefix (or initial substring) probability, motivated by applications in speech recognition and word error correction. For the case where a probabilistic context-free grammar is used as language model, methods for the computation of infix probabilities have been presented in the literature, based on various simplifying assumptions. Here we present a solution that applies to the problem in its full generality.

## 1 Introduction

Probabilistic context-free grammars (PCFGs for short) are a statistical model widely used in natural language processing. Several computational problems related to PCFGs have been investigated in the literature, motivated by applications in modeling of natural language syntax. One such problem is the computation of **prefix probabilities** for PCFGs, where we are given as input a PCFG $\mathcal{G}$ and a string $w$, and we are asked to compute the probability that a sentence generated by $\mathcal{G}$ starts with $w$, that is, has $w$ as a prefix. This quantity is defined as the possibly infinite sum of the probabilities of all strings of the form $wx$, for any string $x$ over the alphabet of $\mathcal{G}$.

The problem of computation of prefix probabilities for PCFGs was first formulated by Persoon and Fu (1975). Efficient algorithms for its solution have been proposed by Jelinek and Lafferty (1991) and Stolcke (1995). Prefix probabilities can be used to compute probability distributions for the next word or part-of-speech, when a prefix of the input has already been processed, as discussed by Jelinek and Lafferty (1991). Such distributions are useful for speech recognition, where the result of the acoustic processor is represented as a lattice, and local choices must be made for a next transition. In addition, distributions for the next word are also useful for applications of word error correction, when one is processing 'noisy' text and the parser recognizes an error that must be recovered by operations of insertion, replacement or deletion.

Motivated by the above applications, the problem of the computation of **infix probabilities** for PCFGs has been introduced in the literature as a generalization of the prefix probability problem. We are now given a PCFG $\mathcal{G}$ and a string $w$, and we are asked to compute the probability that a sentence generated by $\mathcal{G}$ has $w$ as an infix. This probability is defined as the possibly infinite sum of the probabilities of all strings of the form $xwy$, for any pair of strings $x$ and $y$ over the alphabet of $\mathcal{G}$. Besides applications in computation of the probability distribution for the next word token and in word error correction, infix probabilities can also be exploited in speech understanding systems to score partial hypotheses in algorithms based on beam search, as discussed by Corazza et al. (1991).

Corazza et al. (1991) have pointed out that the computation of infix probabilities is more difficult than the computation of prefix probabilities, due to the added ambiguity that several occurrences of the given infix can be found in a single string generated by the PCFG. The authors developed solutions for the case where some distribution can be defined on the distance of the infix from the sentence boundaries, which is a simplifying assumption. The problem is also considered by Fred (2000), which provides algorithms for the case where the language model is a probabilistic regular grammar. However, the algorithm in (Fred, 2000) does not apply to cases with multiple occurrences of the given infix within a string in the language, which is what was pointed out to be the problematic case.

In this paper we adopt a novel approach to the problem of computation of infix probabilities, by removing the ambiguity that would be caused by multiple occurrences of the given infix. Although our result is obtained by a combination of well-known techniques from the literature on PCFG parsing and pattern matching, as far as we know this is the first algorithm for the computation of infix probabilities that works for general PCFG models without any restrictive assumption.

The remainder of this paper is structured as follows. In Section 2 we explain how the sum of the probabilities of all trees generated by a PCFG can be computed as the least fixed-point solution of a non-linear system of equations. In Section 3 we recall the construction of a new PCFG out of a given PCFG and a given finite automaton, such that the language generated by the new grammar is the intersection of the languages generated by the given PCFG and the automaton, and the probabilities of the generated strings are preserved. In Section 4 we show how one can efficiently construct an unambiguous finite automaton that accepts all strings with a given infix. The material from these three sections is combined into a new algorithm in Section 5, which allows computation of the infix probability for PCFGs. This is the main result of this paper. Several extensions of the basic technique are discussed in Section 6. Section 7 discusses implementation and some experiments.

## 2 Sum of probabilities of all derivations

Assume a probabilistic context-free grammar $\mathcal{G}$, represented by a 5-tuple $(\Sigma, N, S, R, p)$, where $\Sigma$ and $N$ are two finite disjoint sets of **terminals** and **nonterminals**, respectively, $S \in N$ is the **start symbol**, $R$ is a finite set of **rules**, each of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\Sigma \cup N)^*$, and $p$ is a function from rules in $R$ to real numbers in the interval $[0, 1]$.

