Res Lang Comput (2008) 6:139–162
DOI 10.1007/s11168-008-9052-8

# Computing Partition Functions of PCFGs

**Mark-Jan Nederhof · Giorgio Satta**

Published online: 22 October 2008


**Abstract** We investigate the problem of computing the partition function of a probabilistic context-free grammar, and consider a number of applicable methods. Particular attention is devoted to PCFGs that result from the intersection of another PCFG and a finite automaton. We report experiments involving the Wall Street Journal corpus.

**Keywords** Equation solving · Formal grammars · Statistical NLP · WSJ corpus

## 1 Introduction

Probabilistic context-free grammars (PCFGs) are commonly thought of as describing probability distributions over strings of terminal symbols and over derivations of such strings. This has many applications in natural language processing, such as for the disambiguation of structurally ambiguous sentences and for ranking hypotheses returned by a speech recognizer. See for example Manning and Schütze (1999) for further discussion.

Probability distributions over strings or derivations by definition imply that the sum of probabilities of all strings or derivations should be 1. If the strings or derivations are generated by a grammar, this condition is often called *consistency*. Although some frameworks may rely on consistency, for others it is not essential, and relaxing this condition creates new opportunities for developing probabilistic methods.

M.-J. Nederhof (✉)
School of Computer Science, University of St Andrews, North Haugh,
St Andrews KY16 9SX, Scotland, UK
e-mail: markjan.nederhof@gmail.com

G. Satta
Department of Information Engineering, University of Padua, Via Gradenigo 6/A, 35131 Padova, Italy
e-mail: satta@dei.unipd.it

Motivated by a number of applications to be discussed later, we consider PCFGs that generate terminal strings with a total probability strictly smaller than 1, and investigate the problem of the computation of such a probability. As auxiliary concept, we introduce a function $Z$ on nonterminals from a given PCFG, called the *partition function* of the PCFG. The value $Z(A)$ is the sum of probabilities of all derivations of terminal strings from $A$. We are thus mainly interested in the computation of the value $Z(S)$, where $S$ is the start symbol. As will be shown, the values of the partition function are intimately related and can be obtained as a specific solution of a system of polynomial, nonlinear equations. In the literature on statistical parsing, such a system is usually solved by means of least fixed-point iteration; see for instance work by Stolcke (1995) and by Abney et al. (1999), and see Kelley (1995) for a general presentation of least fixed-point iteration.

In this article we also consider two alternative methods to solve the above-mentioned system of equations. The first is Newton's method, which has the advantage that it typically converges in very few iterations. Below we will motivate one application of partition functions that involves PCFGs of very large size that are obtained by intersection of another PCFG and a finite automaton. Large PCFGs lead to very large matrices and costly matrix operations, making straightforward use of Newton's method problematic. We will discuss a way around this problem, resulting in smaller matrices. The use of Newton's method for the computation of partition functions has been originally proposed by Etessami and Yannakakis (2005) (see also Wojtczak and Etessami 2007), in the context of recursive Markov chains, a class of probabilistic generative models that are more expressive than PCFGs. A related but different use of Newton's method is as optimisation algorithm. In the field of NLP this has been frequently applied for the purpose of maximum entropy parameter estimation (Malouf 2002).

We further consider Broyden's method for the computation of partition functions. Its intermediate results can be seen as approximations of intermediate results that would be computed by Newton's method. Because of the approximation, a larger number of steps may be required for the method to converge. However, Broyden's method can be implemented without any matrix operations, and consequently each individual step is less costly than a comparable step of Newton's method. Broyden's method typically offers better performance than Newton's method if the dimensions of the matrices would be very large.

The partition function of a PCFG has a large number of applications in statistical natural language processing and more in general in syntactic pattern recognition. For example, let $\mathcal{G}$ be a consistent PCFG, representing a language model, and consider some property $\Phi$ of strings that can be formulated by means of a regular expression or, equivalently, by a finite automaton $\mathcal{M}$. In order to compute the total probability in our language model of all strings that have the property $\Phi$, we can construct a new PCFG $\mathcal{G}_\Phi$ that generates the intersection of the languages generated by $\mathcal{G}$ and $\mathcal{M}$, in such a way that each string generated by $\mathcal{G}_\Phi$ has the same probability as in $\mathcal{G}$. Note that, in general, $\mathcal{G}_\Phi$ is not a consistent PCFG. The desired probability can be obtained by computing the value $Z_\Phi(S)$, where $Z_\Phi$ is the partition function associated with $\mathcal{G}_\Phi$ and $S$ is the start symbol of this grammar. For probabilistic finite automata rather than for PCFGs, the efficient computation of value $Z_\Phi(S)$ is mentioned as an important open problem at Sect. 5, Problem 3 of Vidal et al. (2005).

An example of a property $\Phi$ of interest in statistical natural language processing is the set of all strings that have a given infix (or substring or factor, as it is also called). In this case, $Z_\Phi(S)$ is called the *infix probability*. Applications are found in island-driven and word spotting techniques for speech recognition, when only a few words somewhere within a sentence are recognized and analysis needs to be expanded outward; see Corazza et al. (1991) and Nederhof and Satta (2003) for further discussion. For prefixes (and by symmetry also for suffixes) the above problem was investigated by Persoon and Fu (1975), Jelinek and Lafferty (1991) and Stolcke (1995), and satisfactory algorithms were found. It was pointed out by Corazza et al. (1991) that the problem of the computation of infix probabilities is computationally more difficult than that of the computation of prefix probabilities. The authors developed solutions for the case where a 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 PCFG is a regular grammar, thus reducing the power to finite-state machinery.

The partition function has other applications as well. For example, Thompson (1974) investigates normalization of a consistent PCFG that is non-proper; properness means that for each nonterminal, the sum of probabilities of the rules with that nonterminal as the left-hand side is 1. A slightly more general problem, involving non-proper and non-consistent PCFGs, was investigated by Chi (1999); see also Nederhof and Satta (2006). These methods of normalization heavily rely on the partition function. Normalization can be the second part of a two-step process of restructuring a PCFG. This was for example proposed by Abney et al. (1999) for the transformation into Greibach normal form (see also Huang and Fu 1971); as first step, the grammar is transformed such that consistency is preserved, but not properness.

A related application is the elimination of epsilon rules from a PCFG. This is needed, for instance, as a step in the transformation of a PCFG into Chomsky normal form; see Abney et al. (1999). Another relevant application is the construction of an efficient parser that determines prefix probabilities for general form PCFGs; see for instance (Stolcke 1995). These two applications require the summation of probabilities of all derivations from a nonterminal $A$ that generate the empty string. The problem can be cast in the framework presented above, by considering the PCFG resulting from the intersection of a source PCFG (with above-mentioned nonterminal $A$ as start symbol) and a finite automaton generating only the empty string, and by computing the partition function associated with the resulting PCFG. If the inside-outside algorithm (Baker 1979) is performed with a PCFG that is not in Chomsky normal form, the inside probabilities can be computed in the same way as above.

The structure of this paper is as follows. In Sect. 2 we recall standard notation and terminology, and the partition function of a PCFG is formally defined in Sect. 3, along with a presentation of a straightforward method for its computation. The use of Newton's method for the computation of the partition function is discussed in Sect. 4, and use of Broyden's method is considered in Sect. 5. Section 6 discusses the problem of computing the partition function of a PCFG resulting from intersection of another PCFG and a finite automaton. Section 7 investigates the performance of the discussed methods in practice. We end with some concluding remarks in Sect. 8.

## 2 Preliminaries

The main formalism we consider in this work is a probabilistic extension of context-free grammars (CFGs), as defined for example by Thompson (1974). A *probabilistic* CFG (PCFG) is a tuple $\mathcal{G} = (\Sigma, N, S, R, p)$, where $\Sigma$ and $N$ are two finite disjoint sets of terminals and nonterminals, respectively, $S \in N$ is the start symbol, and $R$ 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 $[0, 1]$.

In what follows, symbol $a$ ranges over the set $\Sigma$, symbol $w$ ranges over the set $\Sigma^*$, symbols $A$ and $B$ range over the set $N$, symbols $X$ and $Y$ range over the set $\Sigma \cup N$, symbols $\alpha, \beta, \ldots$ range over the set $(\Sigma \cup N)^*$, symbol $\pi$ ranges over the set $R$, and symbol $d$ ranges over the set $R^*$. The empty string is denoted by $\varepsilon$.

