Probabilistic Parsing

Mark-Jan Nederhof[1] and Giorgio Satta[2]

[1] School of Computer Science, University of St Andrews, North Haugh, St Andrews, Fife, KY16 9SX, Scotland, http://www.cs.st-andrews.ac.uk/~mjn

[2] Department of Information Engineering, University of Padua, via Gradenigo, 6/A, I-35131 Padova, Italy, satta@dei.unipd.it

1 Introduction

A paper in a previous volume [1] explained *parsing*, which is the process of determining the parses of an input string according to a formal grammar. Also discussed was *tabular parsing*, which solves the task of parsing in polynomial time by a form of dynamic programming. In passing, we also mentioned that parsing of input strings can be easily generalised to parsing of finite automata.

In applications involving natural language, the set of parses for a given input sentence is typically very large. This is because formal grammars often fail to capture subtle properties of structure, meaning and use of language, and consequently allow many parses that humans would not find plausible.

In natural language systems, parsing is commonly one stage of processing amongst several others. The effectiveness of the stages that follow parsing generally relies on having obtained a small set of preferred parses, ideally only one, from amongst the full set of parses. This is called (syntactic) *disambiguation*. There are roughly two ways to achieve this. First, some kind of filter may be applied to the full set of parses, to reject all but a few. This filter may look at the meanings of words and phrases, for example, and may be based on linguistic knowledge that is very different in character from the grammar that was used for parsing.

A second approach is to augment the parsing process so that weights are attached to parses and subparses. The higher the weight of a parse or subparse, the more confident we are that it is correct. This is called *weighted parsing*. If the weights are chosen to define a probability distribution over parses or strings, this may also be called *probabilistic parsing*. Disambiguation is achieved by computing the parse with the highest weight or, where appropriate, highest probability.

The simplest form of probabilistic parsing relies on an assignment of probabilities to individual rules from a context-free grammar. These probabilities are then multiplied upon combination of rules to form parses. Models that are close to this basic idea, such as [2, 3], have been highly influential from the 1990s onward. The success of probabilistic parsing is due to its flexibility and scalability, in contrast to approaches to disambiguation that rely on much deep knowledge of language. For general discussions about statistical natural language processing see [4–6].

In Section 2 we discuss both weighted and probabilistic context-free grammars. We investigate intersection of weighted context-free grammars and finite automata in Section 3. By normalisation, discussed in Section 4, it can be shown that for the sake of disambiguation we may restrict our attention to probabilistic context-free grammars. Parsing is treated in Section 5, and how the probabilities of grammar rules can be obtained empirically is explained in Section 6.

Section 7 discusses the computation of prefix probabilities, and probabilistic push-down automata are the subject of Section 8. By considering semirings, a number of computations involving context-free grammars and push-down automata can be unified, as demonstrated in Section 9. We end with additional bibliographic remarks in Section 10.

2 Weighted and probabilistic context-free grammars

A *weighted context-free grammar* (WCFG) $\mathcal{G}$ is a 5-tuple (Σ, N, S, R, μ), where (Σ, N, S, R) is a context-free grammar and μ is a mapping from rules in R to positive real numbers. We refer to these numbers as *weights*, and they should be thought of as a measure of the desirability of using the corresponding rules. In general, a rule with a high weight is preferred over one with a low weight.

Let $d = \pi_1 \cdots \pi_m \in R^*$ be a string of rules (or alternatively, of labels that uniquely identify rules), and let α and β be strings of grammar symbols. The expression $\alpha \stackrel{d}{\Rightarrow} \beta$ means that β can be obtained from α by a left-most derivation in m steps, and the i-th step replaces the left-most nonterminal A_i by γ_i according to rule $\pi_i = (A_i \rightarrow \gamma_i)$. All derivations in this paper are assumed to be left-most. If $S \stackrel{d}{\Rightarrow} w$, we define the *yield* of d as $y(d) = w$.

We now define $\mu(\alpha \stackrel{d}{\Rightarrow} \beta)$ to be $\prod_{i=1}^{m} \ \mu(\pi_i)$ if $\alpha \stackrel{d}{\Rightarrow} \beta$ holds and to be 0 otherwise. In words, if the expression $\alpha \stackrel{d}{\Rightarrow} \beta$ denotes a valid left-most derivation, we compute the product of the weights of the used rules, and otherwise we take 0. This notation allows us to define the weight of a string w as:

$$\mu(w) = \sum_{d} \ \mu(S \stackrel{d}{\Rightarrow} w). \tag{1}$$

In words, to obtain the weight of a string we sum the weights of all left-most derivations of that string. For choices of d such that $S \stackrel{d}{\Rightarrow} w$ does not denote a valid left-most derivation, nothing is contributed to the sum. If $S \stackrel{d}{\Rightarrow} w$ holds, we also write $\mu(d)$ in place of $\mu(S \stackrel{d}{\Rightarrow} w)$.

Example 1. In the grammar below, the rules are labelled by names π_i and the weights are the numbers between brackets.

$$\begin{array}{ll} \pi_1 : S \rightarrow A \ A & (3) \\ \pi_2 : S \rightarrow a \ a & (1) \\ \pi_3 : A \rightarrow a & (2) \end{array}$$

This grammar is ambiguous, as there are two left-most derivations of aa, namely $S \overset{\pi_1}{\Rightarrow} AA \overset{\pi_3}{\Rightarrow} aA \overset{\pi_3}{\Rightarrow} aa$ with weight $\mu(\pi_1) \cdot \mu(\pi_3) \cdot \mu(\pi_3) = 3 \cdot 2 \cdot 2 = 12$ and $S \overset{\pi_2}{\Rightarrow} aa$ with weight $\mu(\pi_2) = 1$. The weight of aa is therefore $\mu(aa) = \mu(S \overset{\pi_1\pi_3\pi_3}{\Rightarrow} aa) + \mu(S \overset{\pi_2}{\Rightarrow} aa) = 12 + 1 = 13$.

We say a WCFG is *convergent* if $\sum_{d,w} \mu(S \overset{d}{\Rightarrow} w)$ is a finite number. A WCFG can be called a *probabilistic context-free grammar* (PCFG) if μ maps all rules to numbers no greater than 1 [7–10]. Where we are dealing with PCFGs, we will often replace the name μ of the weight assignment by p.

We say a WCFG is *proper* if for every nonterminal A:

$$\sum_{\pi=(A\rightarrow\alpha)} \mu(\pi) = 1. \tag{2}$$

In other words, for each nonterminal A in a parse tree or in a sentential form, μ gives us a probability distribution over the rules that we can apply.

A WCFG is said to be *consistent* if:

$$\sum_{d,w} \mu(S \overset{d}{\Rightarrow} w) = 1. \tag{3}$$

This means that μ is a probability distribution over derivations of terminal strings. An equivalent statement is that μ is a probability distribution over terminal strings, as:

$$\sum_{d,w} \mu(S \overset{d}{\Rightarrow} w) = \sum_{w} \mu(w). \tag{4}$$

Clearly, consistency implies convergence. Properness and consistency are two closely related concepts but, as we will see below, neither implies the other.

An important auxiliary concept for much of the theory that is to follow is the *partition function* Z, which maps each nonterminal A to:

$$Z(A) = \sum_{d,w} \mu(A \overset{d}{\Rightarrow} w). \tag{5}$$

Note that a WCFG is consistent if and only if $Z(S) = 1$.

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

$$Z(A) = \sum_{\pi=(A\rightarrow\alpha)} \mu(\pi) \cdot Z(\alpha), \tag{6}$$

where we define:

$$Z(\epsilon) = 1, \tag{7}$$

$$Z(a\beta) = Z(\beta), \tag{8}$$

$$Z(B\beta) = Z(B) \cdot Z(\beta), \text{ for } \beta \neq \epsilon. \tag{9}$$

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

$$Z_k(A) = \sum_{d,w:\, depth(d) \leq k} p(A \stackrel{d}{\Rightarrow} w), \tag{10}$$

where the depth of a left-most derivation is the largest 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 $\pi = (A \rightarrow X_1 \cdots X_m)$ and $X_i \stackrel{d_i}{\Rightarrow} w_i$ $(1 \leq i \leq m)$, then $depth(\pi d_1 \cdots d_m) = 1 + \max_i depth(d_i)$.

By again decomposing derivations, we obtain a recursive characterisation:

$$Z_{k+1}(A) = \sum_{\pi=(A\rightarrow\alpha)} p(\pi) \cdot Z_k(\alpha), \tag{11}$$

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

$$Z_k(\epsilon) = 1, \tag{12}$$

$$Z_k(a\beta) = Z_k(\beta), \tag{13}$$

$$Z_k(B\beta) = Z_k(B) \cdot Z_k(\beta), \text{ for } \beta \neq \epsilon. \tag{14}$$

Naturally, for all A:

$$\lim_{k\rightarrow\infty} Z_k(A) = Z(A). \tag{15}$$

If we interpret (6) together with (7) through (9) as a system of polynomial equations over variables $Z(A)$, for the set of nonterminals $A \in N$, then there may be several solutions. The intended solution, as given by (5), is the smallest non-negative solution. This follows from the fact that the operation implied by (11) that computes values $Z_{k+1}(A)$ from values $Z_k(B)$ is monotone, and the least fixed-point of this operation corresponds to (5), following (15).

The values $Z(A)$ may be approximated by computing $Z_k(A)$ for $k = 1, \ldots$ until the values stabilise. Another option is to use Newton's method [11]. In special cases, the solution can be found analytically.

Example 2. Consider the following proper WCFG:

$$\begin{array}{ll} S \rightarrow S\ S & (q) \\ S \rightarrow a & (1-q) \end{array}$$

for a certain choice of q between 0 and 1. Using (6) through (9), we obtain:

$$Z(S) = q \cdot Z(S)^2 + (1-q). \tag{16}$$

We can solve this equation, distinguishing between two cases. If $q \leq \frac{1}{2}$, then $Z(S) = 1$ and if $q > \frac{1}{2}$, then $Z(S) = \frac{1-q}{q}$. We make use of the fact that we need the smallest non-negative solution. It follows that the WCFG is consistent

only if $q \leq \frac{1}{2}$. The intuition for the case $q > \frac{1}{2}$ is that probability mass is lost in 'infinite derivations'.