The concept of **left-most derivation** in one step is represented by the notation $\alpha \overset{\pi}{\Rightarrow}_{\mathcal{G}} \beta$, which means that the left-most occurrence of any nonterminal in $\alpha \in (\Sigma \cup N)^*$ is rewritten by means of some rule $\pi \in R$. If the rewritten nonterminal is $A$, then $\pi$ must be of the form $(A \to \gamma)$ and $\beta$ is the result of replacing the occurrence of $A$ in $\alpha$ by $\gamma$. A left-most derivation with any number of steps, using a sequence $d$ of rules, is denoted as $\alpha \overset{d}{\Rightarrow}_{\mathcal{G}} \beta$. We omit the subscript $\mathcal{G}$ when the PCFG is understood. We also write $\alpha \overset{*}{\Rightarrow} \beta$ when the involved sequence of rules is of no relevance. Henceforth, all derivations we discuss are implicitly left-most.

A **complete** derivation is either the empty sequence of rules, or a sequence $d = \pi_1 \cdots \pi_m$, $m \geq 1$, of rules such that $A \overset{d}{\Rightarrow} w$ for some $A \in N$ and $w \in \Sigma^*$. In the latter case, we say the complete derivation starts with $A$, and in the former case, with $d$ an empty sequence of rules, we assume the complete derivation starts and ends with a single terminal, which is left unspecified. It is well-known that there exists a bijective correspondence between left-most complete derivations starting with nonterminal $A$ and parse trees derived by the grammar with root $A$ and a yield composed of terminal symbols only.

The **depth** of a complete derivation $d$ is the length of the longest path from the root to a leaf in the parse tree associated with $d$. The length of a path is defined as the number of nodes it visits. Thus if $d = \pi$ for some rule $\pi = (A \to w)$ with $w \in \Sigma^*$, then the depth of $d$ is 2.

The probability $p(d)$ of a complete derivation $d = \pi_1 \cdots \pi_m$, $m \geq 1$, is:

$$p(d) \quad = \quad \prod_{i=1}^{m} p(\pi_i).$$

We also assume that $p(d) = 1$ when $d$ is an empty sequence of rules. The probability $p(w)$ of a string $w$ is the sum of all complete derivations deriving that string from the start symbol:

$$p(w) \quad = \quad \sum_{d:\, S \overset{d}{\Rightarrow} w} p(d).$$

With this notation, **consistency** of a PCFG is defined as the condition:

$$\sum_{d,w:\ S \overset{d}{\Rightarrow} w} p(d) \quad = \quad 1.$$

In other words, a PCFG is consistent if the sum of probabilities of all complete derivations starting with $S$ is 1. An equivalent definition of consistency considers the sum of probabilities of all strings:

$$\sum_{w} p(w) \quad = \quad 1.$$

See (Booth and Thompson, 1973) for further discussion.

In practice, PCFGs are often required to satisfy the additional condition:

$$\sum_{\pi=(A \rightarrow \alpha)} p(\pi) \quad = \quad 1,$$

for each $A \in N$. This condition is called **properness**. PCFGs that naturally arise by parameter estimation from corpora are generally consistent; see (Sánchez and Benedí, 1997; Chi and Geman, 1998). However, in what follows, neither properness nor consistency is guaranteed.

We define the **partition function** of $\mathcal{G}$ as the function $Z$ that assigns to each $A \in N$ the value

$$Z(A) \quad = \quad \sum_{d,w} p(A \overset{d}{\Rightarrow} w). \tag{1}$$

Note that $Z(S) = 1$ means that $\mathcal{G}$ is consistent. More generally, in later sections we will need to compute the partition function for non-consistent PCFGs.

We can characterize the partition function of a PCFG as a solution of a specific system of equations. Following the approach in (Harris, 1963; Chi, 1999), we introduce generating functions associated with the nonterminals of the grammar. For $A \in N$ and $\alpha \in (N \cup \Sigma)^*$, we write $f(A, \alpha)$ to denote the number of occurrences of symbol $A$ within string $\alpha$. Let $N = \{A_1, A_2, \ldots, A_{|N|}\}$. For each $A_k \in N$, let $m_k$ be the number of rules in $R$ with left-hand side $A_k$, and assume some fixed order for these rules. For each $i$ with $1 \leq i \leq m_k$, let $A_k \rightarrow \alpha_{k,i}$ be the $i$-th rule with left-hand side $A_k$.