For a fixed PCFG, we define the left-most rewriting relation $\Rightarrow$ on triples consisting of two strings $\alpha, \beta \in (\Sigma \cup N)^*$ and a rule $\pi \in R$ by: $\alpha \stackrel{\pi}{\Rightarrow} \beta$ if and only if $\alpha$ is of the form $wA\delta$ and $\beta$ is of the form $w\gamma\delta$, for some $w \in \Sigma^*$ and $\delta \in (\Sigma \cup N)^*$, and $\pi = (A \to \gamma)$. A *left-most derivation* is a string $d = \pi_1 \cdots \pi_m$, $m \geq 0$, such that $X \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} w$, for some grammar symbol $X$ and terminal string $w$. We also write $X \stackrel{d}{\Rightarrow} w$ for such a left-most derivation $d$. In the case of left-most derivation $d = \varepsilon$, this may be expressed as $a \stackrel{d}{\Rightarrow} a$ for any terminal $a$. We write $\alpha \Rightarrow^* \beta$ to denote the existence of a string $\pi_1 \cdots \pi_m$ such that $\alpha \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} \beta$.

For a grammar symbol $X$, a string $d = \pi_1 \cdots \pi_m$, and a terminal string $w$, we define $p(X \stackrel{d}{\Rightarrow} w)$ to be $\prod_{i=1}^{m} p(\pi_i)$ if $d$ is a left-most derivation such that $X \stackrel{d}{\Rightarrow} w$, and 0 otherwise. The probability $p(w)$ of a string $w$ is defined to be the sum of all left-most derivations $d$ from the start symbol $S$, or formally $p(w) = \sum_d p(S \stackrel{d}{\Rightarrow} w)$.

The length $|\pi|$ of a rule $\pi = (A \to X_1 \cdots X_n)$ is defined to be $n + 1$. The size $|\mathcal{G}|$ of a PCFG $\mathcal{G}$ is defined to be $\sum_{\pi \in R} |\pi|$.

The depth $depth(d)$ of a left-most derivation $d$ is the maximal number of rules visited on a path from the root to a leaf in the familiar representation as parse tree. More precisely, $depth(\epsilon) = 0$ and if $\pi = (A \to X_1 \cdots X_m)$ and $X_i \stackrel{d_i}{\Rightarrow} w_i$ $(1 \leq i \leq m)$, then $depth(\pi d_1 \cdots d_m) = 1 + \max_i depth(d_i)$.

## 3 Partition Function

Given a fixed PCFG, the partition function $Z$ maps each nonterminal $A$ to the sum of probabilities of left-most derivations of terminal strings from that nonterminal. Formally:

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

By decomposing derivations into smaller derivations, and by making use of the fact that multiplication distributes over addition, we can derive:

$$Z(A) = \sum_{A \to \alpha} p(A \to \alpha) \cdot Z(\alpha), \tag{2}$$

where we define:

$$Z(\epsilon) = 1, \tag{3}$$
$$Z(a\beta) = Z(\beta), \tag{4}$$
$$Z(B\beta) = Z(B) \cdot Z(\beta), \quad \text{for } \beta \neq \varepsilon. \tag{5}$$

The partition function may be approximated by only considering derivations up to a certain depth. We define for all $A$ and $k \geq 0$:

$$Z_k(A) = \sum_{d, w : depth(d) \leq k} p(A \stackrel{d}{\Rightarrow} w). \tag{6}$$

By again decomposing derivations, we obtain a recursive characterization:

$$Z_{k+1}(A) = \sum_{A \to \alpha} p(A \to \alpha) \cdot Z_k(\alpha), \tag{7}$$

and $Z_0(A) = 0$ for all $A$, where we define:

$$Z_k(\epsilon) = 1, \tag{8}$$
$$Z_k(a\beta) = Z_k(\beta), \tag{9}$$
$$Z_k(B\beta) = Z_k(B) \cdot Z_k(\beta), \text{ for } \beta \neq \varepsilon. \tag{10}$$

Naturally, for all $A$:

$$\lim_{k \to \infty} Z_k(A) = Z(A). \tag{11}$$

We can interpret Eq. 2 as a system of polynomial equations over variables $Z(A)$, for the set of nonterminals $A \in N$. If we choose a fixed ordering of the nonterminals as $A_1, \ldots, A_m$, then this system can be written as:

$$(Z(A_1), \ldots, Z(A_m)) = F\left(Z(A_1), \ldots, Z(A_m)\right), \tag{12}$$

where $F$ is the function from $m$-tuples to $m$-tuples defined by:

$$F\left(Z(A_1), \ldots, Z(A_m)\right) = \left( \sum_{A_1 \to \alpha_1} p(A_1 \to \alpha_1) \cdot Z(\alpha_1), \ldots, \sum_{A_m \to \alpha_m} p(A_m \to \alpha_m) \cdot Z(\alpha_m) \right). \tag{13}$$

The tuple of $m$ values $Z_k(A_i)$, $1 \leq i \leq m$, can be expressed as $F^k(0, \ldots, 0)$.

Equation 12 may have several solutions. By Eqs. 1 and 11 we know that the intended solution is $\lim_{k\to\infty} F^k(0,\ldots,0)$. As $F$ is monotone and continuous, we may invoke Kleene's fixed-point theorem to conclude the intended solution is the smallest solution of Eq. 12 that consists of $m$ non-negative numbers.

As an example, we consider the following PCFG:

$$\begin{array}{ll} S \to S\ S & [0.6] \\ S \to a & [0.4] \end{array}$$

We need to find the smallest solution to:

$$Z(S) = 0.6 \cdot Z(S) \cdot Z(S) + 0.4 \tag{14}$$

Whereas this simple example allows the solution $Z(S) = \frac{2}{3}$ to be derived by analytic means, this is not possible in general, and we need to resort to computing an approximation:

$$\begin{aligned} Z_0(S) &= 0 \\ Z_1(S) &= 0.6 \cdot Z_0(S) \cdot Z_0(S) + 0.4 = 0.4 \\ Z_2(S) &= 0.6 \cdot Z_1(S) \cdot Z_1(S) + 0.4 \approx 0.496 \\ Z_3(S) &= 0.6 \cdot Z_2(S) \cdot Z_2(S) + 0.4 \approx 0.548 \\ &\ldots \\ Z_{20}(S) &= 0.6 \cdot Z_{19}(S) \cdot Z_{19}(S) + 0.4 \approx 0.665 \\ &\ldots \end{aligned}$$

The algorithm that approximates the values $Z(A)$ by iteration, computing the values $Z_k(A)$ for $k = 1, \ldots$ until they stabilize, will be referred to as the *fixed-point method*. This approach has been commonly applied in natural language processing for the computation of the partition function; see for instance the work by Thompson (1974), Stolcke (1995) and Abney et al. (1999). Formal properties are discussed by Etessami and Yannakakis (2005) and Kiefer et al. (2007). Fixed-point iteration is well-known in the numerical calculus literature and is frequently applied to systems of nonlinear equations because it can be easily implemented. When a number of standard conditions are met, each iteration of the algorithm (corresponding to the value of $k$ in the above equations) adds a fixed number of bits to the precision of the approximated solution; see Kelley (1995) for further discussion.

For performance reasons, fixed-point iteration can be preceded by a partitioning of the set $N$ of nonterminals of a PCFG into maximal sets of mutually recursive nonterminals. More precisely, two nonterminals $A$ and $B$ are in the same set of the partition if and only if $A \Rightarrow^* \alpha B \beta$ and $B \Rightarrow^* \gamma A \delta$, some $\alpha$, $\beta$, $\gamma$ and $\delta$. For two distinct sets $N'$ and $N''$ of the partition, we define $N' \prec N''$ if and only if $A \Rightarrow^* \alpha B \beta$, for some $A \in N'$ and $B \in N''$, and some $\alpha$ and $\beta$. The approximations $Z_k$ of the partition function are computed in accordance with this partial order, that is, if $N' \prec N''$, then the computation of the values $Z_k(B)$ is completed for nonterminals $B \in N''$ before the same is done for nonterminals $A \in N'$. In other words, no computation is wasted

on finding rough approximations for the values associated with $N'$ before reliable approximations for the values associated with $N''$ are available.

Consider a maximal set $N'$ of mutually recursive nonterminals and assume that we have already determined a suitable approximation $\tilde{Z}(B)$ of $Z(B)$ for each $B$ in each set $N''$ such that $N' \prec N''$. Let $B\beta$ be a suffix of the right-hand side of a rule with left-hand side $A \in N'$, for some $B$. We split Eq. 10 into two cases, depending on whether $B$ belongs to the current set $N'$ or to another set $N''$:

$$Z_k(B\beta) = \tilde{Z}(B) \cdot Z_k(\beta), \quad \text{for } B \notin N', \tag{15}$$

$$Z_k(B\beta) = Z_k(B) \cdot Z_k(\beta), \quad \text{for } B \in N' \text{ and } \beta \neq \varepsilon. \tag{16}$$

We stop iterating when the difference between $Z_k(A)$ and $Z_{k-1}(A)$ is below some threshold for each $A \in N'$. We then take $Z_k(A)$ to be a suitable approximation of $Z(A)$, which we again denote by $\tilde{Z}(A)$.

## 4 Newton's Method

Newton's method, also called the Newton-Raphson method, is a root-finding algorithm that derives from the Taylor series expansion.