A WCFG can also be consistent without being proper. An example is:

$$\begin{array}{rl} S^{\dagger} \rightarrow S & (\frac{q}{1-q}) \\ S \rightarrow S\ S & (q) \\ S \rightarrow a & (1-q) \end{array}$$

for $\frac{1}{2} < q < 1$.

3 Weighted intersection

It was shown by [12] that context-free languages are closed under intersection with regular languages. The proof relies 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 weighted CFG. For an even more general construction, where also the finite automaton is weighted, we refer to [13].

To avoid a number of technical complications, we assume here that the finite automaton has no epsilon transitions, and has only one final state. Thus, a finite automaton (FA) $\mathcal{M}$ is a 5-tuple $(\Sigma, Q, q_0, q_f, \Delta)$, where Σ and Q are two finite sets of *input symbols* and *states*, respectively, q_0 is the *initial* state, q_f is the *final* state, and Δ is a finite set of *transitions*, each of the form $s \overset{a}{\mapsto} t$, where $s, t \in Q$ and $a \in \Sigma$.

For a FA $\mathcal{M}$ as above and a PCFG $\mathcal{G} = (\Sigma, N, S, R, \mu)$ with the same set Σ, we construct a new PCFG $\mathcal{G}_\cap = (\Sigma, N_\cap, S_\cap, R_\cap, \mu_\cap)$, where $N_\cap = Q \times (\Sigma \cup N) \times Q$, $S_\cap = (q_0, S, q_f)$, and $R_\cap$ is the set of rules that is obtained as follows.

- For each $A \rightarrow X_1 \cdots X_m$ in R and each sequence $s_0, \ldots, s_m \in Q$, with $m \geq 0$, let $(s_0, A, s_m) \rightarrow (s_0, X_1, s_1) \cdots (s_{m-1}, X_m, s_m)$ be in $R_\cap$; if $m = 0$, the new rule is of the form $(s_0, A, s_0) \rightarrow \epsilon$. Function $\mu_\cap$ assigns the same weight to the new rule as μ assigned to the original rule.
- For each $s \overset{a}{\mapsto} t$ in Δ, let $(s, a, t) \rightarrow a$ be in $R_\cap$. Function $\mu_\cap$ assigns weight 1 to this rule.

Observe that a rule of $\mathcal{G}_\cap$ is constructed either 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 $\mathcal{G}_\cap$, $\mathcal{G}$ and $\mathcal{M}$, it can be stated that each derivation $d_\cap$ in $\mathcal{G}_\cap$ deriving a string w corresponds to a unique derivation d in $\mathcal{G}$ deriving the same string and a unique computation c in $\mathcal{M}$ recognising the same string. Conversely, if there is a derivation d in $\mathcal{G}$ deriving string w, and some computation c in $\mathcal{M}$ recognising the same string, then the pair of d and c corresponds to a unique derivation $d_\cap$ in $\mathcal{G}_\cap$ deriving the same string w. Furthermore, the weights of d and $d_\cap$ are equal, by the definition of $\mu_\cap$.

Parsing of a string $w = a_1 \cdots a_n$ can be seen as a special case of the construction, where there is a linear FA, with states $q_0 = s_0, s_1, \ldots, s_n = q_f$,

and transitions of the form $s_{i-1} \overset{a_i}{\mapsto} s_i$ $(1 \leq i \leq n)$. The intersection grammar constructed as explained above can be seen as a succinct representation of all parses of w. As weights are copied unchanged from $\mathcal{G}$ to $\mathcal{G}_\cap$, we can find the parse of w with the highest weight on the basis of $\mathcal{G}_\cap$. We will return to this issue in Section 5.

We say a nonterminal in a CFG is *generating* if at least one terminal string can be derived from that nonterminal. We say a nonterminal is *reachable* if a string containing that nonterminal can be derived from the start symbol. A nonterminal is called *useless* if it is non-generating or non-reachable or both. A grammar $\mathcal{G}_\cap$ as obtained above generally contains a large number of useless nonterminals, to the extent that the construction as given may not be practical.

Introduction of non-generating nonterminals can be avoided by constructing rules in a bottom-up phase. That is, a rule is introduced only if all the members in the right-hand side have been found to be generating. This ensures that the left-hand side nonterminal is also generating. In a following top-down phase, the non-reachable nonterminals can be eliminated, by a standard technique that is linear in the size of the grammar [14].

Below, we will assume one more improvement. The motivation is that the number of rules of the form $(s_0, A, s_m) \rightarrow (s_0, X_1, s_1) \cdots (s_{m-1}, X_m, s_m)$ is exponential in m. Our improvement effectively postpones enumeration of all relevant combinations of $s_1, \ldots, s_{m-1}$ until (s_0, A, s_m) is found to be reachable in the top-down phase. During the bottom-up phase, given in Figure 1, such rules are constructed incrementally by items of the form $(s_0, A \rightarrow \alpha \bullet \beta, s_i)$, where $A \rightarrow \alpha\beta$ is a rule and $i = |\alpha|$. Existence of such an item in table $\mathcal{I}$ means that there are $s_1, \ldots, s_{i-1}$ such that $(s_0, X_1, s_1), \ldots, (s_{i-1}, X_i, s_i)$ are all generating nonterminals, with $\alpha = X_1 \cdots X_i$. We also have a separate table $\mathcal{N}$ to store such generating nonterminals.

The bottom-up phase is similar to a bottom-up variant of the parsing algorithm by [15], and the complexity is very similar. The time complexity in our case is cubic in the number of states of $\mathcal{M}$ and linear in the size of $\mathcal{G}$. The space complexity is quadratic in the number of states of $\mathcal{M}$.

Let us now turn to the construction of $\mathcal{G}_\cap$ out of $\mathcal{N}$ and $\mathcal{I}$ in the top-down phase, given in Figure 2. From the start symbol (q_0, S, q_f), we descend and construct rules for reachable nonterminals that were also found to be generating in the bottom-up phase. Nonterminals are individually added to $N_\cap$ in such a way that rules cannot be constructed more than once.

Some remarks about the implementation are in order. First, the agenda $\mathcal{A}$ is here represented as a set to avoid the presence of duplicate elements. The maximum number of elements the agenda may contain at any given time is thereby quadratic in the number of states of $\mathcal{M}$. If we alternatively represent the agenda as a queue or stack, allowing elements to be present more than once, the space complexity may become cubic.

Second, one may use $\mathcal{I}$ in the top-down phase to guide the search for relevant rules from $\mathcal{G}$ and states from $\mathcal{M}$. This process is further simplified by having the bottom-up phase record a list of the reasons why a certain element is in $\mathcal{N}$ or

```
first_phase:
  𝒩 = ∅ {table of generating nonterminals for N∩}
  ℐ = ∅ {table of items, partially representing rules for R∩}
  𝒜 = ∅ {agenda, items yet to be processed}
  for all (s ↦^a t) ∈ Δ do
    add_symbol(s, a, t)
  for all s ∈ Q do
    for all (A → ϵ) ∈ R do
      𝒜 = 𝒜 ∪ {(s, A → •, s)}
  while 𝒜 ≠ ∅ do
    choose (s, A → α • β, t) ∈ 𝒜
    𝒜 = 𝒜 − {(s, A → α • β, t)}
    add_item(s, A → α • β, t)

add_symbol(s, X, t):
  if (s, X, t) ∉ 𝒩
    𝒩 = 𝒩 ∪ {(s, X, t)}
    for all (r, A → α • Xβ, s) ∈ ℐ do
      𝒜 = 𝒜 ∪ {(r, A → αX • β, t)}
    for all (A → Xβ) ∈ R do
      𝒜 = 𝒜 ∪ {(s, A → X • β, t)}

add_item(r, A → α • β, s):
  if (r, A → α • β, s) ∉ ℐ
    ℐ = ℐ ∪ {(r, A → α • β, s)}
    if β = ϵ
      add_symbol(r, A, s)
    else
      let Xγ = β
      for all (s, X, t) ∈ 𝒩 do
        𝒜 = 𝒜 ∪ {(r, A → αX • γ, t)}
```

Fig. 1. The bottom-up phase of the intersection algorithm. Input are PCFG $\mathcal{G}$ and FA $\mathcal{M}$. The tables $\mathcal{N}$ and $\mathcal{I}$ will be used in the subsequent top-down phase.

$\mathcal{I}$. For example, if $(r, A \rightarrow \alpha X \bullet \beta, t)$ was obtained from $(r, A \rightarrow \alpha \bullet X\beta, s)$ and (s, X, t), then the mentioned list for $(r, A \rightarrow \alpha X \bullet \beta, t)$ contains amongst others the pair consisting of $(r, A \rightarrow \alpha \bullet X\beta, s)$ and (s, X, t). Such a pair is recorded in the list by *add_symbol* if (s, X, t) is added to $\mathcal{N}$ after $(r, A \rightarrow \alpha \bullet X\beta, s)$ is added to $\mathcal{I}$, and it is recorded by *add_item* otherwise.

The additional bookkeeping however is at the cost of having larger tables at the end of the bottom-up phase. This increase is from square to cubic in the number of states of $\mathcal{M}$, as a pair consisting of $(r, A \rightarrow \alpha \bullet X\beta, s)$ and (s, X, t) contains three states. See also [16, Exercise 4.2.21].