For each $k$ with $1 \leq k \leq |N|$, the **generating function** associated with $A_k$ is defined as

$$g_{A_k}(z_1, z_2, \ldots, z_{|N|}) = \sum_{i=1}^{m_k} \Big( p(A_k \rightarrow \alpha_{k,i}) \cdot \prod_{j=1}^{|N|} z_j^{f(A_j, \alpha_{k,i})} \Big). \tag{2}$$

Furthermore, for each $i \geq 1$ we recursively define functions $g_{A_k}^{(i)}(z_1, z_2, \ldots, z_{|N|})$ by

$$g_{A_k}^{(1)}(z_1, z_2, \ldots, z_{|N|}) = g_{A_k}(z_1, z_2, \ldots, z_{|N|}), \tag{3}$$

and, for $i \geq 2$, by

$$\begin{aligned} g_{A_k}^{(i)}(z_1, z_2, \ldots, z_{|N|}) = \\ g_{A_k}(\ & g_{A_1}^{(i-1)}(z_1, z_2, \ldots, z_{|N|}), \\ & g_{A_2}^{(i-1)}(z_1, z_2, \ldots, z_{|N|}), \ldots, \\ & g_{A_{|N|}}^{(i-1)}(z_1, z_2, \ldots, z_{|N|})\ ). \end{aligned} \tag{4}$$

Using induction it is not difficult to show that, for each $k$ and $i$ as above, $g_{A_k}^{(i)}(0, 0, \ldots, 0)$ is the sum of the probabilities of all complete derivations from $A_k$ having depth not exceeding $i$. This implies that, for $i = 0, 1, 2, \ldots$, the sequence of the $g_{A_k}^{(i)}(0, 0, \ldots, 0)$ monotonically converges to $Z(A_k)$.

For each $k$ with $1 \leq k \leq |N|$ we can now write

$$\begin{aligned} & Z(A_k) = \\ & = \lim_{i \rightarrow \infty} g_{A_k}^{(i)}(0, \ldots, 0) \\ & = \lim_{i \rightarrow \infty} g_{A_k}(\ g_{A_1}^{(i-1)}(0, 0, \ldots, 0), \ldots, \\ & \qquad\qquad g_{A_{|N|}}^{(i-1)}(0, 0, \ldots, 0)\ ) \\ & = g_{A_k}(\ \lim_{i \rightarrow \infty} g_{A_1}^{(i-1)}(0, 0, \ldots, 0), \ldots, \\ & \qquad \lim_{i \rightarrow \infty} g_{A_{|N|}}^{(i-1)}(0, 0, \ldots, 0)\ ) \\ & = g_{A_k}(Z(A_1), \ldots, Z(A_{|N|})). \end{aligned}$$

The above shows that the values of the partition function provide a solution to the system of the following equations, for $1 \leq k \leq |N|$:

$$z_k \quad = \quad g_{A_k}(z_1, z_2, \ldots, z_{|N|}). \tag{5}$$

In the case of a general PCFG, the above equations are non-linear polynomials with positive (real) coefficients. We can represent the resulting system in vector form and write $\mathbf{X} = \mathbf{g}(\mathbf{X})$. These systems are called monotone systems of polynomial equations and have been investigated by Etessami and Yannakakis (2009) and Kiefer et al. (2007). The sought solution, that is, the partition function, is the least fixed point solution of $\mathbf{X} = \mathbf{g}(\mathbf{X})$.

For practical reasons, the set of nonterminals of a grammar is usually divided into maximal subsets of mutually recursive nonterminals, that is, for each $A$ and $B$ in such a subset, we have $A \overset{*}{\Rightarrow} uB\alpha$ and $B \overset{*}{\Rightarrow} vA\beta$, for some $u, v, \alpha, \beta$. This corresponds to a **strongly connected component** if we see the connection between the left-hand side of a rule and a nonterminal member in its right-hand side as an edge in a directed graph. For each strongly connected component, there is a separate system of equations of the form $\mathbf{X} = \mathbf{g}(\mathbf{X})$. Such systems can be solved one by one, in a bottom-up order. That is, if one strongly connected component contains nonterminal $A$, and another contains nonterminal $B$, where $A \overset{*}{\Rightarrow} uB\alpha$ for some $u, \alpha$, then the system for the latter component must be solved first.

The solution for a system of equations such as those described above can be irrational and non-expressible by radicals, even if we assume that all the probabilities of the rules in the input PCFG are rational numbers, as observed by Etessami and Yannakakis (2009). Nonetheless, the partition function can still be approximated to any degree of precision by iterative computation of the relation in (4), as done for instance by Stolcke (1995) and by Abney et al. (1999). This corresponds to the so-called fixed-point iteration method, which is well-known in the numerical calculus literature and is frequently applied to systems of non-linear equations because it can be easily implemented.