### 4.1 Basic Definitions

In the case of a function $f$ involving a single variable, the method approximates a solution to $f(x) = 0$ by the series $x^{(0)}, x^{(1)}, \ldots$, where $x^{(0)}$ is the initial estimation, and for $k \geq 0$:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \tag{17}$$

where $f'$ denotes the first derivative of $f$.

In this article, we consider multivariate functions of the form:

$$F(\vec{x}) = (f_1(\vec{x}), \ldots, f_m(\vec{x}))^T, \tag{18}$$

where $\vec{x}$ denotes a column vector $(x_1, \ldots, x_m)^T$ of variables ranging over the real values. The $T$ stands for transposition, which is used here to turn row vectors into column vectors.

Newton's method approximates a solution to $f_i(\vec{x}) = 0$ for all $i$ $(1 \leq i \leq m)$ by the series $\vec{x}^{(0)}, \vec{x}^{(1)}, \ldots$, where $\vec{x}^{(0)}$ is the initial estimation, and for $k \geq 0$:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - JF(\vec{x}^{(k)})^{-1} \cdot F(\vec{x}^{(k)}). \tag{19}$$

The expression $JF(\vec{x})$ denotes the Jacobian matrix of $F$, that is:

$$JF(\vec{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_m} \end{pmatrix}. \tag{20}$$

Further, $JF(\vec{x}^{(k)})$ denotes the real-valued matrix that results by substituting $x_i^{(k)}$ for each variable $x_i$ $(1 \leq i \leq m)$ in $JF(\vec{x})$. The superscript in $JF(\vec{x}^{(k)})^{-1}$ denotes matrix inversion. Practical implementations of matrix inversion have a cubic time complexity in the number of variables (Cormen et al. 1990), and therefore this is generally the most expensive operation needed to compute $\vec{x}^{(k+1)}$ from $\vec{x}^{(k)}$.

### 4.2 Standard Application

To cast the partition function into a form allowing application of Newton's method, we again divide the set of nonterminals of a PCFG into maximal sets of mutually recursive nonterminals. Consider one such set $N'$, and assume a fixed ordering of the nonterminals in this set as $A_1, \ldots, A_m$. Assume that appropriate approximations $\tilde{Z}(B)$ have already been computed for each $B$ in each set $N''$ such that $N' \prec N''$.

The intended values $Z(A_1), \ldots, Z(A_m)$ are now the smallest non-negative solution to the equations $f_i(\vec{x}) = 0$, where $f_i$ $(1 \leq i \leq m)$ is defined by:

$$f_i(\vec{x}) = -x_i + \sum_{A_i \to \alpha} p(A_i \to \alpha) \cdot f_\alpha(\vec{x}), \tag{21}$$

where we define:

$$f_\epsilon(\vec{x}) = 1, \tag{22}$$
$$f_{a\beta}(\vec{x}) = f_\beta(\vec{x}), \tag{23}$$
$$f_{B\beta}(\vec{x}) = \tilde{Z}(B) \cdot f_\beta(\vec{x}), \quad \text{for } B \notin N', \tag{24}$$
$$f_{B\beta}(\vec{x}) = x_i \cdot f_\beta(\vec{x}), \quad \text{for } B = A_i. \tag{25}$$

The element in row $i$ and column $j$ of $JF(\vec{x})$ is $\frac{\partial f_i}{\partial x_j}(\vec{x})$. For notational convenience, we write it as $df_{i,j}(\vec{x})$. We can make use of the fact that for any pair of differentiable functions $f$ and $g$:

$$\frac{\partial (f \cdot g)}{\partial x_j} = f \cdot \frac{\partial g}{\partial x_j} + \frac{\partial f}{\partial x_j} \cdot g. \tag{26}$$

In addition:

$$\frac{\partial x_i}{\partial x_j} = \delta_{i,j}, \tag{27}$$

with the Kronecker delta defined as usual by $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ otherwise. From Eqs. 21 to 25 we then obtain:

$$df_{i,j}(\vec{x}) = -\delta_{i,j} + \sum_{A_i \to \alpha} p(A_i \to \alpha) \cdot df_{\alpha,j}(\vec{x}), \tag{28}$$

where we define:

$$df_{\epsilon,j}(\vec{x}) = 0, \tag{29}$$

$$df_{a\beta,j}(\vec{x}) = df_{\beta,j}(\vec{x}), \tag{30}$$

$$df_{B\beta,j}(\vec{x}) = \tilde{Z}(B) \cdot df_{\beta,j}(\vec{x}), \text{ for } B \notin N', \tag{31}$$

$$df_{B\beta,j}(\vec{x}) = x_i \cdot df_{\beta,j}(\vec{x}) + \delta_{i,j} \cdot f_\beta(\vec{x}), \text{ for } B = A_i. \tag{32}$$

In our experiments with Newton's method, reported in Sect. 7, we always take $x^{(0)} = (0, \ldots, 0)^T$ of length $m$. It has been recently shown by Etessami and Yannakakis (2005) and Kiefer et al. (2007) that, with this initial estimation, Newton's method converges to the same solution as the fixed-point method from Sect. 3. This holds for the application under discussion, which involves systems of polynomial, nonlinear equations with only positive coefficients. In the general case, allowing negative coefficients, Newton's method may not converge, or it may converge to a solution other than the least fixed-point.

When applying Newton's method, we partition the set of nonterminals into maximal sets of mutually recursive nonterminals, as we did in the case of the fixed-point method. Whereas in the latter case the motivation was to improve performance, in the case of Newton's method such a partition is essential. More precisely, it has been proved by Etessami and Yannakakis (2005) that the Jacobian matrix corresponding to a set of nonterminals from an arbitrary PCFG cannot be inverted in general, unless that set consists of mutually recursive nonterminals.

For the application under discussion, Etessami and Yannakakis (2005) have shown that the convergence of Newton's method is asymptotically at least as fast as that of the fixed-point method, and exponentially faster in many cases. One such case is if the PCFG is critical. Informally, a proper PCFG is critical if it lies at the boundary between consistency and inconsistency; see Chi (1999) for a more precise definition. For critical PCFGs, the number of iterations needed by the fixed-point method to compute the first $c$ bits of the solution may grow exponentially in $c$. We will see an example of this degenerate behaviour in Sect. 7. In contrast, it has been shown by Kiefer et al. (2007) that for Newton's method, the number of iterations grows linearly in $c$. As the running time of one iteration is polynomial in the number of bits of the required floating point representation, it follows that the total running time is polynomial in $c$ as well.

Whereas Newton's method has a favourable time complexity for a fixed grammar and a variable precision, the asymptotic complexity in terms of the grammar size is more problematic. For a family of grammars presented by Kiefer et al. (2007), the number of iterations needed to compute the first $c$ bits of the desired solution, for some constant $c$, is exponential in the number of maximal sets of mutually recursive nonterminals. This also holds for the fixed-point method and all related methods however.

### 4.3 Linking Nonterminals

A major problem with the standard application of Newton's method for the computation of the partition function as discussed above is that maximal sets of mutually recursive nonterminals can be quite large. One important application where this arises will be discussed in Sect. 6. Large sets of mutually recursive nonterminals give rise to large matrices that are to be inverted, and matrix inversion is generally the most costly operation in an iteration of Newton's method.

In this section, we investigate how this problem can be alleviated by preventing many nonterminals from acting as variables in the required operation of matrix inversion. More precisely, we choose a subset $N_l$ for each maximal set $N'$ of mutually recursive nonterminals, with the following property. There is a bound $b$ such that if $A_0 \to \alpha_1 A_1 \beta_1$, $A_1 \to \alpha_2 A_2 \beta_2$, ..., $A_{n-1} \to \alpha_n A_n \beta_n$ and $A_1, \ldots, A_n \in N_l$, then $n \leq b$. In words, if we descend downwards in a derivation passing through nonterminals in $N_l$, we must encounter a nonterminal not in $N_l$ within a predetermined number of steps. Although all nonterminals in $N'$ are recursive (provided $N'$ contains at least two elements), the recursive structure can be captured in terms of nonterminals in $N' - N_l$, which are linked by nonterminals in $N_l$. We therefore refer to a set $N_l$ as a set of *linking* nonterminals. In terms of the partition function, we may say that the value assigned to a non-linking nonterminal can be expressed as a combination of values assigned to other non-linking nonterminals, and the values assigned to linking nonterminals merely represent intermediate results.

For each set $N'$ there may be several valid choices of $N_l$. A general method to obtain a suitable set of linking nonterminals consists of the following. We map the grammar to a graph whose vertices are the nonterminals. There is an edge from $A$ to $B$ if and only if there is a rule of the form $A \to \alpha B \beta$. We then do a depth-first search of the entire grammar starting from $S$, following Cormen et al. (1990). Upon leaving a recursive call, the corresponding nonterminal is assigned a unique number. As such numbers are assigned in increasing order, this number for a nonterminal $A$ is called its *finishing time* and denoted here as $\mathit{finish}(A)$.