With or without the above optimisations, the space complexity is $\mathcal{O}(|Q|^{r+1})$, where r is the length of the longest right-hand side. This can be reduced to $\mathcal{O}(|Q|^3)$, either by transforming the original grammar to binary form (that is,

second_phase:
  $make_rules(q_0, S, q_f)$

$make_rules(r, A, s)$: {if second argument is nonterminal}
  if $(r, A, s) \notin N_\cap$
    $N_\cap = N_\cap \cup \{(r, A, s)\}$
    for all $\pi = (A \to X_1 \cdots X_m) \in R$ **do**
      $s_0 = r$
      $s_m = s$
      for all $s_1, \ldots, s_{m-1} \in Q$ such that $(s_0, X_1, s_1), \ldots, (s_{m-1}, X_m, s_m) \in \mathcal{N}$ **do**
        $\rho = (r, A, s) \to (s_0, X_1, s_1) \cdots (s_{m-1}, X_m, s_m)$
        $R_\cap = R_\cap \cup \{\rho\}$
        $\mu_\cap(\rho) = \mu(\pi)$
        for all i such that $1 \leq i \leq m$ **do**
          $make_rules(s_{i-1}, X_i, s_i)$

$make_rules(r, a, s)$: {if second argument is terminal}
  if $(r, a, s) \notin N_\cap$
    $N_\cap = N_\cap \cup \{(r, a, s)\}$
    $\rho = (r, a, s) \to a$
    $R_\cap = R_\cap \cup \{\rho\}$
    $\mu_\cap(\rho) = 1$

Fig. 2. The top-down phase of the intersection algorithm. On the basis of table $\mathcal{N}$, the nonterminals and rules of $\mathcal{G}_\cap$ are constructed, together with the weight function $\mu_\cap$ on rules.

with $r = 2$) before the intersection, or by refining the intersection algorithm to return a grammar in binary form.

Example 3. Figure 3 shows the end result $\mathcal{G}_\cap$ of applying the algorithm in Figures 1 and 2 on an example PCFG $\mathcal{G}$ and FA $\mathcal{M}$.

4 Normalisation

An obvious question is whether general convergent WCFGs have any advantages over proper and consistent PCFGs. In this section we will show that if we are only interested in the ratios between the weights of derivations, rather than in absolute values, the answer is negative. This allows us to restrict our attention to proper and consistent PCFGs for the purpose of disambiguation.

The argument hinges on *normalisation* of WCFGs [17, 18, 13, 19], which can be defined as the construction of a proper and consistent PCFG (Σ, N, S, R, p) out of a convergent WCFG (Σ, N, S, R, μ). The function p is given by:

$$p(\pi) = \frac{\mu(\pi) \cdot Z(\alpha)}{Z(A)}, \tag{17}$$

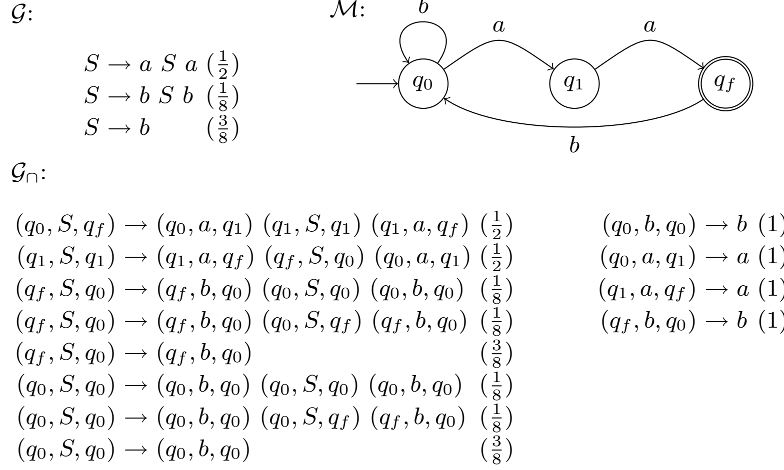

Fig. 3. Example of intersection of PCFG $\mathcal{G}$ and FA $\mathcal{M}$, resulting in $\mathcal{G}_\cap$, which is presented here without useless nonterminals.

for each rule $\pi = (A \to \alpha)$. In words, the probability of a rule is normalised to the portion it represents of the total weight mass of derivations from the left-hand side nonterminal A.

That the ratios between weights of derivations are not affected by normalisation follows from a result in [13]:

$$p(S \stackrel{d}{\Rightarrow} w) = \frac{\mu(S \stackrel{d}{\Rightarrow} w)}{Z(S)}, \tag{18}$$

for each derivation d and string w. In other words, the weights of all derivations change by the same factor. Note that this factor is 1 if the original grammar is already consistent. This implies that consistent WCFGs and proper and consistent PCFGs describe the same class of probability distributions over derivations.

Example 4. Let us return to the WCFG from Example 2, with the values of μ between brackets:

$$\begin{array}{ll} S \to S\ S & (q) \\ S \to a & (1-q) \end{array}$$

The result of normalisation is the proper and consistent PCFG below, with the values of p between brackets:

$$\begin{array}{ll} S \to S\ S & (q') \\ S \to a & (1-q') \end{array}$$

For $q \leq \frac{1}{2}$, we have $q' = q$. For $q > \frac{1}{2}$ however, we have:

$$q' = \tfrac{q \cdot Z(SS)}{Z(S)} = \tfrac{q \cdot Z(S)^2}{Z(S)} = q \cdot Z(S) = q \cdot \tfrac{1-q}{q} = 1 - q. \tag{19}$$

The values of $Z(S)$ and q' as functions of q are represented in Figure 4.

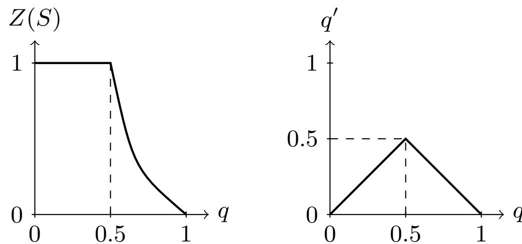

Fig. 4. The values of $Z(S)$ and q' as functions of q, for Example 4.

5 Parsing

As explained in Section 3, context-free parsing is strongly related to computing the intersection of a context-free grammar and a finite automaton. If the input grammar is probabilistic, the probabilities of the rules are simply copied to the intersection grammar. The remaining task is to find the most probable derivation in the intersection grammar.

Note that the problem of finding the most probable derivation in a PCFG does not rely on that PCFG being the intersection of another PCFG and a FA. Let us therefore consider an arbitrary PCFG $\mathcal{G} = (\Sigma, N, S, R, p)$, and our task is to find d and w such that $p(S \stackrel{d}{\Rightarrow} w)$ is maximal. Let p_{max} denote this maximal value. We further define $p_{max}(X)$ to be the maximal value of $p(X \stackrel{d}{\Rightarrow} w)$, for any d and w, where X can be a terminal or nonterminal. Naturally, $p_{max} = p_{max}(S)$ and $p_{max}(a) = 1$ for each terminal a.

Much of the following discussion will focus on computing p_{max} rather than on computing a choice of d and w such that $p_{max} = p(S \stackrel{d}{\Rightarrow} w)$. The justification is that most algorithms to compute p_{max} can be easily extended to a computation of relevant d and w using additional data structures that record how intermediate results were obtained. These data structures however make the discussion less transparent, and are therefore largely ignored.

Consider the graph that consists of the nonterminals as vertices, with an edge from A to B iff there is a rule of the form $A \rightarrow \alpha B \beta$. If $\mathcal{G}$ is non-recursive, then this graph is acyclic. Consequently, the nonterminals can be arranged in a topological sort $A_1, \ldots, A_{|N|}$. This allows us to compute for $j = |N|, \ldots, 1$ in this order:

$$p_{max}(A_j) = \max_{\pi=(A_j \rightarrow X_1 \cdots X_m)} p(\pi) \cdot p_{max}(X_1) \cdot \ldots \cdot p_{max}(X_m). \tag{20}$$

The topological sort ensures that any value for a nonterminal in the right-hand side has been computed at an earlier step.

A topological sort can be found in linear time in the size of the graph [20]. See [21] for an application strongly related to ours. In many cases however, there is a topological sort that follows naturally from the way that $\mathcal{G}$ is constructed. For

example, assume that $\mathcal{G}$ is the intersection of a PCFG $\mathcal{G}'$ in Chomsky normal form and a linear FA with states $s_0, \ldots, s_n$ as in Section 3. We impose an arbitrary linear ordering $\prec_N$ on the set of nonterminals from $\mathcal{G}'$. As topological sort we can now take the linear ordering $\prec$ defined by:

$$(s_i, A, s_j) \prec (s_{i'}, A', s_{j'}) \text{ iff } \begin{array}{l} j > j' \lor \\ (j = j' \land i < i') \lor \\ (j = j' \land i = i' \land A \prec_N A'). \end{array} \tag{21}$$

By this ordering, the computation of the values in (20) can be seen as a probabilistic extension of CYK parsing [22]. This amounts to a generalised form of Viterbi's algorithm [23], which was designed for probabilistic models with a finite-state structure.

If $\mathcal{G}$ is recursive, then a different algorithm is needed. We may use the fact that the probability of a derivation is always smaller than (or equal to) that of any of its subderivations. The reason is that the probability of a derivation is the product of the probabilities of a list of rules, and these are positive numbers not exceeding 1. We also rely on monotonicity of multiplication, i.e. for any positive numbers c_1, c_2, c_3, if $c_1 < c_2$ then $c_1 \cdot c_3 < c_2 \cdot c_3$.

The algorithm in Figure 5 is a special case of an algorithm by Knuth [24], which generalises Dijkstra's algorithm to compute the shortest path in a weighted graph [20]. In each iteration, the value of $p_{max}(A)$ is established for a nonterminal A. The set $\mathcal{E}$ contains all grammar symbols X for which $p_{max}(X)$ has already been established; this is initially Σ, as we set $p_{max}(a) = 1$ for each $a \in \Sigma$. The set $\mathcal{F}$ contains the nonterminals not yet in $\mathcal{E}$ that are candidates to be added next. Each nonterminal A in $\mathcal{F}$ is such that a derivation from A exists consisting of a rule $A \rightarrow X_1 \cdots X_m$, and derivations from $X_1, \ldots, X_m$ matching the values of $p_{max}(X_1), \ldots, p_{max}(X_m)$ found earlier. The nonterminal A for which such a derivation has the highest probability is then added to $\mathcal{E}$.

Knuth's algorithm can be combined with construction of the intersection grammar, along the lines of [25], which also allows for variants expressing particular parsing strategies. See also [26].

A problem related to finding the most probable parse is to find the k most probable parses. This was investigated by [27–29].

Much of the discussed theory of probabilistic parsing carries over to more powerful formalisms, such as probabilistic tree adjoining grammars [30, 31].