When a number of standard conditions are met, each iteration of (4) adds a fixed number of bits to the precision of the solution; see Kelley (1995, Chapter 4). Since each iteration can easily be implemented to run in polynomial time, this means that we can approximate the partition function of a PCFG in polynomial time in the size of the PCFG itself and in the number of bits of the desired precision.

In practical applications where large PCFGs are empirically estimated from data sets, the standard conditions mentioned above for the polynomial time approximation of the partition function are usually met. However, there are some degenerate cases for which these standard conditions do not hold, resulting in exponential time behaviour of the fixed-point iteration method. This has been firstly observed in (Etessami and Yannakakis, 2005).

An alternative iterative algorithm for the approximation of the partition function has been proposed by Etessami and Yannakakis (2009), based on Newton's method for the solution of non-linear systems of equations. From a theoretical perspective, Kiefer et al. (2007) have shown that, after a certain number of initial iterations, Newton's method adds a fixed number of bits to the precision of the approximated solution, even in the above mentioned cases in which the fixed-point iteration method shows exponential time behaviour. However, these authors also show that, in some degenerate cases, the number of iterations needed to compute the first bit of the solution can be at least exponential in the size of the system.

Experiments with Newton's method for the approximation of the partition functions of PCFGs have been carried out in several application-oriented settings, by Wojtczak and Etessami (2007) and by Nederhof and Satta (2008), showing considerable improvements over the fixed-point iteration method.

## 3 Intersection of PCFG and FA

It was shown by Bar-Hillel et al. (1964) that context-free languages are closed under intersection with regular languages. Their proof relied on the construction of a new CFG out of an input CFG and an input finite automaton. Here we extend that construction by letting the input grammar be a probabilistic CFG. We refer the reader to (Nederhof and Satta, 2003) for more details.

To avoid a number of technical complications, we assume the finite automaton has no epsilon transitions, and has only one final state. In the context of our use of this construction in the following sections, these restrictions are without loss of generality. Thus, a finite automaton (FA) $\mathcal{M}$ is represented by a 5-tuple $(\Sigma, Q, q_0, q_f, \Delta)$, where $\Sigma$ and $Q$ are two finite sets of **terminals** and **states**, respectively, $q_0$ is the **initial** state, $q_f$ is the **final** state, and $\Delta$ is a finite set of **transitions**, each of the form $s \overset{a}{\mapsto} t$, where $s, t \in Q$ and $a \in \Sigma$.

A **complete computation** of $\mathcal{M}$ accepting string

$w = a_1 \cdots a_n$ is a sequence $c = \tau_1 \cdots \tau_n$ of transitions such that $\tau_i = (s_{i-1} \overset{a_i}{\mapsto} s_i)$ for each $i$ ($1 \leq i \leq n$), for some $s_0, s_1, \ldots, s_n$ with $s_0 = q_0$ and $s_n = q_f$. The language of all strings accepted by $\mathcal{M}$ is denoted by $L(\mathcal{M})$. A FA is **unambiguous** if at most one complete computation exists for each accepted string. A FA is **deterministic** if there is at most one transition $s \overset{a}{\mapsto} t$ for each $s$ and $a$.

For a FA $\mathcal{M}$ as above and a PCFG $\mathcal{G} = (\Sigma, N, S, R, p)$ with the same set of terminals, we construct a new PCFG $\mathcal{G}' = (\Sigma, N', S', R', p')$, where $N' = Q \times (\Sigma \cup N) \times Q$, $S' = (q_0, S, q_f)$, and $R'$ is the set of rules that is obtained as follows.

- For each $A \rightarrow X_1 \cdots X_m$ in $R$ and each sequence $s_0, \ldots, s_m$ with $s_i \in Q$, $0 \leq i \leq m$, and $m \geq 0$, let $(s_0, A, s_m) \rightarrow (s_0, X_1, s_1) \cdots (s_{m-1}, X_m, s_m)$ be in $R'$; if $m = 0$, the new rule is of the form $(s_0, A, s_0) \rightarrow \epsilon$. Function $p'$ assigns the same probability to the new rule as $p$ assigned to the original rule.
- For each $s \overset{a}{\mapsto} t$ in $\Delta$, let $(s, a, t) \rightarrow a$ be in $R'$. Function $p'$ assigns probability 1 to this rule.