We then choose a nonterminal $B$ to be linking in the relevant maximal set of mutually recursive nonterminals if there is no rule of the form $A \to \alpha B \beta$ with $\mathit{finish}(A) \leq \mathit{finish}(B)$. It is clear that if $A_0 \to \alpha_1 A_1 \beta_1$, ..., $A_{n-1} \to \alpha_n A_n \beta_n$ and $A_1$, ..., $A_n$ are linking by the above construction, then $\mathit{finish}(A_0) > \mathit{finish}(A_1) > \cdots > \mathit{finish}(A_n)$, and therefore $n$ is bounded by the number of nonterminals.

The dimension $m$ of a square matrix $JF(\vec{x}^{(k)})$ that we construct for each $N'$ at each iteration of Newton's method is now $|N' - N_l|$, where $N'$ is a maximal set of mutually recursive nonterminals, and $N_l$ is the subset thereof of linking nonterminals. Let us assume a fixed ordering of the non-linking nonterminals as $A_1, \ldots, A_m$. This warrants specialization of Eq. 25 into two cases:

$$f_{B\beta}(\vec{x}) = x_i \cdot f_\beta(\vec{x}), \quad \text{for } B = A_i, \tag{33}$$

$$f_{B\beta}(\vec{x}) = f_B(\vec{x}) \cdot f_\beta(\vec{x}), \quad \text{for } B \in N_l, \tag{34}$$

where we define for $B \in N_l$:

$$f_B(\vec{x}) = \sum_{B \to \alpha} p(B \to \alpha) \cdot f_\alpha(\vec{x}). \tag{35}$$

Accordingly, we replace Eq. 32 by two equations:

$$df_{B\beta,j}(\vec{x}) = x_i \cdot df_{\beta,j}(\vec{x}) + \delta_{i,j} \cdot f_\beta(\vec{x}), \quad \text{for } B = A_i, \tag{36}$$
$$df_{B\beta,j}(\vec{x}) = f_B(\vec{x}) \cdot df_{\beta,j}(\vec{x}) + df_{B,j}(\vec{x}) \cdot f_\beta(\vec{x}), \quad \text{for } B \in N_l, \tag{37}$$

where we define for $B \in N_l$:

$$df_{B,j}(\vec{x}) = \sum_{B \to \alpha} p(B \to \alpha) \cdot df_{\alpha,j}(\vec{x}). \tag{38}$$

An example of the use of these equations will be given in Sect. 6.4.

For the computation of one real-valued matrix $JF(\vec{x}^{(k)})$, some of the auxiliary values may be used more than once. For example, one value $df_{B,j}(\vec{x}^{(k)})$, as defined in Eq. 38, may be needed for the computation of the right-hand side of Eq. 37 for several distinct choices of $B\beta$ in the left-hand side. By tabulating values such as $df_{B,j}(\vec{x}^{(k)})$, the computation of matrix $JF(\vec{x}^{(k)})$ can be performed in quadratic time in the size of the PCFG. This can be easily verified by counting the number of distinct instantiations of each of the mentioned equations, in terms of the number of suffixes of right-hand sides of rules and the number of nonterminals in the grammar.

In the experiments reported below, where we look at running time, we always assume tabulation of auxiliary values. This also holds for the standard application of Newton's method *without* linking nonterminals.

One more optimization is based on the following observation. A value $df_{\beta,j}(\vec{x})$ is predictably zero if $f_\beta(\vec{x})$ does not depend, directly or indirectly via linking nonterminals, on the value of the non-linking nonterminal $x_j$. We perform a static analysis at the beginning of the computation to determine such dependencies, so that some unnecessary work may be avoided.

### 4.4 Further Optimisations in the Experiments

There is a way to avoid matrix inversion for computing the value in (19). First we solve the linear system:

$$JF(\vec{x}^{(k)}) \cdot \vec{y}^{(k)} = -F(\vec{x}^{(k)}), \tag{39}$$

for a vector $\vec{y}^{(k)}$ of variables, using LU decomposition (Luenberger 1973). We then compute:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{y}^{(k)}. \tag{40}$$

This reduces the time costs for large and sparse matrices, but there is little benefit for small and dense matrices. Therefore, for matrices smaller than dimension 2000, we

have applied instead the classical modified Newton method (Luenberger 1973). This means that the inverted Jacobian matrix is reused in $K$ iterations. More concretely, in Eq. 19 we reuse $JF(\vec{x}^{(k)})^{-1}$ instead of $JF(\vec{x}^{(k+1)})^{-1}$, …, $JF(\vec{x}^{(k+K)})^{-1}$, for each $k$ that is a multiple of $K+1$. In our experiments, we found that $K = 5$ was a good choice.

The implemented termination condition, both for Newton's method and for the fixed-point iteration method, is if two successive iterations provide the same values for $\vec{x}$. Non-termination due to oscillating values, caused by round-off errors, is avoided by never allowing a decrease of the values. It should be noted here that all vectors $\vec{x}^{(k)}$ computed by Newton's method are smaller than or equal to the desired solution, barring round-off errors. This is due to properties of the considered systems of equations, which make that each iteration of Newton's method represents a monotone operation, as proven by Kiefer et al. (2007).

## 5 Broyden's Method

The computation of the Jacobian matrix $JF(\vec{x}^{(k)})$ in Eq. 19 as well as its inversion and multiplication with $F(\vec{x}^{(k)})$ can be expensive operations, especially if the number of variables is large. Broyden's method, which is also called a quasi-Newton method, avoids any use of matrices, by approximating $-JF(\vec{x}^{(k)})^{-1} \cdot F(\vec{x}^{(k)})$ by a vector $\vec{s}^{(k)}$. In our implementation following Kelley (1995), this vector is computed on the basis of $F(\vec{x}^{(k)})$ and $\vec{s}^{(k')}$ with $k' < k$. Broyden's method can be seen as a generalization to multiple dimensions of the secant method, which replaces the first derivative of a function with a finite difference approximation.

As initial value $\vec{s}^{(0)}$, we take $-M \cdot F(\vec{x}^{(0)})$, where $M$ is a fixed matrix of the appropriate dimension that approximates $JF(\vec{x}^{(0)})^{-1}$. It is practical to take either $M = I$ or $M = -I$, where $I$ is the identity matrix. For the functions $F$ that we consider here, the diagonal of $JF(\vec{x}^{(0)})$ contains only negative values and therefore $M = -I$ is a reasonable choice. Consequently, we can obtain $\vec{s}^{(0)} = F(\vec{x}^{(0)})$ without matrix operations.

The computation of $\vec{s}^{(k+1)}$ for $k \geq 0$ can also be implemented without any matrix operations, at the expense of having a loop in each iteration $k$, which computes:

1. $\vec{x}^{(k+1)} \leftarrow \vec{x}^{(k)} + \vec{s}^{(k)}$;
2. $\vec{z} \leftarrow F(\vec{x}^{(k+1)})$;
3. for $j = 0, \ldots, k-1$ do:
   $\vec{z} \leftarrow \vec{z} + \vec{s}^{(j+1)} \cdot \dfrac{\vec{s}^{(j)} \cdot \vec{z}}{\vec{s}^{(j)} \cdot \vec{s}^{(j)}}$;
4. $\vec{s}^{(k+1)} \leftarrow \vec{z} \Big/ \left(1 - \dfrac{\vec{s}^{(k)} \cdot \vec{z}}{\vec{s}^{(k)} \cdot \vec{s}^{(k)}}\right)$.

(We let $\cdot$ with two vectors as arguments stand for the inner product.)

As each iteration is more expensive than the previous one due to the loop, it is common to 'restart' every $K$ iterations, that is, to repeat the above procedure after setting $k \leftarrow 0$ and $\vec{x}^{(0)} \leftarrow \vec{x}^{(K)}$. The number $K$ is fixed, and we have found $K = 20$ to be a good choice.

Values in $\vec{s}^{(k)}$ tend to contain both negative and positive values, letting the respective $\vec{x}^{(k)}$ move back and forth close to the root for many iterations. It is therefore more difficult to avoid oscillation and non-termination than in the case of the fixed-point method and Newton's method. The implemented termination conditions are: $|\vec{s}_i^{(k)}/\vec{x}_i^{(k+1)}| < 10^{-10}$ for all $i$ or the computation of $\vec{s}^{(k)} \cdot \vec{s}^{(k)}$ causes underflow.

Much as in the previous section, use of linking nonterminals can be separated from the mathematical treatment of the root-finding problem. This means that the length of vectors manipulated by Broyden's method is determined by the number of non-linking nonterminals. The linking nonterminals only appear internally in the evaluation of $F$.

## 6 Application to Intersection

In this section we consider the computation of the partition function for a PCFG obtained from the intersection of another PCFG and a finite automaton.