We want to emphasise that finding the most probable string is much harder than finding the most probable derivation. In fact, the decision version of the former problem is NP-complete if there is a specified bound on the string length [32], and it remains so even if the PCFG is replaced by a probabilistic finite automaton [33]; see also [34]. If the bound on the string length is dropped, then this problem becomes undecidable, as shown in [35, 36].

6 Parameter estimation

Whereas rules of grammars are often written by linguists, or derived from structures defined by linguists, it is very difficult to correctly estimate the probabil-

$\mathcal{E} = \Sigma$
repeat
  $\mathcal{F} = \{A \mid A \notin \mathcal{E} \wedge \exists A \rightarrow X_1 \cdots X_m [X_1, \ldots, X_m \in \mathcal{E}]\}$
  if $\mathcal{F} = \emptyset$
    report failure and halt
  for all $A \in \mathcal{F}$ **do**
    $q(A) = \max_{\substack{\pi=(A\rightarrow X_1\cdots X_m): \\ X_1,\ldots,X_m \in \mathcal{E}}} p(\pi) \cdot p_{max}(X_1) \cdot \ldots \cdot p_{max}(X_m)$
  choose $A \in \mathcal{F}$ such that $q(A)$ is maximal
  $p_{max}(A) = q(A)$
  $\mathcal{E} = \mathcal{E} \cup \{A\}$
until $S \in \mathcal{E}$
output $p_{max}(S)$

Fig. 5. Knuth's generalisation of Dijkstra's algorithm, applied to finding the most probable derivation in a PCFG.

ities that should be attached to these rules on the basis of linguistic intuitions. Instead, one often relies on two techniques called *supervised* and *unsupervised* estimation.

6.1 Supervised estimation

Supervised estimation relies on explicit access to a sample of data in which one can observe the events whose probabilities are to be estimated. In the case of PCFGs, this sample is a bag $\mathcal{D}$ of derivations of terminal strings, often called a *tree bank*. We assume a fixed order $d_1, \ldots, d_m$ of the derivations in tree bank $\mathcal{D}$. The bag is assumed to be representative for the language at hand, and the probability of a rule is estimated by the ratio of its frequency in the tree bank and the total frequency of rules with the same left-hand side. This is a form of *relative frequency estimation*.

Formally, define $C(\pi, d)$ to be the number of occurrences of rule π in derivation d. Similarly, $C(A, d)$ is the number of times nonterminal A is expanded in derivation d, or equivalently, the sum of all $C(\pi, d)$ such that π has left-hand side A. Summing these numbers for all derivations in the tree bank, we obtain:

$$C(\pi, \mathcal{D}) = \sum_{1 \leq h \leq m} C(\pi, d_h), \tag{22}$$

$$C(A, \mathcal{D}) = \sum_{1 \leq h \leq m} C(A, d_h). \tag{23}$$

Our estimation for the probability of a rule $\pi = (A \rightarrow \alpha)$ now is:

$$p_{\mathcal{D}}(\pi) = \frac{C(\pi, \mathcal{D})}{C(A, \mathcal{D})}. \tag{24}$$

One justification for this estimation is that it maximises the likelihood of the tree bank [37]. This likelihood for given p is defined by:

$$p(\mathcal{D}) = \prod_{1 \leq h \leq m} p(d_h). \tag{25}$$

The PCFG that results by taking estimation $p_{\mathcal{D}}$ as above is guaranteed to be consistent [38, 39, 37].

Note that supervised estimation assigns probability 0 to rules that do not occur in the tree bank, which means that probabilistic parsing algorithms ignore such rules. A tree bank may contain zero occurrences of rules because it is too small to contain all phenomena in a language, and some rules that do not occur in one tree bank may in fact be valid and would occur if the tree bank were larger. To circumvent this problem one may apply a form of *smoothing*, which means shifting some probability mass from observed events to those that did not occur. Rules that do not occur in the tree bank thereby obtain a small but non-zero probability. For a study of smoothing techniques used for natural language processing, see [40].

Example 5. Consider the following CFG:

$$\begin{aligned}
\pi_1 &: S \rightarrow S\ S \\
\pi_2 &: S \rightarrow a\ S\ b \\
\pi_3 &: S \rightarrow a\ b \\
\pi_4 &: S \rightarrow b\ a \\
\pi_5 &: S \rightarrow c
\end{aligned}$$

Assume a tree bank consisting of only two derivations, $\pi_1\pi_3\pi_5$ and $\pi_2\pi_3$, with yields *abc* and *aabb*, respectively. Without smoothing, the estimation is $p(\pi_1) = \frac{1}{5}$, $p(\pi_2) = \frac{1}{5}$, $p(\pi_3) = \frac{2}{5}$, $p(\pi_4) = \frac{0}{5}$, $p(\pi_5) = \frac{1}{5}$.

6.2 Unsupervised estimation

We define an (unannotated) *corpus* as a bag $\mathcal{W}$ of strings in a language. As in the case of tree banks, the bag is assumed to be representative for the language at hand. We assume a fixed order $w_1, \ldots, w_m$ of the strings in corpus $\mathcal{W}$. Estimation of a probability assignment p to rules of a CFG on the basis of a corpus is called unsupervised as there is no direct access to frequencies of rules. A string from the corpus may possess several derivations, each representing different bags of rule occurrences.

A common unsupervised estimation for PCFGs is a form of Expectation-Maximisation (EM) algorithm [41]. It computes a probability assignment p by successive refinements $p_0, p_1, \ldots$, until the values stabilise. The initial assignment p_0 may be arbitrarily chosen, and subsequent estimates p_{t+1} are computed on the basis of p_t, in a way to be explained below. In each step, the likelihood $p_t(\mathcal{W})$ of the corpus increases. This likelihood for given p is defined by:

$$p(\mathcal{W}) = \prod_{1 \leq h \leq m} p(w_h). \tag{26}$$

The algorithm converges to a local optimum (or a saddlepoint) with respect to the likelihood of the corpus, but no algorithm is known to compute the global optimum, that is, the assignment p such that $p(\mathcal{W})$ is maximal.

Computation of p_{t+1} on the basis of p_t corresponds to a simple idea. With unsupervised estimation, we do not have access to a single derivation for each string in the corpus, and therefore cannot determine frequencies of rules by simple counts. Instead, we consider all derivations for each string, and the counts we would obtain for individual derivations are combined by taking a weighted average. The weighting of this average is determined by the current assignment p_t, which offers us probabilities $\frac{p_t(d)}{p_t(w)}$, where $y(d) = w$, which is the conditional probability of derivation d given string w.

More precisely, an estimated count $C_t(\pi)$ of a rule π in a corpus, given assignment p_t, can be defined by:

$$C_t(\pi) = \sum_{1 \leq h \leq m} \sum_{d: y(d) = w_h} \frac{p_t(d)}{p_t(w_h)} \cdot C(\pi, d). \tag{27}$$

Similarly:

$$C_t(A) = \sum_{1 \leq h \leq m} \sum_{d: y(d) = w_h} \frac{p_t(d)}{p_t(w_h)} \cdot C(A, d). \tag{28}$$

Using these values we compute the next estimation $p_{t+1}(\pi)$ for each rule $\pi = (A \to \alpha)$ as:

$$p_{t+1}(\pi) = \frac{C_t(\pi)}{C_t(A)}. \tag{29}$$

Note the similarity of this to (24).

Example 6. Consider the CFG from Example 5, with a corpus consisting of strings $w_1 = abc$, $w_2 = acb$ and $w_3 = abab$. The first two strings can only be derived by $d_1 = \pi_1\pi_3\pi_5$ and $d_2 = \pi_2\pi_5$, respectively. However, w_3 is ambiguous as it can be derived by $d_3 = \pi_1\pi_3\pi_3$ and $d_4 = \pi_2\pi_4$.

For a given p_t, we have:

$$\begin{aligned}
C_t(\pi_1) &= 1 + \frac{p_t(d_3)}{p_t(w_3)} \\
C_t(\pi_2) &= 1 + \frac{p_t(d_4)}{p_t(w_3)} \\
C_t(\pi_3) &= 1 + 2 \cdot \frac{p_t(d_3)}{p_t(w_3)} \\
C_t(\pi_4) &= \frac{p_t(d_4)}{p_t(w_3)} \\
C_t(\pi_5) &= 2 \\
C_t(S) &= 5 + 3 \cdot \frac{p_t(d_3)}{p_t(w_3)} + 2 \cdot \frac{p_t(d_4)}{p_t(w_3)}
\end{aligned}$$

The assignment that p_t converges to depends on the initial choice of p_0. We investigate two such initial choices:

	$p_t(\pi_1)$	$p_t(\pi_2)$	$p_t(\pi_3)$	$p_t(\pi_4)$	$p_t(\pi_5)$
$t = 0$	0.200	0.200	0.200	0.200	0.200
$t = 1$	0.163	0.256	0.186	0.116	0.279
$t = 2$	0.162	0.257	0.184	0.117	0.279
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t = \infty$	0.160	0.260	0.180	0.120	0.280

and:

	$p_t(\pi_1)$	$p_t(\pi_2)$	$p_t(\pi_3)$	$p_t(\pi_4)$	$p_t(\pi_5)$
$t = 0$	0.100	0.100	0.600	0.100	0.100
$t = 1$	0.229	0.156	0.330	0.028	0.257
$t = 2$	0.236	0.146	0.344	0.019	0.255
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t = \infty$	0.250	0.125	0.375	0.000	0.250

In the first case, the likelihood of the corpus is $2.14 \cdot 10^{-5}$ and in the second case $2.57 \cdot 10^{-5}$.