Intuitively, a rule of $\mathcal{G}'$ is either constructed out of a rule of $\mathcal{G}$ or out of a transition of $\mathcal{M}$. On the basis of this correspondence between rules and transitions of $\mathcal{G}'$, $\mathcal{G}$ and $\mathcal{M}$, it is not difficult to see that each derivation $d'$ in $\mathcal{G}'$ deriving string $w$ corresponds to a unique derivation $d$ in $\mathcal{G}$ deriving the same string and a unique computation $c$ in $\mathcal{M}$ accepting the same string. Conversely, if there is a derivation $d$ in $\mathcal{G}$ deriving string $w$, and some computation $c$ in $\mathcal{M}$ accepting the same string, then the pair of $d$ and $c$ corresponds to a unique derivation $d'$ in $\mathcal{G}'$ deriving the same string $w$. Furthermore, the probabilities of $d$ and $d'$ are equal, by definition of $p'$.

Let us now assume that each string $w$ is accepted by at most one computation, i.e. $\mathcal{M}$ is unambiguous. If a string $w$ is accepted by $\mathcal{M}$, then there are as many derivations deriving $w$ in $\mathcal{G}'$ as there are in $\mathcal{G}$. If $w$ is not accepted by $\mathcal{M}$, then there are zero derivations deriving $w$ in $\mathcal{G}'$. Consequently:

$$\sum_{\substack{d',w: \\ S' \overset{d'}{\Rightarrow}_{\mathcal{G}'} w}} p'(d') = \sum_{\substack{d,w: \\ S \overset{d}{\Rightarrow}_{\mathcal{G}} w \wedge w \in L(\mathcal{M})}} p(d),$$

or more succinctly:

$$\sum_{w} p'(w) = \sum_{w \in L(\mathcal{M})} p(w).$$

Note that the above construction of $\mathcal{G}'$ is exponential in the largest value of $m$ in any rule from $\mathcal{G}$. For this reason, $\mathcal{G}$ is usually brought in binary form before the intersection, i.e. the input grammar is transformed to let each right-hand side have at most two members. Such a transformation can be realized in linear time in the size of the grammar. We will return to this issue in Section 7.

## 4 Obtaining unambiguous FAs

In the previous section, we explained that unambiguous finite automata have special properties with respect to the grammar $\mathcal{G}'$ that we may construct out of a FA $\mathcal{M}$ and a PCFG $\mathcal{G}$. In this section we discuss how unambiguity can be obtained for the special case of finite automata accepting the language of all strings with given infix $w \in \Sigma^*$:

$$L_{\mathit{infix}}(w) = \{xwy \mid x, y \in \Sigma^*\}.$$

Any deterministic automaton is also unambiguous. Furthermore, there seem to be no practical algorithms that turn FAs into equivalent unambiguous FAs other than the algorithms that also determinize them. Therefore, we will henceforth concentrate on deterministic rather than unambiguous automata.

Given a string $w = a_1 \cdots a_n$, a finite automaton accepting $L_{\mathit{infix}}(w)$ can be straightforwardly constructed. This automaton has states $s_0, \ldots, s_n$, transitions $s_0 \overset{a}{\mapsto} s_0$ and $s_n \overset{a}{\mapsto} s_n$ for each $a \in \Sigma$, and transition $s_{i-1} \overset{a_i}{\mapsto} s_i$ for each $i$ ($1 \leq i \leq n$). The initial state is $s_0$ and the final state is $s_n$. Clearly, there is nondeterminism in state $s_0$.

One way to make this automaton deterministic is to apply the general algorithm of determinization of finite automata; see e.g. (Aho and Ullman, 1972). This algorithm is exponential for general FAs. An alternative approach is to construct a deterministic finite automaton directly from $w$, in line with the Knuth-Morris-Pratt algorithm (Knuth et al., 1977; Gusfield, 1997). Both approaches result in the same deterministic FA, which we denote by $\mathcal{I}_w$. However, the latter approach is easier to implement in such a way that the time complexity of constructing the automaton is linear in $|w|$.

The automaton $\mathcal{I}_w$ is described as follows. There are $n+1$ states $t_0, \ldots, t_n$, where as before $n$ is the length of $w$. The initial state is $t_0$ and the final state is $t_n$. The intuition is that $\mathcal{I}_w$ reads a string $x = b_1 \cdots b_m$ from left to right, and when it has read the prefix $b_1 \cdots b_j$ ($0 \leq j \leq m$), it is in state $t_i$ ($0 \leq i < n$) if and only if $a_1 \cdots a_i$ is the longest prefix of $w$ that is also a suffix of $b_1 \cdots b_j$. If the automaton is in state $t_n$, then this means that $w$ is an infix of $b_1 \cdots b_j$.