### 6.1 Finite Automata

To simplify the definitions in the next section, we consider only finite automata without epsilon transitions. Thus, a *finite automaton* (FA) is represented as a tuple $\mathcal{M} = (\Sigma, Q, q_I, E, T)$, where $\Sigma$ and $Q$ are two finite sets of terminals and states, respectively, $q_I \in Q$ is the initial state, $E \subseteq Q$ is the set of final states, and $T$ is a finite set of transitions, each of the form $s \overset{a}{\mapsto} t$, where $s, t \in Q$ and $a \in \Sigma$.

In what follows, symbols $s$ and $t$ range over the set $Q$, symbol $\tau$ ranges over the set $T$, and symbol $c$ ranges over the set $T^*$.

For a fixed FA $\mathcal{M}$, we define a *configuration* to be an element of $Q \times \Sigma^*$. We also define the relation $\vdash$ on triples consisting of two configurations and a transition $\tau \in T$ by: $(s, w) \overset{\tau}{\vdash} (t, w')$ if and only if $w$ is of the form $aw'$, for some $a \in \Sigma$, and $\tau = (s \overset{a}{\mapsto} t)$. We say $\mathcal{M}$ *recognizes* string $w$ if $(q_I, w) \overset{\tau_1}{\vdash} (s_1, w_1) \overset{\tau_2}{\vdash} \cdots \overset{\tau_m}{\vdash} (s_m, w_m)$, some $s_i$, $w_i$ and $\tau_i$ $(1 \leq i \leq m)$, where $s_m \in E$ and $w_m = \varepsilon$. We then call the string $c = \tau_1 \cdots \tau_m$ a *computation* for $w$. The automaton recognizes string $\varepsilon$ if $q_I \in E$ by taking computation $c = \varepsilon$. The language $L(\mathcal{M})$ accepted by $\mathcal{M}$ is the set of all recognized strings. We say a FA is *non-ambiguous* if there is at most one computation $c$ for each string. We say a FA is *deterministic* if there is at most one transition $s \overset{a}{\mapsto} t$ for each $s$ and $a$. Determinism is a sufficient condition for non-ambiguity.

The size $|\mathcal{M}|$ of a FA $\mathcal{M}$ is defined to be $|T|+|E|$. We assume that a finite automaton does not contain useless states, which implies that $|\mathcal{M}| \geq |Q|$.

### 6.2 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. We extend this by letting the input grammar be a probabilistic CFG (although the finite automaton is still discrete).

We will refer to this construction as *weighted intersection*, as it is applicable to a pair consisting of a weighted CFG and a weighted FA; we will not discuss the construction in its fullest generality, however, since only the probabilistic case is needed here.

For a PCFG $\mathcal{G} = (\Sigma, N, S, R, p)$ and a FA $\mathcal{M} = (\Sigma, Q, q_I, E, T)$ with the same set of terminals, we construct a new PCFG $\mathcal{G}_\cap = (\Sigma, N_\cap, S_\cap, R_\cap, p_\cap)$, where $N_\cap = \{S_\cap\} \cup Q \times (\Sigma \cup N) \times Q$, with $S_\cap$ a new symbol, and $R_\cap$ is the set of rules that is obtained as follows.

- For each $A \to X_1 \cdots X_n$ in $R$ and each sequence $s_0, \ldots, s_n \in Q$, with $n \geq 0$, let $(s_0, A, s_n) \to (s_0, X_1, s_1) \cdots (s_{n-1}, X_n, s_n)$ be in $R_\cap$; if $n = 0$, the new rule is of the form $(s_0, A, s_0) \to \epsilon$. Function $p_\cap$ assigns to the new rule the same probability assigned by $p$ to the original rule.
- For each $s \stackrel{a}{\mapsto} t$ in $T$, let $(s, a, t) \to a$ be in $R_\cap$. Function $p_\cap$ assigns probability 1 to this rule.
- For each final state $s \in E$, let $S_\cap \to (q_I, S, s)$ be in $R_\cap$. Function $p_\cap$ assigns probability 1 to this rule.

In the general case, grammar $\mathcal{G}_\cap$ contains useless rules and nonterminals. These can be removed in linear time by a process called *reduction* (Hopcroft and Ullman 1979). We assume that the intersection grammar $\mathcal{G}_\cap$ is reduced before it is subjected to further operations, in particular, the computation of the partition function.

PCFGs considered in practice are often *proper*, which means that for each nonterminal $A$, the sum of probabilities of all rules with $A$ as left-hand side is 1. Note that even if $\mathcal{G}$ is proper, $\mathcal{G}_\cap$ in general is not, and this sum may be smaller than one or greater than one.

If $\mathcal{M}$ is non-ambiguous, then $p_\cap(w) = p(w)$ if $w$ is recognized by $\mathcal{M}$ and $p_\cap(w) = 0$ otherwise. This and other formal properties of weighted intersection are investigated by Nederhof and Satta (2003).

The first clause above can at worst contribute the quantity $|\mathcal{G}| \cdot |\mathcal{M}|^r$ to the size of the intersection grammar $\mathcal{G}_\cap$, where $r$ is the length of the longest rule in $R$. As $r$ may be very large in practical cases, the intersection grammar may grow very large. This is certainly the case for a common PCFG reported in Sect. 7. It is therefore useful if not essential to turn the source grammar $\mathcal{G}$ into binary form before the intersection, that is, to transform the grammar to let $r = 3$, preserving the probabilities of strings, and allowing a straightforward mapping from derivations in the transformed grammar to derivations in the original grammar. If $\mathcal{G}$ is so transformed before intersection with $\mathcal{M}$, the size of the intersection grammar becomes linear in $|\mathcal{G}|$ and cubic in $|\mathcal{M}|$, where $\mathcal{G}$ is the binarized grammar.

Note that we do not need to transform to Chomsky normal form (Harrison 1978; Huang and Fu 1971), and a simpler procedure suffices. Here we will apply a transformation that replaces each rule $A \to \alpha$ such that $|\alpha| > 2$ with the $|\alpha| - 1$ rules below, involving new nonterminals of the form $[\beta]$, where $\beta$ is a proper suffix of $\alpha$ of length 2 or more.

- $A \to X\ [\beta]$ with the same probability as $A \to \alpha$, where $X\beta = \alpha$,
- $[X\beta] \to X\ [\beta]$ with probability 1, where $X\beta$ is a suffix of $\alpha$ of length 3 or more, and
- $[XY] \to X\ Y$ with probability 1, where $XY$ is the suffix of $\alpha$ of length 2.

If the right-hand sides of two distinct rules $A_1 \to \alpha_1$ and $A_2 \to \alpha_2$ share a proper suffix $\beta$, there is a single new rule with left-hand side $[\beta]$.

### 6.3 Determining Linking Nonterminals

Let us assume from this point onwards that the PCFG at hand was obtained by binarization, followed by intersection with a finite automaton. Two observations are relevant. First, the set of nonterminals introduced by binarization, following the procedure above, is a valid set of linking nonterminals. In particular, the bound $b$ mentioned in Sect. 4.3 is determined by the maximum length of a right-hand side of a rule in the grammar before binarization. To see this, consider a sequence of newly introduced nonterminals $[\alpha]$, for suffixes $\alpha$ of a right-hand side of a rule, where $|\alpha|$ is decreasing.

Second, if we compute a new PCFG from a binarized PCFG and a finite automaton using intersection, then the obtained set of nonterminals of the form $(s, [\alpha], t)$ also constitutes a valid set of linking nonterminals, with the same bound $b$. This follows from the way in which parts of the structure of the intersection grammar are copied from the source grammar.

Experiments have revealed that sets of linking nonterminals determined in this way are almost identical to those obtained by the general algorithm from Sect. 4.3. In the experiments in Sect. 7, linking nonterminals were determined as side-effect of the binarization process as described above.

### 6.4 Example

We consider the following PCFG:

$$\begin{array}{lll} S \to S\ A\ S & [0.2] \\ S \to a & [0.8] \\ A \to A\ A & [0.4] \\ A \to a & [0.5] \\ A \to b & [0.1] \end{array}$$

Binarization results in:

$$\begin{array}{lll} S \to S\ [AS] & [0.2] \\ S \to a & [0.8] \\ [AS] \to A\ S & [1] \\ A \to A\ A & [0.4] \\ A \to a & [0.5] \\ A \to b & [0.1] \end{array}$$

We take $[AS]$ to be the only linking nonterminal.

The intersection of the binarized grammar and the FA from Fig. 1 is given in Fig. 2, after reduction. There are two sets of mutually recursive nonterminals, viz. $\{(s_1, A, s_0), (s_1, A, s_1), (s_0, A, s_1), (s_0, A, s_0)\}$ and $\{(s_0, S, s_1), (s_1, [AS], s_1)\}$.

