A Fast Algorithm for Computing Prefix Probabilities

Franz Nowak   Ryan Cotterell
ETH Zürich
{`fnowak`, `rcotterell`}@ethz.ch

Abstract

Multiple algorithms are known for efficiently calculating the prefix probability of a string under a probabilistic context-free grammar (PCFG). Good algorithms for the problem have a runtime cubic in the length of the input string. However, some proposed algorithms are suboptimal with respect to the size of the grammar. This paper proposes a novel speed-up of Jelinek and Lafferty's (1991) algorithm, which runs in $\mathcal{O}(N^3|\mathcal{N}|^3 + |\mathcal{N}|^4)$, where N is the input length and $|\mathcal{N}|$ is the number of non-terminals in the grammar. In contrast, our speed-up runs in $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$.

 https://github.com/rycolab/
prefix-parsing

1 Introduction

Probabilistic context-free grammars (PCFGs) are an important formalism in NLP (Eisenstein, 2019, Chapter 10). One common use of PCFGs is to construct a language model. For instance, PCFGs form the backbone of many neural language models, e.g., recurrent neural network grammars (RNNGs; Dyer et al., 2016; Dyer, 2017; Kim et al., 2019). However, in order to use a PCFG as a language model, one needs to be able to compute prefix probabilities, i.e., the probability that the yield of a derivation starts with the given string. In notation, given a string $\boldsymbol{w} = w_1 \cdots w_N$, we seek the probability $p(\mathrm{S} \overset{*}{\Rightarrow} \boldsymbol{w} \cdots)$ where S is the distinguished start symbol of the grammar and $\overset{*}{\Rightarrow}$ is the closure over applications of derivation rules of the grammar.[1] Our paper gives a more efficient algorithm for the simultaneous computation of the prefix probabilities of *all* prefixes of a string $\boldsymbol{w}$ under a PCFG.

The authors are aware of two existing efficient algorithms to compute prefix probabilities under a PCFG.[2] The first is Jelinek and Lafferty's (1991)

algorithm which is derived from CKY (Kasami, 1965; Younger, 1967; Cocke and Schwartz, 1970) and, thus, requires the grammar to be in Chomsky normal form (CNF). Jelinek–Lafferty runs in $\mathcal{O}(N^3|\mathcal{N}|^3 + |\mathcal{N}|^4)$ time, where N is the length of the input and $\mathcal{N}$ is the number of non-terminals of the grammar, slower than the $\mathcal{O}(N^3|\mathcal{N}|^3)$ required for parsing with CKY, when the number of non-terminals $|\mathcal{N}|$ is taken into account.

The second, due to Stolcke (1995), is derived from Earley parsing (Earley, 1970) and can parse arbitrary PCFGs,[3] with a runtime of $\mathcal{O}(N^3|\mathcal{N}|^3)$. Many previous authors have improved the runtime of Earley's (Graham et al., 1980; Leermakers et al., 1992; Moore, 2000, *inter alia*), and Opedal et al. (2023) successfully applied this speed-up to computing prefix probabilities, achieving a runtime of $\mathcal{O}(N^3|\mathcal{G}|)$, where $|\mathcal{G}|$ is the size of the grammar, that is, the sum of the number of symbols in all production rules.

Our paper provides a more efficient version of Jelinek and Lafferty (1991) for the computation of prefix probabilities under a PCFG in CNF. Specifically, we give an $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$ time algorithm, which is the fastest attested in the literature for dense grammars in CNF,[4] matching the complexity of CKY adapted for dense grammars by Eisner and Blatz (2007).[5] We provide a full derivation and proof of correctness, as well as an open-source implementation on GitHub. We also briefly discuss how our improved algorithm can be extended to work for semiring-weighted CFGs.

2 Preliminaries

We start by introducing the necessary background on probabilistic context-free grammars.

---

[1] Specifically, $\boldsymbol{\alpha} \overset{*}{\Rightarrow} \boldsymbol{\beta}$ means that there exists an $n \geq 0$ such that $\boldsymbol{\alpha} \underbrace{\Rightarrow \cdots \Rightarrow}_{n \text{ times}} \boldsymbol{\beta}$, where $\Rightarrow$ marks a derivation step.

[2] Upon publication of this work, the authors were made aware of two other algorithms for finding prefix probabilities in the special case of idempotent semirings (Corazza et al. 1994; Sánchez and Benedí 1997). See App. B for a discussion of prefix parsing under a semiring.

[3] Note that Earley's and, by extension, Stolcke's algorithms also implicitly binarize the grammar during execution by using dotted rules as additional non-terminals.

[4] A PCFG in CNF is dense if for every $X, Y, Z \in \mathcal{N}$, we have a production rule $X \to Y\,Z \in \mathcal{R}$.

[5] Note that there exist approximate parsing algorithms with lower complexity bounds (Cohen et al., 2013). Moreover, there are parsing algorithms that asymptotically run in sub-cubic time in the input length using fast matrix multiplication (Valiant, 1975; Benedí and Sánchez, 2007). However, they are of limited practical use (Lee, 1997).