In more detail, for each $i$ ($1 \leq i \leq n$) and each $a \in \Sigma$, there is a transition $t_{i-1} \overset{a}{\mapsto} t_j$, where $j$ is the length of the longest string that is both a prefix of $w$ and a suffix of $a_1 \cdots a_{i-1}a$. If $a = a_i$, then clearly $j = i$, and otherwise $j < i$. To ensure that we remain in the final state once an occurrence of infix $w$ has been found, we also add transitions $t_n \overset{a}{\mapsto} t_n$ for each $a \in \Sigma$. This construction is illustrated in Figure 1.

## 5 Infix probability

With the material developed in the previous sections, the problem of computing the infix probabilities can be effectively solved. Our goal is to compute for given infix $w \in \Sigma^*$ and PCFG $\mathcal{G} = (\Sigma, N, S, R, p)$:

$$\sigma_{\mathit{infix}}(w, \mathcal{G}) = \sum_{z \in L_{\mathit{infix}}(w)} p(z).$$

In Section 4 we have shown the construction of finite automaton $\mathcal{I}_w$ accepting $L_{\mathit{infix}}(w)$, by which we obtain:

$$\sigma_{\mathit{infix}}(w, \mathcal{G}) = \sum_{z \in L(\mathcal{I}_w)} p(z).$$

As $\mathcal{I}_w$ is deterministic and therefore unambiguous, the results from Section 3 apply and if $\mathcal{G}' = (\Sigma, N', S', R', p')$ is the PCFG constructed out of $\mathcal{G}$ and $\mathcal{I}_w$ then:

$$\sigma_{\mathit{infix}}(w, \mathcal{G}) = \sum_{z} p'(z).$$

Finally, we can compute the above sum by applying the iterative method discussed in Section 2.

## 6 Extensions

The approach discussed above allows for a number of generalizations. First, we can replace the infix $w$ by a **sequence** of infixes $w_1, \ldots, w_m$, which have to occur in the given order, one strictly after the other, with arbitrary infixes in between:

$$\sigma_{\mathit{island}}(w_1, \ldots, w_m, \mathcal{G}) = \sum_{x_0, \ldots, x_m \in \Sigma^*} p(x_0 w_1 x_1 \cdots w_m x_m).$$

This problem was discussed before by (Corazza et al., 1991), who mentioned applications in speech recognition. Further applications are found in computational biology, but their discussion is beyond the scope of this paper; see for instance (Apostolico et al., 2005) and references therein. In order to solve the problem, we only need a small addition to the procedures we discussed before. First we construct separate automata $\mathcal{I}_{w_j}$ ($1 \leq j \leq m$) as explained in Section 4. These automata are then composed into a single automaton $\mathcal{I}_{(w_1, \ldots, w_m)}$. In this composition, the outgoing transitions of the final state of $\mathcal{I}_{w_j}$, for each $j$ ($1 \leq j < m$), are removed and that final state is merged with the initial state of the next automaton $\mathcal{I}_{w_{j+1}}$. The initial state of the composed automaton is the initial state of $\mathcal{I}_{w_1}$, and the final state is the final state of $\mathcal{I}_{w_m}$. The time costs of constructing $\mathcal{I}_{(w_1, \ldots, w_m)}$ are linear in the sum of the lengths of the strings $w_j$.

Another way to generalize the problem is to replace $w$ by a finite set $L = \{w_1, \ldots, w_m\}$. The objective is to compute:

$$\sigma_{\mathit{infix}}(L, \mathcal{G}) = \sum_{w \in L, x, y \in \Sigma^*} p(xwy)$$

Again, this can be solved by first constructing a deterministic FA, which is then intersected with $\mathcal{G}$. This FA can be obtained by determinizing a straightforward nondeterministic FA accepting $L$, or by directly constructing a deterministic FA along the lines of the Aho-Corasick algorithm (Aho and Corasick, 1975). Construction of the automaton with the latter approach takes linear time.

Further straightforward generalizations involve formalisms such as probabilistic tree adjoining grammars (Schabes, 1992; Resnik, 1992). The technique from Section 3 is also applicable in this case,

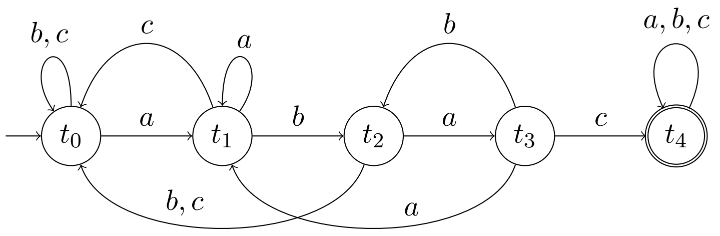

Figure 1: Deterministic automaton that accepts all strings over alphabet $\{a, b, c\}$ with infix *abac*.

as the construction from Bar-Hillel et al. (1964) carries over from context-free grammars to tree adjoining grammars, and more generally to the linear context-free rewriting systems of Vijay-Shanker et al. (1987).