The former of these two sets is treated first. Let us order the four nonterminals as $A_1 = (s_1, A, s_0)$, $A_2 = (s_1, A, s_1)$, $A_3 = (s_0, A, s_1)$, $A_4 = (s_0, A, s_0)$. The objective is to find the smallest non-negative solution to the system:

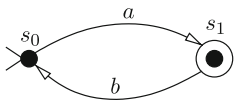


**Fig. 1** Finite automaton in the example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $S_\cap$ | $\to$ | $(s_0, S, s_1)$ | [1] | $S_\cap$ | $\to$ | $B_1$ | [1] |
| $(s_0, S, s_1)$ | $\to$ | $(s_0, S, s_1)\ (s_1, [AS], s_1)$ | [0.2] | $B_1$ | $\to$ | $B_1\ C$ | [0.2] |
| $(s_0, S, s_1)$ | $\to$ | $(s_0, a, s_1)$ | [0.8] | $B_1$ | $\to$ | $A'$ | [0.8] |
| $(s_1, [AS], s_1)$ | $\to$ | $(s_1, A, s_0)\ (s_0, S, s_1)$ | [1] | $C$ | $\to$ | $A_1\ B_1$ | [1] |
| $(s_1, A, s_0)$ | $\to$ | $(s_1, A, s_0)\ (s_0, A, s_0)$ | [0.4] | $A_1$ | $\to$ | $A_1\ A_4$ | [0.4] |
| $(s_1, A, s_0)$ | $\to$ | $(s_1, A, s_1)\ (s_1, A, s_0)$ | [0.4] | $A_1$ | $\to$ | $A_2\ A_1$ | [0.4] |
| $(s_1, A, s_0)$ | $\to$ | $(s_1, b, s_0)$ | [0.1] | $A_1$ | $\to$ | $B'$ | [0.1] |
| $(s_1, A, s_1)$ | $\to$ | $(s_1, A, s_0)\ (s_0, A, s_1)$ | [0.4] | $A_2$ | $\to$ | $A_1\ A_3$ | [0.4] |
| $(s_1, A, s_1)$ | $\to$ | $(s_1, A, s_1)\ (s_1, A, s_1)$ | [0.4] | $A_2$ | $\to$ | $A_2\ A_2$ | [0.4] |
| $(s_0, A, s_1)$ | $\to$ | $(s_0, A, s_0)\ (s_0, A, s_1)$ | [0.4] | $A_3$ | $\to$ | $A_4\ A_3$ | [0.4] |
| $(s_0, A, s_1)$ | $\to$ | $(s_0, A, s_1)\ (s_1, A, s_1)$ | [0.4] | $A_3$ | $\to$ | $A_3\ A_2$ | [0.4] |
| $(s_0, A, s_1)$ | $\to$ | $(s_0, a, s_1)$ | [0.5] | $A_3$ | $\to$ | $A'$ | [0.5] |
| $(s_0, A, s_0)$ | $\to$ | $(s_0, A, s_0)\ (s_0, A, s_0)$ | [0.4] | $A_4$ | $\to$ | $A_4\ A_4$ | [0.4] |
| $(s_0, A, s_0)$ | $\to$ | $(s_0, A, s_1)\ (s_1, A, s_0)$ | [0.4] | $A_4$ | $\to$ | $A_3\ A_1$ | [0.4] |
| $(s_0, a, s_1)$ | $\to$ | $a$ | [1] | $A'$ | $\to$ | $a$ | [1] |
| $(s_1, b, s_0)$ | $\to$ | $b$ | [1] | $B'$ | $\to$ | $b$ | [1] |

**Fig. 2** Intersection grammar in the example, with original nonterminal names on the left. The same rules with nonterminal names consistently renamed, for ease of reference, are given on the right

$$
\begin{aligned}
0 &= f_1(x_1, x_2, x_3, x_4) = -x_1 + 0.4 \cdot x_1 \cdot x_4 + 0.4 \cdot x_2 \cdot x_1 + 0.1 \\
0 &= f_2(x_1, x_2, x_3, x_4) = -x_2 + 0.4 \cdot x_1 \cdot x_3 + 0.4 \cdot x_2 \cdot x_2 \\
0 &= f_3(x_1, x_2, x_3, x_4) = -x_3 + 0.4 \cdot x_4 \cdot x_3 + 0.4 \cdot x_3 \cdot x_2 + 0.5 \\
0 &= f_4(x_1, x_2, x_3, x_4) = -x_4 + 0.4 \cdot x_4 \cdot x_4 + 0.4 \cdot x_3 \cdot x_1
\end{aligned}
$$

As no linking nonterminals are involved here, we only need to look at the equations from Sect. 4.2, to derive for example:

$$
\begin{aligned}
df_{1,1}(\vec{x}) &= -\delta_{1,1} + p(A_1 \to A_1 A_4) \cdot df_{A_1 A_4, 1}(\vec{x}) \\
&\quad + p(A_1 \to A_2 A_1) \cdot df_{A_2 A_1, 1}(\vec{x}) \\
&\quad + p(A_1 \to B') \cdot df_{B', 1}(\vec{x}) && (41) \\
&= -1 + 0.4 \cdot (x_1 \cdot df_{A_4,1} + \delta_{1,1} \cdot x_4) \\
&\quad + 0.4 \cdot (x_2 \cdot df_{A_1,1} + \delta_{2,1} \cdot x_1) \\
&\quad + 0.1 \cdot 0 && (42) \\
&= -1 + 0.4 \cdot (x_1 \cdot 0 + 1 \cdot x_4) + 0.4 \cdot (x_2 \cdot 1 + 0 \cdot x_1) && (43) \\
&= -1 \ + \ 0.4 \cdot x_4 \ + \ 0.4 \cdot x_2 && (44) \\
df_{1,2}(\vec{x}) &= -\delta_{1,2} + 0.4 \cdot (x_1 \cdot df_{A_4,2} + \delta_{1,2} \cdot x_4) \\
&\quad + 0.4 \cdot (x_2 \cdot df_{A_1,2} + \delta_{2,2} \cdot x_1) && (45) \\
&= 0.4 \cdot (x_1 \cdot 0 + 0 \cdot x_4) + 0.4 \cdot (x_2 \cdot 0 + 1 \cdot x_1) \ = \ 0.4 \cdot x_1 && (46)
\end{aligned}
$$

Together with the remaining of the 16 expressions $df_{i,j}(\vec{x})$, $1 \leq i, j \leq 4$, we obtain:

$$JF(x_1, x_2, x_3, x_4) = \begin{pmatrix} -1 + 0.4 \cdot x_4 + 0.4 \cdot x_2 & 0.4 \cdot x_1 & 0 & 0.4 \cdot x_1 \\ 0.4 \cdot x_3 & -1 + 0.8 \cdot x_2 & 0.4 \cdot x_1 & 0 \\ 0 & 0.4 \cdot x_3 & -1 + 0.4 \cdot x_4 + 0.4 \cdot x_2 & 0.4 \cdot x_3 \\ 0.4 \cdot x_3 & 0 & 0.4 \cdot x_1 & -1 + 0.8 \cdot x_4 \end{pmatrix} \tag{47}$$

Application of Newton's method results in the values $\tilde{Z}(A_1)$, $\tilde{Z}(A_2)$, $\tilde{Z}(A_3)$, $\tilde{Z}(A_4)$.

We now turn to the set $\{(s_0, S, s_1), (s_1, [AS], s_1)\}$. If we let $C = (s_1, [AS], s_1)$ act as linking nonterminal, then the manipulated matrices have dimension 1, where the sole entry corresponds to nonterminal $B_1 = (s_0, S, s_1)$.

Involving the equations from Sect. 4.3, we now obtain:

$$\begin{aligned} df_{1,1}(\vec{x}) &= -\delta_{1,1} + 0.2 \cdot (x_1 \cdot df_{C,1}(\vec{x}) + \delta_{1,1} \cdot f_C(\vec{x})) && (48) \\ &= -1 + 0.2 \cdot (x_1 \cdot df_{C,1}(\vec{x}) + f_C(\vec{x})), && (49) \end{aligned}$$

where

$$\begin{aligned} df_{C,1}(\vec{x}) &= p(C \to A_1 B_1) \cdot df_{A_1 B_1, 1}(\vec{x}) && (50) \\ &= 1 \cdot \tilde{Z}(A_1) \cdot df_{B_1,1}(\vec{x}) \;=\; \tilde{Z}(A_1) && (51) \\ f_C(\vec{x}) &= p(C \to A_1 B_1) \cdot \tilde{Z}(A_1) \cdot x_1 \;=\; \tilde{Z}(A_1) \cdot x_1 && (52) \end{aligned}$$

Note that it is an essential element of our approach that the right-hand sides of Eqs. 51 and 52 are *not* substituted in the right-hand side of Eq. 49, which in the general case avoids combinatorial explosions for long right-hand sides in the original grammar.

Application of Newton's method provides the value $\tilde{Z}(B_1)$, after which the main value of interest $\tilde{Z}(S_\cap) = p(S_\cap \to B_1) \cdot \tilde{Z}(B_1) = \tilde{Z}(B_1)$ is computed.

## 7 Experimental Results

We first investigate three artificial examples, to illustrate some difficulties underlying the fixed-point method, and we then look at performance for grammars used in practice. We have implemented the algorithms in C++, on a PC with a 3.4 GHz Pentium D processor.