As strings may allow a large number of derivations, a direct implementation of (27) and (28) is often not feasible. To obtain a more practical algorithm, we first rewrite $C_t(\pi)$ as below. Treatment of $C_t(A)$ is similar.

$$C_t(\pi) = \sum_{1 \leq h \leq m} \frac{1}{p_t(w_h)} \sum_{d: y(d) = w_h} p_t(d) \cdot C(\pi, d). \tag{30}$$

The value $p_t(w_h)$ is just $Z(S_{t,h})$, where $S_{t,h}$ is the start symbol of the intersection of the PCFG with probability assignment p_t and the linear FA accepting the singleton language $\{w_h\}$. How this value can be computed has already been explained in Section 2. Let us therefore concentrate on the second part of the above expression, fixing an assignment p, rule $\pi = (A \rightarrow \alpha)$ and string $w = a_1 \cdots a_n$. We rewrite:

$$\begin{aligned} \sum_{d: y(d) = w} p(d) \cdot C(\pi, d) &= \sum_{i,j} \sum_{d_1, d_2, d_3, \beta} p(S \stackrel{d_1}{\Rightarrow} a_1 \cdots a_i A \beta) \cdot p(\pi) \cdot \\ &\qquad\qquad p(\alpha \stackrel{d_2}{\Rightarrow} a_{i+1} \cdots a_j) \cdot \\ &\qquad\qquad p(\beta \stackrel{d_3}{\Rightarrow} a_{j+1} \cdots a_n) \qquad (31) \\ &= \sum_{i,j} \mathit{outer}(A, i, j) \cdot p(\pi) \cdot \mathit{inner}(\alpha, i, j), \qquad (32) \end{aligned}$$

where we define:

$$\mathit{outer}(A, i, j) = \sum_{d_1, d_3, \beta} p(S \stackrel{d_1}{\Rightarrow} a_1 \cdots a_i A \beta) \cdot p(\beta \stackrel{d_3}{\Rightarrow} a_{j+1} \cdots a_n), \tag{33}$$

$$\mathit{inner}(\alpha, i, j) = \sum_{d_2} p(\alpha \stackrel{d_2}{\Rightarrow} a_{i+1} \cdots a_j). \tag{34}$$

The intuition is that the occurrences of π in the different d such that $y(d) = w$ are grouped according to the substring $a_{i+1} \cdots a_j$ that they cover. For each choice of i and j we look at the sum of probabilities of matching derivations, dividing them into the subderivations that are 'inside' and 'outside' the relevant occurrence of π.

The values of $\mathit{inner}(\alpha, i, j)$ can be computed similarly to the computation of the partition function Z, which was explained in Section 2. For the remaining values, we have:

$$\begin{aligned} &\mathit{outer}(A, i, j) = \\ &\quad \delta(A = S \wedge i = 0 \wedge j = n) + \\ &\quad \sum_{\pi=(B\to\gamma A\eta),i',j'} \mathit{outer}(B, i', j') \cdot p(\pi) \cdot \mathit{inner}(\gamma, i', i) \cdot \mathit{inner}(\eta, j, j'), \end{aligned} \tag{35}$$

with δ defined to return 1 if its argument is true and 0 otherwise. Here we divide the derivations 'outside' a nonterminal occurrence into parts outside parent nonterminal occurrences, and the parts below siblings on the left and on the right. A special case is if the nonterminal occurrence can be the root of the parse tree, which corresponds to a value of 1, which is the product of zero rule probabilities.

If we fill in the values for *inner*, we obtain a system of linear equations with $\mathit{outer}(A, i, j)$ as variables, which can be solved in polynomial time. The system is of course without cyclic dependencies if the grammar is without cycles.

The algorithm we have described is called the *inside-outside algorithm* [42, 43, 22, 44]. It generalises the *forward-backward algorithm* for probabilistic models with a finite-state structure [45]. Generalised PCFGs, with right-hand sides representing regular languages, were considered in [46]. The inside-outside algorithm is guaranteed to result in consistent PCFGs [39, 37, 19].

7 Prefix probabilities

Let p be the probability assignment of a PCFG. The *prefix probability* of a string w is defined to be $\mathit{Pref}(w) = \sum_v p(wv)$. Prefix probabilities have important applications in speech recognition. For example, assume a prefix of the input is w, and the next symbol suggested by the speech recogniser is a. The probability that a is the next symbol according to the PCFG is given by:

$$\frac{\mathit{Pref}(wa)}{\mathit{Pref}(w)}. \tag{36}$$

For given w, there may be infinitely many v such that $p(wv) > 0$. As we will show, the difficulty of summing infinitely many values can be overcome by isolating a finite number of auxiliary values whose computation can be carried out 'off-line', that is, independent of any particular w. On the basis of these values, $\mathit{Pref}(w)$ can be computed in cubic time for any given w.

We first extend left-most derivations to 'dotted' derivations written as $S \overset{d}{\Rightarrow} w \bullet \alpha$. The dot indicates a position in the sentential form separating the known

prefix w and a string α of grammar symbols together generating an unknown suffix v. No symbol to the right of the dot may be rewritten. The rationale is that this would lead to probability mass being included more than once in the theory that is to follow.

Formally, a dotted derivation can be either $A \overset{\epsilon}{\Rightarrow} \bullet A$, which represents the empty derivation, or it can be of the form $A \overset{d\pi}{\Rightarrow} wv \bullet \alpha\beta$, where $\pi = (B \rightarrow v\alpha)$ with $v \neq \epsilon$, to represent the (left-most) derivation $A \overset{d}{\Rightarrow} wB\beta \overset{\pi}{\Rightarrow} wv\alpha\beta$.

In the remainder of this section, we will assume that the PCFG is proper and consistent. This allows us to rewrite:

$$Pref(w) = \sum_{d,\alpha} p(S \overset{d}{\Rightarrow} w \bullet \alpha) \cdot \sum_{d',v} p(\alpha \overset{d'}{\Rightarrow} v) = \sum_{d,\alpha} p(S \overset{d}{\Rightarrow} w \bullet \alpha). \quad (37)$$

Note that derivations leading from any α in the above need not be considered individually, as the sum of their probabilities is 1 for proper and consistent PCFGs.

Example 7. We investigate the prefix probability of bb, for the following PCFG:

$$\begin{array}{ll} \pi_1 : S \rightarrow A\ a & (0.2) \\ \pi_2 : S \rightarrow b & (0.8) \\ \pi_3 : A \rightarrow S\ a & (0.4) \\ \pi_4 : A \rightarrow S\ b & (0.6) \end{array}$$

The set of derivations d such that $S \overset{d}{\Rightarrow} bb \bullet \alpha$, some α, can be described by the regular expression $(\pi_1(\pi_3 \cup \pi_4))^* \pi_1 \pi_4 \pi_2$. By summing the probabilities of these derivations, we get:

$$\begin{aligned} Pref(bb) &= \sum_{m \geq 0} \big(p(\pi_1) \cdot (p(\pi_3) + p(\pi_4))\big)^m \cdot p(\pi_1) \cdot p(\pi_4) \cdot p(\pi_2) \\ &= \sum_{m \geq 0} p(\pi_1)^m \cdot p(\pi_1) \cdot p(\pi_4) \cdot p(\pi_2). \end{aligned}$$

As $\sum_{m \geq 0} p(\pi_1)^m = \frac{1}{1 - p(\pi_1)} = 1.25$, we obtain:

$$Pref(bb) = 1.25 \cdot 0.2 \cdot 0.6 \cdot 0.8 = 0.12.$$

The remainder of this section derives a practical solution for computing the value in (37), due to [47]. This requires that the underlying CFG is in Chomsky normal form, or more precisely that every rule has the form $A \rightarrow BC$ or $A \rightarrow a$. We will ignore rules $S \rightarrow \epsilon$ here.

We first distinguish two kinds of subderivation. For the first kind, the yield falls entirely within the known prefix $w = a_1 \cdots a_n$. For the second, the yield includes the boundary between known prefix w and unknown suffix v. We do not have to investigate the third kind of subderivation, whose yield falls entirely within the unknown suffix, because the factors involved are always 1, as explained before.

For subderivations within the known prefix we have values of the form:

$$\textstyle\sum_d p(A \overset{d}{\Rightarrow} a_{i+1} \cdots a_j), \tag{38}$$

with i and j between 0 and n. These values can be computed using techniques already discussed, in Section 2 for Z, and in Section 6.2 for *inner*. Here, the computation can be done in cubic time in the length of the prefix, since there are no cyclic dependencies.

Let us now look at derivations at the boundary between w and v. If the relevant part of w is empty, we have:

$$\sum_d p(A \overset{d}{\Rightarrow} \bullet\, A) = 1. \tag{39}$$

If the relevant part of w is only one symbol a_n, we have:

$$\sum_{d,\alpha} p(A \overset{d}{\Rightarrow} a_n \bullet \alpha) = \sum_{\pi=(B\to a_n)} \sum_{d,\alpha} p(A \overset{d}{\Rightarrow} B\alpha) \cdot p(\pi). \tag{40}$$

Here B plays the role of the last nonterminal in a path in a parse tree from A down to a_n, taking the left-most child at each step.

It is easy to see that:

$$\sum_{d,\alpha} p(A \overset{d}{\Rightarrow} B\alpha) = \delta(A = B) + \sum_{\pi=(A\to CD)} p(\pi) \cdot \sum_{d,\alpha} p(C \overset{d}{\Rightarrow} B\alpha). \tag{41}$$

If we replace expressions of the form $\sum_{d,\alpha} p(A \overset{d}{\Rightarrow} B\alpha)$ by variables $chain(A,B)$, then (41) represents a system of linear equations, for fixed B and different A. This system can be solved with a time complexity that is cubic in the number of nonterminals. Note that this is independent of the known prefix w, and is therefore an off-line computation.

If the derivation covers a larger portion of the prefix ($i+1 < n$) we have:

$$\begin{aligned}
&\sum_{d,\alpha} p(A \overset{d}{\Rightarrow} a_{i+1} \cdots a_n \bullet \alpha) = \\
&\quad \sum_{\pi=(D\to BC)} \sum_{d,\alpha} p(A \overset{d}{\Rightarrow} D\alpha) \cdot p(\pi) \cdot \\
&\quad \sum_{k:i<k\le n} \sum_{d_1} p(B \overset{d_1}{\Rightarrow} a_{i+1} \cdots a_k) \cdot \sum_{d_2,\beta} p(C \overset{d_2}{\Rightarrow} a_{k+1} \cdots a_n \bullet \beta).
\end{aligned} \tag{42}$$

The intuition is as follows. In a path in the parse tree from the indicated occurrence of A to the occurrence of a_{i+1}, there is a first node, labelled B, whose yield is entirely within the known prefix. The yield of its sibling, labelled C, includes the remainder of the prefix to the right as well as part of the unknown suffix.