## 7 Implementation

We have conducted experiments with the computation of infix probabilities. The objective was to identify parts of the computation that have a high time or space demand, and that might be improved. The experiments were run on a desktop with a 3.0 GHz Pentium 4 processor. The implementation language is C++.

The set-up of the experiments is similar to that in (Nederhof and Satta, 2008). A probabilistic context-free grammar was extracted from sections 2-21 of the Penn Treebank version II. Subtrees that generated the empty string were systematically removed. The result was a CFG with 10,035 rules, 28 nonterminals and 36 parts-of-speech. The rule probabilities were determined by maximum likelihood estimation. The grammar was subsequently binarized, to avoid exponential behaviour, as explained in Section 3.

We have considered 10 strings of length 7, randomly generated, assuming each of the parts-of-speech has the same probability. For all prefixes of those strings from length 2 to length 7, we then computed the infix probability. The duration of the full computation, averaged over the 10 strings of length 7, is given in the first row of Table 1.

In order to solve the non-linear systems of equations, we used Broyden's method. It can be seen as an approximation of Newton's method. It requires more iterations, but seems to be faster overall, and more scalable to large problem sizes, due to the avoidance of matrix inversion, which sometimes makes Newton's method prohibitively expensive. In our experiments, Broyden's method was generally faster than Newton's method and much faster than the simple iteration method by the relation in (4). For further details on Broyden's method, we refer the reader to (Kelley, 1995).

The main obstacle to computation for infixes substantially longer than 7 symbols is the memory consumption rather than the running time. This is due to the required square matrices, the dimension of which is the number of nonterminals. The number of nonterminals (of the intersection grammar) naturally grows as the infix becomes longer.

As explained in Section 2, the problem is divided into smaller problems by isolating disjoint sets of mutually recursive nonterminals, or strongly connected components. We found that for the application to the automata discussed in Section 4, there were exactly three strongly connected components that contained more than one element, throughout the experiments. For an infix of length $n$, these components are:

- $C_1$, which consists of nonterminals of the form $(t_i, A, t_j)$, where $i < n$ and $j < n$,
- $C_2$, which consists of nonterminals of the form $(t_i, A, t_j)$, where $i = j = n$, and
- $C_3$, which consists of nonterminals of the form $(t_i, A, t_j)$, where $i < j = n$.

This can be easily explained by looking at the structure of our automata. See for example Figure 1, with cycles running through states $t_0, \ldots, t_{n-1}$, and cycles through state $t_n$. Furthermore, the grammar extracted from the Penn Treebank is heavily recursive,

| infix length | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| total running time | 1.07 | 1.95 | 5.84 | 11.38 | 23.93 | 45.91 |
| Broyden's method for $C_1$ | 0.46 | 0.90 | 3.42 | 6.63 | 12.91 | 24.38 |
| Broyden's method for $C_2$ | 0.08 | 0.04 | 0.07 | 0.04 | 0.03 | 0.09 |
| Broyden's method for $C_3$ | 0.20 | 0.36 | 0.81 | 1.74 | 5.30 | 9.02 |

Table 1: Running time for infixes from length 2 to length 7. The infixes are prefixes of 10 random strings of length 7, and reported CPU times (in seconds) are averaged over the 10 strings.

so that almost every nonterminal can directly or indirectly call any other.

The strongly connected component $C_2$ is always the same, consisting of 2402 nonterminals, for each infix of any length. (Note that binarization of the grammar introduced artificial nonterminals.) The last three rows of Table 1 present the time costs of Broyden's method, for the three strongly connected components.

The strongly connected component $C_3$ happens to correspond to a *linear* system of equations. This is because a rule in the intersection grammar with a left-hand side $(t_i, A, t_j)$, where $i < j = n$, must have a right-hand side of the form $(t_i, A', t_j)$, or of the form $(t_i, A_1, t_k)$ $(t_k, A_2, t_j)$, with $k \leq n$. If $k < n$, then only the second member can be in $C_3$. If $k = n$, only first member can be in $C_3$. Hence, such a rule corresponds to a linear equation within the system of equations for the entire grammar.