The experiments were performed with floating-point numbers of type 'double' (8 bytes on our machine).

### 7.1 Artificial Grammars

Consider the family of PCFGs given by the following rules with indicated probabilities:

$$\begin{aligned} S &\to S\ S \quad [\tfrac{1}{2} - \tfrac{1}{2^i}] \\ S &\to a \quad\ [\tfrac{1}{2} + \tfrac{1}{2^i}], \end{aligned} \tag{53}$$

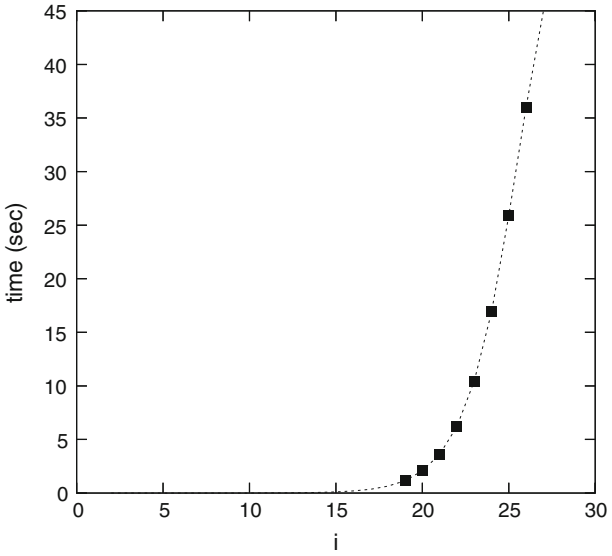


**Fig. 3** Running time of the fixed-point method for the family of grammars in (53)

with $i$ from 2 to 27. The value $Z(S)$ is one for each choice of $i$, but $Z_k(S)$ quickly decreases for increasing $i$, and thereby the fixed-point method becomes very slow, as illustrated by Fig. 3. For $i = 27$, 45 s are required, for more than $1.47 \cdot 10^8$ iterations.

This contrasts with Newton's method, which for $i = 27$ requires $2.7 \cdot 10^{-5}$ s, or 28 iterations. Broyden's method requires $1.0 \cdot 10^{-4}$ s, or 41 iterations. Upon halting, the distance of the computed value to the true fixed-point $Z(S) = 1$ is slightly smaller for Newton's method than for the fixed-point method, for all $i$. For $i = 27$ this is $1.2 \cdot 10^{-9}$ (Newton's method) versus $4.8 \cdot 10^{-9}$ (fixed-point method); Broyden's method is less precise than either, with distance $5.9 \cdot 10^{-9}$.

The fixed-point method is also sensitive to a combination of cycles in the grammar, which is illustrated by the following family of (linear) PCFGs:[1]

$$\begin{array}{lll} A_1 & \to A_2 & \left[\frac{1}{2}\right] \\ A_1 & \to A_1 & \left[\frac{1}{2}\right] \\ A_2 & \to A_3 & \left[\frac{1}{2}\right] \\ A_2 & \to A_1 & \left[\frac{1}{2}\right] \\ & \vdots & \\ A_{i-1} & \to A_i & \left[\frac{1}{2}\right] \\ A_{i-1} & \to A_1 & \left[\frac{1}{2}\right] \\ A_i & \to \varepsilon & [1], \end{array} \tag{54}$$

with $i$ from 1 to 19. The start symbol is $A_1$.

[1] This example is due to Kousha Etessami and Mihalis Yannakakis (personal communication), with whose kind permission it is presented here.

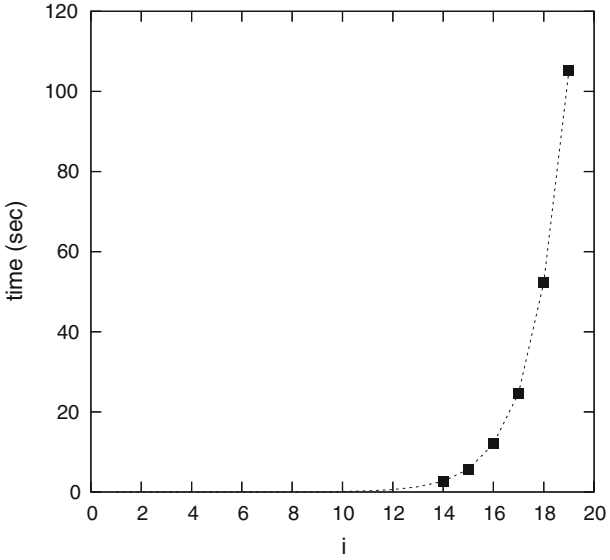


**Fig. 4** Running time of the fixed-point method for the family of grammars in (54)

Figure 4 shows that for increasing values of $i$, there is a steep increase of the running time of the fixed-point method. In fact, analytical arguments indicate that the number of iterations required to reach a given accuracy grows exponentially in $i$. For $i = 19$, the fixed-point method requires $1.2 \cdot 10^7$ iterations, or 105 s, whereas Newton's method throughout requires two iterations, or $1.0 \cdot 10^{-4}$ s for $i = 19$. In fact, a single iteration would be sufficient, as Newton's method here effectively solves a linear system of equations during the first iteration, which already yields the required values. A second iteration is performed in our implementation before it is detected that the computed values are stable. Newton's method throughout computes the exact value $Z(A_1) = 1$ for all bits in the binary representation, whereas for the fixed-point method, the difference is as great as $1.9 \cdot 10^{-11}$ for $i = 19$. Broyden's method throughout requires $i + 1$ iterations, or $4.6 \cdot 10^{-4}$ s for $i = 19$, and computes the exact value.

A similar phenomenon can also be achieved with a fixed underlying CFG and probabilities of increasing bit-size, as in the following family of PCFGs:

$$\begin{aligned} A_1 &\to A_2 \ [\tfrac{1}{2^i}] \\ A_1 &\to A_1 \ [1 - \tfrac{1}{2^i}] \\ A_2 &\to \varepsilon \quad [1], \end{aligned} \tag{55}$$

with $i$ increasing from 1. Again, the fixed-point method showed exponential behaviour in the experiments (figure omitted), whereas Newton's method requires two iterations throughout, and Broyden's method three iterations. Once more, the exponential behaviour is easily explained by analytical arguments.

### 7.2 WSJ Corpus

Experiments were performed on the Wall Street Journal (WSJ) corpus, from the Penn Treebank version II. During its development, the software was tested on the basis of Sect. 22, and the measurements presented here involve sections 02–21. First a context-free grammar was derived from the 'stubs' of the combined trees, taking parts of speech as leaves of the trees, omitting all affixes from the nonterminal names. Such preprocessing of the WSJ corpus is consistent with earlier work that used CFGs derived from the same corpus, as e.g. by Johnson (1998).

For the first experiment, we have preserved $\varepsilon$-generating subtrees, which leads to a CFG with 10,266 rules, 28 nonterminals and 36 terminals. The probabilities of the PCFG were obtained by relative frequency estimation, without smoothing. As proven by Chi and Geman (1998), the partition function of a PCFG obtained in this way assigns one to all nonterminals.

We binarized the CFG, leading to 15,052 rules and 4,814 nonterminals. We then computed the sum of probabilities of $\varepsilon$-generating derivations for each nonterminal of the original grammar. As explained in the introduction, this is done by letting a given nonterminal be the start symbol, and taking the intersection with an automaton recognizing only the empty string. The partition function applied to the start symbol of the intersection grammar is the required value. For only 5 of the 28 nonterminals, the empty string cannot be generated, and thereby the determined value is 0. The highest value, namely 0.218, is obtained for nonterminal WHADVP.

The fixed-point method takes 0.30 s for all nonterminals together, whereas Newton's method takes 0.11 s and Broyden's method takes 0.08 s, not including the time costs of intersection, which is of course identical for the three methods. The application of Newton's method and Broyden's method was combined with the concept of linking nonterminals, as explained in Sect. 4. The highest number of iterations per maximal set of mutually recursive nonterminals is 20 for the fixed-point method, six for Newton's method, and eight for Broyden's method. The values produced by the three methods agree in the first 13 digits of the significant for all nonterminals.

Although the difference in performance between the three methods is small, we feel the above results are noteworthy because they are one argument in support of our thesis that the computation of sums of derivations is a problem of practical relevance that can be effectively solved in many interesting cases.

For the following experiments, in which we estimate infix probabilities, we constructed a PCFG as above, with the exception that $\varepsilon$-generating subtrees were removed from the corpus before construction of the PCFG. This leads to 10,035 rules and 28 nonterminals before binarization and 14,676 rules and 4,669 nonterminals after binarization. We have taken 50 strings of length 6, which were generated randomly, giving each of the 36 terminals an equal probability. To give an impression of the order of magnitude of the infix probabilities, the first five strings are given in Table 1, together with their infix probabilities.