We already know how to compute $\sum_{d,\alpha} p(A \overset{d}{\Rightarrow} D\alpha)$ and $\sum_{d_1} p(B \overset{d_1}{\Rightarrow} a_{i+1} \cdots a_k)$. If we now replace expressions of the form $\sum_{d,\alpha} p(A \overset{d}{\Rightarrow} a_{i+1} \cdots a_n \bullet$

α) in (42) by variables $\mathit{prefix_inside}(A, i)$, then we obtain a system of equations, which define values $\mathit{prefix_inside}(A, i)$ in terms of $\mathit{prefix_inside}(B, k)$ with $k > i$. These values can be computed in quadratic time if the other values in (42) have already been obtained. The resulting value of $\mathit{prefix_inside}(S, 0)$ is the required prefix probability of w.

In many applications, the prefix probabilities need to be computed for increasingly long strings. For example in real-time speech recognition, the input grows as the acoustic signal is processed and sequences of sounds are recognised as words. The algorithm above has the disadvantage that the values in (42) are specific to the length n of the prefix w. If w grows on the right by one more symbol, the computation of the values has to be done anew. The algorithm by [48], which is based on Earley's algorithm, suffers less from this problem. Most of the values it computes can be reused as the prefix grows. A very similar algorithm was described by [49]. It differs from [48] in that it does not explicitly isolate any off-line computations.

Prefix probabilities for tree adjoining and linear indexed grammars were investigated by [50, 51].

8 Probabilistic push-down automata

A paper in a previous volume [1] argued that a parsing strategy can be formalised as a mapping from CFGs to push-down automata that preserves the described languages. In this section we investigate the extension of this notion to probabilistic parsing strategies [52], which are to preserve probability distributions over strings.

As in [1], our type of push-down automaton does not possess states. Hence, a *push-down automaton* (PDA) $\mathcal{M}$ is a 5-tuple $(\Sigma, \Gamma, X_{init}, X_{final}, \Delta)$, where Σ is a finite set of *input symbols*, Γ is a finite set of *stack symbols*, $X_{init} \in \Gamma$ is the *initial stack symbol*, $X_{final} \in \Gamma$ is the *final stack symbol*, and Δ is the set of *transitions*. Each transition can have one of the following three forms: $X \overset{\epsilon}{\mapsto} XY$ (a push transition), $YX \overset{\epsilon}{\mapsto} Z$ (a pop transition), or $X \overset{x}{\mapsto} Y$ (a swap transition); here $X, Y, Z \in \Gamma$, $x \in \Sigma \cup \{\epsilon\}$. Note that in our notation, stacks grow from left to right, i.e., the top-most stack symbol will be found at the right end.

Without loss of generality, we assume that any PDA is such that for a given stack symbol $X \neq X_{final}$, there are either one or more push transitions $X \overset{\epsilon}{\mapsto} XY$, or one or more pop transitions $YX \overset{\epsilon}{\mapsto} Z$, or one or more swap transitions $X \overset{x}{\mapsto} Y$, but no combinations of different kinds of transition. If a PDA does not satisfy this normal form, it can easily be brought in this form by introducing for each stack symbol $X \neq X_{final}$ three new stack symbols X_{push}, X_{pop} and X_{swap} and new swap transitions $X \overset{\epsilon}{\mapsto} X_{push}$, $X \overset{\epsilon}{\mapsto} X_{pop}$ and $X \overset{\epsilon}{\mapsto} X_{swap}$. In each existing transition that operates on top-of-stack X, we then replace X by one from X_{push}, X_{pop} or X_{swap}, depending on the type of that transition. We also assume that X_{final} does not occur in the left-hand side of any transition, again without loss of generality.

As usual, the process of recognition of a string w starts with a configuration consisting of the singleton stack X_{init}. If a list of transitions leads to singleton stack X_{final} when the entire input w has been scanned, then we say that w is recognised. Such a list of transitions is called a *computation*. The language accepted by the PDA is the set of all strings that can be recognised. We assume below that a PDA is always *reduced*, which means that each stack symbol can be used in some computation that recognises a string. For more precise definitions, we refer to [1].

A *weighted push-down automaton* (WPDA) $\mathcal{M}$ is a 6-tuple (Σ, Γ, X_{init}, X_{final}, Δ, μ), where (Σ, Γ, X_{init}, X_{final}, Δ) is a PDA, and μ is a mapping from transitions in Δ to positive real numbers. Thereby, a WPDA assigns weights to computations and strings, in the same way as WCFGs assign weights to derivations and strings.

A *probabilistic push-down automaton* (PPDA) is a WPDA with the restriction that the values assigned to transitions are no greater than 1 [53]. Consistency is defined as for WCFGs. We say a WPDA is *proper* if:

- $\Sigma_{\tau=(X \overset{\epsilon}{\mapsto} XY)}\, p(\tau) = 1$ for each $X \in \Gamma$ such that there is at least one transition of the form $X \overset{\epsilon}{\mapsto} XY$;
- $\Sigma_{\tau=(X \overset{x}{\mapsto} Y)}\, p(\tau) = 1$ for each $X \in \Gamma$ such that there is at least one transition of the form $X \overset{x}{\mapsto} Y$; and
- $\Sigma_{\tau=(YX \overset{\epsilon}{\mapsto} Z)}\, p(\tau) = 1$ for each $X, Y \in \Gamma$ such that there is at least one transition of the form $YX \overset{\epsilon}{\mapsto} Z$.

For each stack that may arise in the recognition of a string, exactly one of the above three clauses applies, depending on the symbol on top (provided this is not X_{final}). The conditions ensure that the sum of probabilities of next possible transitions is always 1.

An obvious question is whether parsing strategies, mapping CFGs to PDAs, preserve the capacity to describe probability distributions on strings. In many cases, PDAs are able to describe a wider range of probability distributions than the CFGs they were derived from by a parsing strategy.

Consider for example the parsing strategy of top-down parsing. The stack symbols of the constructed PDA are of the form $[A \rightarrow \alpha \bullet \beta]$, where $A \rightarrow \alpha\beta$ is a rule in the CFG. The transitions are given by:

- $[A \rightarrow \alpha \bullet a\beta] \overset{a}{\mapsto} [A \rightarrow \alpha a \bullet \beta]$ for each rule $A \rightarrow \alpha a \beta$;
- $[A \rightarrow \alpha \bullet B\beta] \overset{\epsilon}{\mapsto} [A \rightarrow \alpha \bullet B\beta]\ [B \rightarrow \bullet\, \gamma]$ for each pair of rules $A \rightarrow \alpha B \beta$ and $\pi = B \rightarrow \gamma$; and
- $[A \rightarrow \alpha \bullet B\beta]\ [B \rightarrow \gamma \bullet] \overset{\epsilon}{\mapsto} [A \rightarrow \alpha B \bullet \beta]$.

We assume without loss of generality that the start symbol has only one defining rule, say $S \rightarrow \sigma$. The initial stack symbol is then $[S \rightarrow \bullet\, \sigma]$ and the final stack symbol is $[S \rightarrow \sigma \bullet]$.

The probability distribution described by a proper and consistent PCFG can be carried over to a PPDA implementing the top-down parsing strategy if we let

transitions of the first and third kind above have probability 1, and let those of the second kind have the same probability as the rule $B \to \gamma$ from the PCFG.

The reverse does not hold in general. In the PPDA we may assign different probabilities to two different transitions of the form:

– $[A \to \alpha \bullet B\beta] \overset{\epsilon}{\mapsto} [A \to \alpha \bullet B\beta]\ [B \to \bullet\ \gamma]$; and
– $[A' \to \alpha' \bullet B\beta'] \overset{\epsilon}{\mapsto} [A' \to \alpha' \bullet B\beta']\ [B \to \bullet\ \gamma]$.

Such a distinction between the different contexts for an occurrence of nonterminal B cannot normally be encoded into the original CFG. This observation is related to a technique from [54] that allows probability distributions more refined than those that can be expressed in terms of a given CFG. See also [55].

Example 8. Consider the CFG:

$$\begin{aligned}
\pi_1 &: S \to A \\
\pi_2 &: A \to a\ B \\
\pi_3 &: A \to b\ B \\
\pi_4 &: B \to c \\
\pi_5 &: B \to d
\end{aligned}$$

If p is the probability distribution over strings induced by a proper PCFG that extends the CFG above, then we must have:

$$\frac{p(ac)}{p(ad)} = \frac{p(bc)}{p(bd)}. \tag{43}$$

Another way of looking at this is that we have a 2-dimensional parameter space, as there are only two free parameters: once we choose $p(\pi_2)$ and $p(\pi_4)$, then $p(\pi_3)$ must be $1 - p(\pi_2)$ and $p(\pi_5)$ must be $1 - p(\pi_4)$. Naturally $p(\pi_1) = 1$.

Consider now the corresponding top-down PDA. If we are to turn this into a proper PPDA, all transitions must have probability 1, except the following six:

– $[S \to \bullet\ A] \overset{\epsilon}{\mapsto} [S \to \bullet\ A]\ [A \to \bullet\ aB]$,
– $[S \to \bullet\ A] \overset{\epsilon}{\mapsto} [S \to \bullet\ A]\ [A \to \bullet\ bB]$,
– $[A \to a \bullet B] \overset{\epsilon}{\mapsto} [A \to a \bullet B]\ [B \to \bullet\ c]$,
– $[A \to a \bullet B] \overset{\epsilon}{\mapsto} [A \to a \bullet B]\ [B \to \bullet\ d]$,
– $[A \to b \bullet B] \overset{\epsilon}{\mapsto} [A \to b \bullet B]\ [B \to \bullet\ c]$, and
– $[A \to b \bullet B] \overset{\epsilon}{\mapsto} [A \to b \bullet B]\ [B \to \bullet\ d]$.