A linear system of equations can be solved analytically, for example by Gaussian elimination, rather than approximated through Newton's method or Broyden's method. This means that the running times in the last row of Table 1 can be reduced by treating $C_3$ differently from the other strongly connected components. However, the running time for $C_1$ dominates the total time consumption.

The above investigations were motivated by two questions, namely whether any part of the computation can be precomputed, and second, whether infix probabilities can be computed incrementally, for infixes that are extended to the left or to the right. The first question can be answered affirmatively for $C_2$, as it is always the same. However, as we can see in Table 1, the computation of $C_2$ amounts to a small portion of the total time consumption.

The second question can be rephrased more precisely as follows. Suppose we have computed the infix probability of a string $w$ and have kept intermediate results in memory. Can the computation of the infix probability of a string of the form $aw$ or $wa$, $a \in \Sigma$, be computed by relying on the existing results, so that the computation is substantially faster than if the computation were done from scratch?

Our investigations so far have not found a positive answer to this second question. In particular, the systems of equations for $C_1$ and $C_3$ change fundamentally if the infix is extended by one more symbol, which seems to at least make incremental computation very difficult, if not impossible. Note that the algorithms for the computation of prefix probabilities by Jelinek and Lafferty (1991) and Stolcke (1995) do allow incrementality, which contributes to their practical usefulness for speech recognition.

## 8 Conclusions

We have shown that the problem of infix probabilities for PCFGs can be solved by a construction that intersects a context-free language with a regular language. An important constraint is that the finite automaton that is input to this construction be unambiguous. We have shown that such an automaton can be efficiently constructed. Once the input probabilistic PCFG and the FA have been combined into a new probabilistic CFG, the infix probability can be straightforwardly solved by iterative algorithms. Such algorithms include Newton's method, and Broyden's method, which was used in our experiments. Our discussion ended with an open question about the possibility of incremental computation of infix probabilities.

## References

S. Abney, D. McAllester, and F. Pereira. 1999. Relating probabilistic grammars and automata. In *37th Annual*

*Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 542–549, Maryland, USA, June.

A.V. Aho and M.J. Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, June.

A.V. Aho and J.D. Ullman. 1972. *Parsing*, volume 1 of *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, Englewood Cliffs, N.J.

A. Apostolico, M. Comin, and L. Parida. 2005. Conservative extraction of overrepresented extensible motifs. In *Proceedings of Intelligent Systems for Molecular Biology (ISMB05)*.

Y. Bar-Hillel, M. Perles, and E. Shamir. 1964. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison-Wesley, Reading, Massachusetts.

T.L. Booth and R.A. Thompson. 1973. Applying probabilistic measures to abstract languages. *IEEE Transactions on Computers*, C-22:442–450.

Z. Chi and S. Geman. 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Z. Chi. 1999. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.

A. Corazza, R. De Mori, R. Gretter, and G. Satta. 1991. Computation of probabilities for an island-driven parser. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):936–950.

K. Etessami and M. Yannakakis. 2005. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In *22nd International Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 340–352, Stuttgart, Germany. Springer-Verlag.

K. Etessami and M. Yannakakis. 2009. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66.

A.L.N. Fred. 2000. Computation of substring probabilities in stochastic grammars. In A. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Artificial Intelligence*, pages 103–114. Springer-Verlag.

D. Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge.

T.E. Harris. 1963. *The Theory of Branching Processes*. Springer-Verlag, Berlin, Germany.

F. Jelinek and J.D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

C.T. Kelley. 1995. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

S. Kiefer, M. Luttenberger, and J. Esparza. 2007. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 217–266.

D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt. 1977. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350.

M.-J. Nederhof and G. Satta. 2003. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, LORIA, Nancy, France, April.

M.-J. Nederhof and G. Satta. 2008. Computing partition functions of PCFGs. *Research on Language and Computation*, 6(2):139–162.

E. Persoon and K.S. Fu. 1975. Sequential classification of strings generated by SCFG's. *International Journal of Computer and Information Sciences*, 4(3):205–217.

P. Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proc. of the fifteenth International Conference on Computational Linguistics*, Nantes, August, pages 418–424.

J.-A. Sánchez and J.-M. Benedí. 1997. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055, September.

Y. Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proc. of the fifteenth International Conference on Computational Linguistics*, Nantes, August, pages 426–432.

A. Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):167–201.

K. Vijay-Shanker, D.J. Weir, and A.K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 104–111, Stanford, California, USA, July.

D. Wojtczak and K. Etessami. 2007. PReMo: an analyzer for Probabilistic Recursive Models. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference*, volume 4424 of *Lecture Notes in Computer Science*, pages 66–71, Braga, Portugal. Springer-Verlag.