In order to investigate the impact of the length of infixes on the running time of the method, we have also considered the prefixes of length 2–6 of the 50 random strings.

The running times are presented in Fig. 5. Newton's method and Broyden's method improve on the fixed-point method most convincingly for small problem sizes. For

**Table 1** The first 5 of the 50 random strings used in the experiment, together with their estimated infix probabilities

| Random string | Infix probability |
|---|---|
| FW NNPS JJS JJS SYM NNS | $2.11 \cdot 10^{-19}$ |
| CC MD JJ WP VBZ VB | $5.01 \cdot 10^{-11}$ |
| WP PDT IN RBR DT VBP | $2.59 \cdot 10^{-15}$ |
| IN NNPS POS NN WP$ NNS | $1.89 \cdot 10^{-10}$ |
| WP RB WP VB JJ WP$ | $1.93 \cdot 10^{-12}$ |

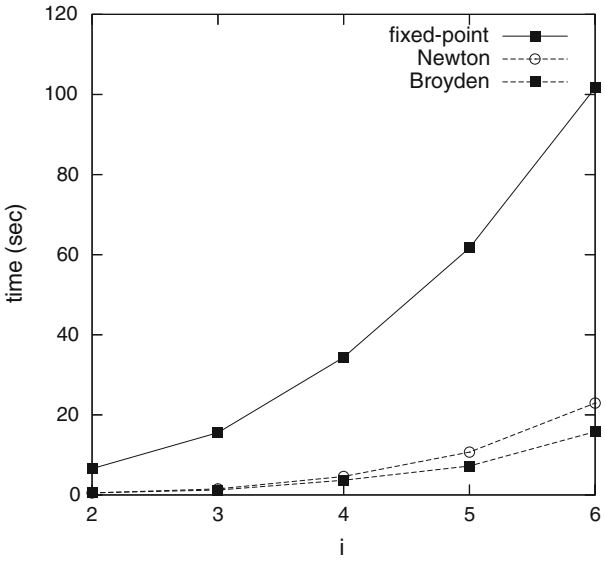


**Fig. 5** Running time of the methods, against the length $i$ of infixes

infixes of length 2, the fixed-point method requires on the average 6.57 s, whereas Broyden's method requires 0.52 s, an improvement by a factor of more than 12. For infixes of length 6, this factor is still more than 6. On the average, the probabilities computed by the three methods agree in more than nine decimal digits, for all infix lengths.

In our last experiment, we have compared Newton's method and Broyden's method with the technique involving linking nonterminals to the same methods without that technique. As we found unsurmountable difficulties with large problem sizes, we have looked only at infixes of length 3, with different grammar sizes. The grammar size was varied by forming the PCFG on the basis of sections 02 to $i$ of the WSJ corpus, where $i$ ranges from 05 to 12. The running time is given in Table 2. We also give the running time of the fixed-point method as the base line. Again, no significant differences in accuracy were observed between the respective methods.

The data show that there is more than one order of magnitude between the performance of Newton's method with and without the technique involving linking nonterminals. In fact, without this technique, Newton's method performs far worse than

**Table 2** The number of rules in the grammar before binarization, and the running time of the fixed-point method, and of Newton's method and Broyden's method with and without distinguishing linking nonterminals, for different subsets of the WSJ corpus

| WSJ sections | $\|R\|$ | Fixed-point (s) | Newton (s) | | Broyden (s) | |
|---|---|---|---|---|---|---|
| | | | Link | No link | Link | No link |
| 02–05 | 4277 | 6.2 | 0.6 | 12.1 | 0.5 | 0.8 |
| 02–06 | 4765 | 6.8 | 0.6 | 16.4 | 0.5 | 0.9 |
| 02–07 | 5326 | 7.7 | 0.7 | 17.9 | 0.6 | 1.1 |
| 02–08 | 5437 | 7.9 | 0.7 | 22.8 | 0.6 | 1.1 |
| 02–09 | 5904 | 8.8 | 0.8 | 31.2 | 0.7 | 1.2 |
| 02–10 | 6289 | 9.9 | 1.0 | 48.0 | 0.8 | 1.5 |
| 02–11 | 6738 | 10.6 | 1.0 | 38.6 | 0.9 | 1.7 |
| 02–12 | 7116 | 11.5 | 1.1 | 41.0 | 1.0 | 2.5 |

the fixed-point method. In order to demonstrate in which way the high costs of matrix operations are responsible for this, we look closer at one substring, which we arbitrarily take to be "FW NNPS JJS", a prefix of the first substring from Table 1, in combination with sections 02–05 of the WSJ corpus. The largest maximal set of mutually recursive nonterminals contains 5741 nonterminals. Without distinguishing linking nonterminals, the relevant matrix has dimension 5741. As discussed in Sect. 4.4, one step of Newton's method can be done by matrix inversion or by solving a sparse system of equations. In both cases, the time costs are a little over 1 s here.

By distinguishing between 186 non-linking and 5555 linking nonterminals, matrix inversion now takes about 0.02 s, and construction of the matrix before inversion becomes the greatest cost factor and takes about 0.06 s. The number of iterations per set of nonterminals is comparable for Newton's method with or without distinguishing linking nonterminals.

As Broyden's method does not manipulate matrices, there is a much smaller difference between the performance with or without the technique involving linking nonterminals. The technique still has a clear beneficial influence on the performance however, with a speed-up exceeding a factor of 2.5 for the largest problem size considered here.

Experiments with Newton's method applied on NLP grammars have been reported before, by Wojtczak and Etessami (2007). However, they considered grammars directly extracted from treebanks, as opposed to grammars resulting from intersection with FAs. Consequently, the problem sizes they obtained were of a considerably smaller magnitude.

## 8 Conclusions

We have discussed applications of partition functions of PCFGs, and have considered the fixed-point method, Newton's method, and Broyden's method for their computation. We have argued theoretically that the fixed-point method suffers from a number of problems with time complexity, and we have shown using practical experiments that it can be considerably slower than Newton's method and Broyden's method.

However, it should also be observed that for large problem sizes, the advantages of Newton's method over the fixed-point method may become weaker, due to the high costs of matrix inversion. This problem may be avoided by Broyden's method, which does not manipulate matrices. Although it typically requires more iterations than Newton's method, each iteration commonly has lower time costs.

Our version of Newton's method distinguishes between two types of variables, a first set that captures the recursive structure in a system of equations, and a second set of variables that mediate between variables in the first set without contributing to recursion on their own. As far as we know, this is a novel variant of Newton's method, which is particularly effective for probabilistic context-free grammars, as we have shown.

## References


Abney, S., McAllester, D., & Pereira, F. (1999). Relating probabilistic grammars and automata. In *37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, Maryland, USA, pp. 542–549.

Baker, J. (1979). Trainable grammars for speech recognition. In J. Wolf & D. Klatt (Eds.), *Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America*, pp. 547–550.

Bar-Hillel, Y., Perles, M., & Shamir, E. (1964). On formal properties of simple phrase structure grammars. In Y. Bar-Hillel (Ed.), *Language and information: Selected essays on their theory and application* (Chap. 9, pp. 116–150). Reading, Massachusetts: Addison-Wesley.

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

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

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

Cormen, T., Leiserson, C., & Rivest, R. (1990). *Introduction to algorithms*. The MIT Press.

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

Fred, A. (2000). Computation of substring probabilities in stochastic grammars. In A. Oliveira (Ed.), *Grammatical inference: Algorithms and applications, Vol. 1891 of Lecture Notes in Artificial Intelligence* (pp. 103–114). Springer-Verlag.

Harrison, M. (1978). *Introduction to formal language theory*. Addison-Wesley.

Hopcroft, J., & Ullman, J. (1979) *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Huang, T., & Fu, K. (1971). On stochastic context-free languages. *Information Sciences, 3*, 201–224.

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

Johnson, M. (1998). PCFG models of linguistic tree representations. *Computational Linguistics, 24*(4), 613–632.

Kelley, C. (1995). *Iterative methods for linear and nonlinear equations*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Kiefer, S., Luttenberger, M., & Esparza, J. (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*, pp. 217–266.

Luenberger, D. (1973). *Introduction to linear and nonlinear programming*. Addison-Wesley.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning*, Taipei, Taiwan, pp. 49–55.

Manning, C., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

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

Nederhof, M.-J., & Satta, G. (2006). Estimation of consistent probabilistic context-free grammars. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, New York, USA, pp. 343–350.

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

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

Thompson, R. (1974). Determination of probabilistic grammars for functionally specified probability-measure languages. *IEEE Transactions on Computers, C-23*, 603–614.

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., & Carrasco, R. (2005). Probabilistic finite-state machines—part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(7), 1026–1039.

Wojtczak, D., & Etessami, K. (2007). PReMo: An analyzer for probabilistic recursive models. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, Vol. 4424 of Lecture Notes in Computer Science*, Braga, Portugal (pp. 66–71). Springer-Verlag.