Now (43) no longer restricts the space of available probability distributions. Seen in a different way, we have a 3-dimensional parameter space, as there are three free parameters; the probabilities of the first, third and fifth transitions above determine those of the others.

Another parsing strategy that preserves the allowable probability distributions is left-corner parsing [56]. This preservation does not hold for all parsing strategies however, as pointed out for bottom-up parsing by [18]. Another strategy for which it does not hold is LR parsing. In [52], an example is given of

a PCFG with a probability distribution that cannot be expressed in terms of the corresponding PDA implementing the LR strategy. However, as shown by [57], this problem disappears if we abandon the requirement that the PPDA be proper.

9 Semirings

Let us compare the computation of Z (Section 2) with the computation of p_{max} (Section 5). An important similarity is that values coming from members in the right-hand side of a rule are multiplied, as can be witnessed in both (9) and (20). An important difference is that in (6) we add the values coming from alternative derivations, whereas in (20) these values are combined by maximisation. By allowing other operations in place of those mentioned above, possibly with another domain of weights, we obtain a general class of computations involving context-free grammars. The domain and operations are subject to a number of constraints, which can be expressed as an algebraic structure.

Formally, a *semiring* is a 5-tuple $(\mathbb{D}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, where $\mathbb{D}$ is a set, $\oplus$ and $\otimes$ are binary operations on $\mathbb{D}$, and $\mathbf{0}, \mathbf{1} \in \mathbb{D}$, with the following properties for all $a, b, c \in \mathbb{D}$:

additive identity $a \oplus \mathbf{0} = \mathbf{0} \oplus a = a$,
additive commutativity $a \oplus b = b \oplus a$,
additive associativity $(a \oplus b) \oplus c = a \oplus (b \oplus c)$,
multiplicative identity $a \otimes \mathbf{1} = \mathbf{1} \otimes a = a$,
annihilation $a \otimes \mathbf{0} = \mathbf{0} \otimes a = \mathbf{0}$,
multiplicative associativity $(a \otimes b) \otimes c = a \otimes (b \otimes c)$,
distributivity $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.

The semiring $(\mathbb{R}^+, +, \cdot, 0, 1)$ underlies the computation of Z, where $\mathbb{R}^+$ stands for the non-negative real numbers, and $+$ and $\cdot$ stand for ordinary addition and multiplication. By replacing $+$ by max, we obtain the semiring $(\mathbb{R}^+, \max, \cdot, 0, 1)$, which underlies the computation of p_{max}. These and several other semirings were discussed in relation to context-free grammars by [58]. Semirings are firmly rooted in the theory of context-free grammars and other formalisms, often in connection with formal power series [59]. For applications of semirings with a focus on finite-state transducers, see [60].

In the remainder of this section, we investigate the semiring $(\mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0)$. The domain includes the symbol ∞, which also acts as the 'zero' element. This means that $\min(a, \infty) = \min(\infty, a) = a$ and $a + \infty = \infty + a = \infty$ for all a. We will show that this semiring is useful for error correction of programming languages, which is closely related to the problem of computing the optimal parse on the basis of WCFGs, as is known from [61].

Assume a CFG $\mathcal{G}$, and assume two functions on terminals, called d and i, and a function s on pairs of terminals. These functions define the costs of correcting a string by deleting or inserting a terminal, or by substituting one terminal by another. Costs are non-negative real numbers. We assume that $s(a, b) \leq$

$d(a) + i(b)$, or in words, it is at least as costly to delete a and insert b as to substitute a by b. Naturally, $s(a, a) = 0$ for all a, which means that leaving a terminal unaffected can be treated as substituting it by itself.

The *minimum edit distance* between two strings w and v, denoted by $dist(w, v)$, is defined as the minimum sum of costs of a list of deletions, insertions and substitutions needed to turn w into v [62]. We are now asked to solve the following problem. Given a string w, compute the string v in the language generated by $\mathcal{G}$ that minimises $dist(w, v)$. Similarly to our presentation in Section 5, the algorithm we will show computes this minimal value $dist(w, v)$, but not the relevant v itself nor the used edit operations. These can be computed by a simple extension of the basic mechanism.

Let us assume a PDA $\mathcal{A}$ instead of a CFG as representation of a context-free language, which slightly simplifies the discussion. The PDA may implement any parsing strategy. We now construct a WPDA $\mathcal{A}'$ as follows. For each stack symbol X in $\mathcal{A}$, $\mathcal{A}'$ has two distinct stack symbols X and X_{del}. A symbol of the form X_{del} will be on top of the stack immediately after a substitution, or at the beginning of the input. While it is on top of the stack, we allow an uninterrupted sequence of deletions, but no other actions. We thereby effectively force a canonical ordering on the edit operation and stack manipulations, placing deletions as early as possible. If the initial stack symbol of $\mathcal{A}$ is X, then that of $\mathcal{A}'$ is X_{del}. Further:

- Pop and push transitions are copied unchanged from $\mathcal{A}$ to $\mathcal{A}'$. Also swap transitions of the form $X \stackrel{\epsilon}{\mapsto} Y$ are copied unchanged. The weight of all these transitions is 0.
- For each transition $X \stackrel{a}{\mapsto} Y$ in $\mathcal{A}$, $\mathcal{A}'$ has the following transitions:
  - $X \stackrel{b}{\mapsto} Y_{del}$ with weight $s(b, a)$, for each b, and
  - $X \stackrel{\epsilon}{\mapsto} Y$ with weight $i(a)$.
- For each stack symbol X_{del}, $\mathcal{A}'$ has the following transitions:
  - $X_{del} \stackrel{a}{\mapsto} X_{del}$ with weight $d(a)$, for each a, and
  - $X_{del} \stackrel{\epsilon}{\mapsto} X$ with weight 0.

As dictated by the specified semiring, weights in a computation are added. In the presence of ambiguity, we take the minimum weight of all the computations that recognise a string.

Example 9. Consider the following grammar and the top-down PDA obtained from it:

$$
\begin{aligned}
S &\rightarrow a\ A\ a \\
A &\rightarrow b\ A\ b \\
A &\rightarrow a
\end{aligned}
$$

One of several ways to recognise the string abb, while allowing for error correction, is by application of the list of transitions in Figure 6. The total weight of the computation is $i(a)+s(b, a)+d(b)$. There are other computations recognising the same string, which may have lower weight, depending on the values of i, d and s.

$$
\begin{array}{ll}
[S \rightarrow \bullet\ aAa]_{del} \overset{\epsilon}{\mapsto} [S \rightarrow \bullet\ aAa] & (0) \\
[S \rightarrow \bullet\ aAa] \overset{\epsilon}{\mapsto} [S \rightarrow a \bullet Aa] & (i(a)) \\
[S \rightarrow a \bullet Aa] \overset{\epsilon}{\mapsto} [S \rightarrow a \bullet Aa]\ [A \rightarrow \bullet\ a] & (0) \\
[A \rightarrow \bullet\ a] \overset{a}{\mapsto} [A \rightarrow a \bullet]_{del} & (0) \\
[A \rightarrow a \bullet]_{del} \overset{\epsilon}{\mapsto} [A \rightarrow a \bullet] & (0) \\
[S \rightarrow a \bullet Aa]\ [A \rightarrow\ a \bullet] \overset{\epsilon}{\mapsto} [S \rightarrow aA \bullet a] & (0) \\
[S \rightarrow aA \bullet a] \overset{b}{\mapsto} [S \rightarrow aAa \bullet]_{del} & (s(b,a)) \\
[S \rightarrow aAa \bullet]_{del} \overset{b}{\mapsto} [S \rightarrow aAa \bullet]_{del} & (d(b)) \\
[S \rightarrow aAa \bullet]_{del} \overset{\epsilon}{\mapsto} [S \rightarrow aAa \bullet] & (0)
\end{array}
$$

Fig. 6. One possible list of transitions that can be applied in order to recognise string *abb* with error correction, in Example 9. The weights of these transitions are given between brackets.

In transforming a PDA to become an error-correcting WPDA, new nondeterminism is introduced. By tabulation however, all computations can be simulated in cubic time in the input length, in a way that allows extraction of the computation with the lowest weight [63]. Depending on the chosen parsing strategy, the result may be similar to Earley's algorithm [64, 65], to CYK parsing [61], or to tabular LR parsing [66, 67].

10 Further references

Some interpretations of probabilistic formalisms differ from what we have described above in that they define acceptance by *cut-point*. This means that a probabilistic grammar or automaton with probability assignment p is paired with a number c between 0 and 1. The language that is thereby defined consists of all strings w such that $p(w) > c$. For PCFGs this was investigated by [68], and for PPDAs by [7, 69–71].

Assume a PCFG with probability assignment p. If we let $\lambda_\pi = \log_e p(\pi)$ for each rule π, then the probability of a derivation can be rewritten as:

$$p(d) = \prod_{\pi} p(\pi)^{C(\pi,d)} = \prod_{\pi} e^{\lambda_\pi \cdot C(\pi,d)}. \tag{44}$$

This equation stresses that the probability of a derivation is determined only by the frequencies of individual rules occurring in it. We cannot express, say, preference for combinations or patterns of rules. For this, we need to generalise the framework to *log-linear models* [72, 73]. Such a model allows us to specify a number of arbitrary features $c_1, \ldots, c_m$ on derivations. The features map derivations to non-negative numbers. Further, there is an equal number of weights λ_1, $\ldots$, λ_m. The model thereby defines a probability distribution on derivations as:

$$p(d) = \frac{1}{z} \prod_{i} e^{\lambda_i \cdot c_i(d)} = \frac{1}{z} e^{\sum_i \lambda_i \cdot c_i(d)}, \tag{45}$$

where z is a normalisation constant, that is, z is the sum of $e^{\sum_i \lambda_i \cdot c_i(d)}$ for all d that are valid left-most derivations.

References

1. Nederhof, M.J., Satta, G.: Tabular parsing. In Martín-Vide, C., Mitrana, V., Păun, G., eds.: Formal Languages and Applications. Springer (2004) 529–549
2. Collins, M.: Three generative, lexicalised models for statistical parsing. In: 35th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Madrid, Spain (1997) 16–23
3. Eisner, J., Satta, G.: Efficient parsing for bilexical context-free grammars and head automaton grammars. In: 37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Maryland, USA (1999) 457–464
4. Charniak, E.: Statistical Language Learning. MIT Press (1993)
5. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press (1999)
6. Bod, R., Hay, J., Jannedy, S., eds.: Probabilistic Linguistics. MIT Press (2003)
7. Huang, T., Fu, K.: On stochastic context-free languages. Information Sciences **3** (1971) 201–224
8. Hutchins, S.: Moments of strings and derivation lengths of stochastic context-free grammars. Information Sciences **4** (1972) 179–191
9. Booth, T., Thompson, R.: Applying probabilistic measures to abstract languages. IEEE Transactions on Computers **C-22** (1973) 442–450
10. Wetherell, C.: Probabilistic languages: A review and some open questions. Computing Surveys **12** (1980) 361–379
11. Etessami, K., Yannakakis, M.: 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., Stuttgart, Germany, Springer-Verlag (2005) 340–352
12. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. In Bar-Hillel, Y., ed.: Language and Information: Selected Essays on their Theory and Application. Addison-Wesley, Reading, Massachusetts (1964) 116–150
13. Nederhof, M.J., Satta, G.: Probabilistic parsing as intersection. In: 8th International Workshop on Parsing Technologies, LORIA, Nancy, France (2003) 137–148
14. Sippu, S., Soisalon-Soininen, E.: Parsing Theory, Vol. I: Languages and Parsing. Volume 15 of EATCS Monographs on Theoretical Computer Science. Springer-Verlag (1988)
15. Graham, S., Harrison, M., Ruzzo, W.: An improved context-free recognizer. ACM Transactions on Programming Languages and Systems **2** (1980) 415–462
16. Aho, A., Ullman, J.: Parsing. Volume 1 of The Theory of Parsing, Translation and Compiling. Prentice-Hall, Englewood Cliffs, N.J. (1972)
17. Thompson, R.: Determination of probabilistic grammars for functionally specified probability-measure languages. IEEE Transactions on Computers **C-23** (1974) 603–614
18. Abney, S., McAllester, D., Pereira, F.: Relating probabilistic grammars and automata. In: 37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Maryland, USA (1999) 542–549

19. Nederhof, M.J., Satta, G.: Estimation of consistent probabilistic context-free grammars. In: Proceedings of the Human Language Technology Conference of the NAACL, Main Conference, New York, USA (2006) 343–350
20. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press (1990)
21. Martelli, A., Montanari, U.: Optimizing decision trees through heuristically guided search. Communications of the ACM **21** (1978) 1025–1039
22. Jelinek, F., Lafferty, J., Mercer, R.: Basic methods of probabilistic context free grammars. In Laface, P., De Mori, R., eds.: Speech Recognition and Understanding — Recent Advances, Trends and Applications. Springer-Verlag (1992) 345–360
23. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory **IT-13** (1967) 260–269
24. Knuth, D.: A generalization of Dijkstra's algorithm. Information Processing Letters **6** (1977) 1–5
25. Nederhof, M.J.: Weighted deductive parsing and Knuth's algorithm. Computational Linguistics **29** (2003) 135–143
26. Klein, D., Manning, C.: Parsing and hypergraphs. In: Proceedings of the Seventh International Workshop on Parsing Technologies, Beijing, China (2001)
27. Jiménez, V., Marzal, A.: Computation of the n best parse trees for weighted and stochastic context-free grammars. In: Advances in Pattern Recognition. Volume 1876 of Lecture Notes in Computer Science., Alicante, Spain, Springer-Verlag (2000) 183–192
28. Nielsen, L., Pretolani, D., Andersen, K.: Finding the k shortest hyperpaths using reoptimization. Operations Research Letters **34** (2006) 155–164
29. Huang, L., Chiang, D.: Better k-best parsing. In: Proceedings of the Ninth International Workshop on Parsing Technologies, Vancouver, British Columbia, Canada (2005) 53–64
30. Resnik, P.: Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In: Proc. of the fifteenth International Conference on Computational Linguistics. Volume 2., Nantes (1992) 418–424
31. Schabes, Y.: Stochastic lexicalized tree-adjoining grammars. In: Proc. of the fifteenth International Conference on Computational Linguistics. Volume 2., Nantes (1992) 426–432
32. Sima'an, K.: Computational complexity of probabilistic disambiguation. Grammars **5** (2002) 125–151
33. Casacuberta, F., de la Higuera, C.: Computational complexity of problems on probabilistic grammars and transducers. In Oliveira, A., ed.: Grammatical Inference: Algorithms and Applications. Volume 1891 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2000) 15–24
34. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.: Probabilistic finite-state machines — part I. IEEE Transactions on Pattern Analysis and Machine Intelligence **27** (2005) 1013–1025
35. Paz, A.: Introduction to Probabilistic Automata. Academic Press, New York (1971)
36. Blondel, V., Canterini, C.: Undecidable problems for probabilistic automata of fixed dimension. Theory of Computing systems **36** (2003) 231–245
37. Chi, Z., Geman, S.: Estimation of probabilistic context-free grammars. Computational Linguistics **24** (1998) 299–305
38. Chaudhuri, R., Pham, S., Garcia, O.: Solution of an open problem on probabilistic grammars. IEEE Transactions on Computers **C-32** (1983) 748–750

39. Sánchez, J.A., Benedí, J.M.: Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997) 1052–1055
40. Chen, S., Goodman, J.: An empirical study of smoothing techniques for language modeling. In: 34th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Santa Cruz, California, USA (1996) 310–318
41. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. Royal Statistical Society Series **B 39** (1977) 1–38
42. Baker, J.: Trainable grammars for speech recognition. In Wolf, J., Klatt, D., eds.: Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America. (1979) 547–550
43. Lari, K., Young, S.: The estimation of stochastic context-free grammars using the Inside-Outside algorithm. Computer Speech and Language **4** (1990) 35–56
44. Prescher, D.: Inside-outside estimation meets dynamic EM. In: Proceedings of the Seventh International Workshop on Parsing Technologies, Beijing, China (2001)
45. Baum, L.: An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. Inequalities **3** (1972) 1–8
46. Kupiec, J.: Hidden Markov estimation for unrestricted stochastic context-free grammars. In: ICASSP'92. Volume I. (1992) 177–180
47. Jelinek, F., Lafferty, J.: Computation of the probability of initial substring generation by stochastic context-free grammars. Computational Linguistics **17** (1991) 315–323
48. Stolcke, A.: An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computational Linguistics **21** (1995) 167–201
49. Persoon, E., Fu, K.: Sequential classification of strings generated by SCFG's. International Journal of Computer and Information Sciences **4** (1975) 205–217
50. Nederhof, M.J., Sarkar, A., Satta, G.: Prefix probabilities from stochastic tree adjoining grammars. In: 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics. Volume 2., Montreal, Quebec, Canada (1998) 953–959
51. Nederhof, M.J., Sarkar, A., Satta, G.: Prefix probabilities for linear indexed grammars. In: Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks, Institute for Research in Cognitive Science, University of Pennsylvania (1998) 116–119
52. Nederhof, M.J., Satta, G.: Probabilistic parsing strategies. Journal of the ACM **53** (2006) 406–436
53. Santos, E.: Probabilistic grammars and automata. Information and Control **21** (1972) 27–47
54. Chitrao, M., Grishman, R.: Statistical parsing of messages. In: Speech and Natural Language, Proceedings, Hidden Valley, Pennsylvania (1990) 263–266
55. Johnson, M.: PCFG models of linguistic tree representations. Computational Linguistics **24** (1998) 613–632
56. Tendeau, F.: Stochastic parse-tree recognition by a pushdown automaton. In: Fourth International Workshop on Parsing Technologies, Prague and Karlovy Vary, Czech Republic (1995) 234–249
57. Nederhof, M.J., Satta, G.: An alternative method of training probabilistic LR parsers. In: 42nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Barcelona, Spain (2004) 551–558
58. Goodman, J.: Semiring parsing. Computational Linguistics **25** (1999) 573–605

59. Kuich, W.: Semirings and formal power series: their relevance to formal languages and automata. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages, Vol. 1. Springer, Berlin (1997) 609–677
60. Mohri, M.: Statistical natural language processing. In Lothaire, M., ed.: Applied Combinatorics on Words. Cambridge University Press (2005) 210–240
61. Teitelbaum, R.: Context-free error analysis by evaluation of algebraic power series. In: Conference Record of the Fifth Annual ACM Symposium on Theory of Computing. (1973) 196–199
62. Wagner, R., Fischer, M.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–173
63. Lang, B.: A generative view of ill-formed input processing. In: ATR Symposium on Basic Research for Telephone Interpretation, Kyoto, Japan (1989)
64. Aho, A., Peterson, T.: A minimum distance error-correcting parser for context-free languages. SIAM Journal on Computing **1** (1972) 305–312
65. Lyon, G.: Syntax-directed least-errors analysis for context-free languages: A practical approach. Communications of the ACM **17** (1974) 3–14
66. Lavie, A., Tomita, M.: GLR* – an efficient noise-skipping parsing algorithm for context free grammars. In: Third International Workshop on Parsing Technologies, Tilburg (The Netherlands) and Durbuy (Belgium) (1993) 123–134
67. Lavie, A.: An integrated heuristic scheme for partial parse evaluation. In: 32nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Las Cruces, New Mexico, USA (1994) 316–318
68. Salomaa, A.: Probabilistic and weighted grammars. Information and Control **15** (1969) 529–544
69. Fu, K., Huang, T.: Stochastic grammars and languages. International Journal of Computer and Information Sciences **1** (1972) 135–170
70. Santos, E.: Probabilistic pushdown automata. Journal of Cybernetics **6** (1976) 173–187
71. Freivalds, R.: Probabilistic machines can use less running time. In: Proceedings of IFIP Congress 77, Toronto (1977) 839–842
72. Berger, A., Della Pietra, S., Della Pietra, V.: A maximum entropy approach to natural language processing. Computational Linguistics **22** (1996) 39–71
73. Chi, Z.: Statistical properties of probabilistic context-free grammars. Computational Linguistics **25** (1999) 131–160