Definition 1. *A **probabilistic context-free grammar** (PCFG) is a five-tuple $\mathcal{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{R}, p)$, made up of:*

- *An finite set of non-terminal symbols $\mathcal{N}$;*

- *An alphabet of terminal symbols Σ;*

- *A distinguished start symbol $\mathrm{S} \in \mathcal{N}$;*

- *A finite set of production rules $\mathcal{R} \subset \mathcal{N} \times (\mathcal{N} \cup \Sigma)^*$ where each rule is written as $\mathrm{X} \to \boldsymbol{\alpha}$ with $\mathrm{X} \in \mathcal{N}$ and $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$. Here, * denotes the Kleene closure;*

- *A weighting function $p : \mathcal{R} \to [0, 1]$ assigning each rule $r \in \mathcal{R}$ a probability such that p is **locally normalized**, meaning that for all $\mathrm{X} \in \mathcal{N}$ that appear on the left-hand side of a rule, $\sum_{\mathrm{X} \to \boldsymbol{\alpha} \in \mathcal{R}} p(\mathrm{X} \to \boldsymbol{\alpha}) = 1$.*

Note that not every locally normalized PCFG constitutes a valid distribution over Σ^*. Specifically, some may place probability mass on infinite trees (Chi and Geman, 1998). PCFGs that *do* constitute a valid distribution over Σ^* are referred to as **tight**. Furthermore, if all non-terminals of the grammar can be reached from the start non-terminal via production rules, we say the PCFG is **trim**.

Definition 2. *A PCFG $\mathcal{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{R}, p)$ is in **Chomsky normal form** (CNF) if each production rule in $\mathcal{R}$ is in one of the following forms:*

$$\mathrm{X} \to \mathrm{Y}\,\mathrm{Z} \qquad (1)$$
$$\mathrm{X} \to a \qquad (2)$$
$$\mathrm{S} \to \varepsilon \qquad (3)$$

where $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \mathcal{N}$ are non-terminals, $a \in \Sigma$ are terminal symbols, and ε is the empty string.

Definition 3. *A **derivation step** $\boldsymbol{\alpha} \Rightarrow \boldsymbol{\beta}$ is an application of the binary relation $\Rightarrow : (\mathcal{N} \cup \Sigma)^* \times (\mathcal{N} \cup \Sigma)^*$, which rewrites the left-most non-terminal in $\boldsymbol{\alpha}$ according to a rule in $\mathcal{R}$ from the left-hand side of that rule to its right-hand side, resulting in $\boldsymbol{\beta}$.*

Definition 4. *A **derivation** under a grammar $\mathcal{G}$ is a sequence $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_m$, where $\boldsymbol{\alpha}_0 \in \mathcal{N}, \boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_{m-1} \in (\mathcal{N} \cup \Sigma)^*$, and $\boldsymbol{\alpha}_m \in \Sigma^*$, in which each $\boldsymbol{\alpha}_{i+1}$ is formed by applying a derivation step to $\boldsymbol{\alpha}_i$. $\boldsymbol{\alpha}_m = w_1 \cdots w_N \in \Sigma^*$ is called the **yield** of the derivation. If $\boldsymbol{\alpha}_0$ is not the start symbol S, we call it a **partial derivation**. We write $\boldsymbol{\alpha}_0 \overset{*}{\Rightarrow} w_1 \cdots w_N$, where $\overset{*}{\Rightarrow}$ is the closure over the binary relation $\Rightarrow$ introduced in definition 3.*

We represent derivations as trees whose structure corresponds to production rules, where any parent node is the non-terminal on the left-hand side of a rule and its children are the symbols from the right-hand side. The leaves of the tree, when read from left to right, form the yield. Such a tree, when rooted S, is called a **derivation tree**. Otherwise, it is called a **derivation subtree**.

Definition 5. *The probability of a derivation tree (or derivation subtree) $\boldsymbol{\tau}$ is the product of the probabilities of all its corresponding production rules:*

$$p(\boldsymbol{\tau}) \overset{\text{def}}{=} \prod_{(\boldsymbol{\alpha} \to \boldsymbol{\beta}) \in \boldsymbol{\tau}} p(\boldsymbol{\alpha} \to \boldsymbol{\beta}) \qquad (4)$$

Definition 6. *We define $\mathcal{T}_{\mathrm{X}}(w_i \cdots w_k)$ as the set of all derivation subtrees $\boldsymbol{\tau}$ rooted at X with yield $w_i \cdots w_k$.*

Definition 7. *Given a PCFG $\mathcal{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{R}, p)$, a non-terminal $\mathrm{X} \in \mathcal{N}$, and a string $\boldsymbol{w} = w_1 \cdots w_N \in \Sigma^*$, the **inside probability** of X between indices i and k (where $1 \le i \le k \le N$) is defined as:*

$$\beta(i, k \mid \mathrm{X}) \overset{\text{def}}{=} p(\mathrm{X} \overset{*}{\Rightarrow} w_i \cdots w_k) \qquad (5)$$
$$= \sum_{\boldsymbol{\tau} \in \mathcal{T}_{\mathrm{X}}(w_i \cdots w_k)} p(\boldsymbol{\tau}) \qquad (6)$$

That is, the sum of the probability of all derivation trees $\boldsymbol{\tau}$ starting at X that have yield $w_i \cdots w_k$.

Definition 8. *Given a PCFG $\mathcal{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{R}, p)$, a non-terminal $\mathrm{X} \in \mathcal{N}$, and a string $\boldsymbol{w} = w_1 \cdots w_N \in \Sigma^*$, we define the **prefix probability** p_π, i.e., the probability of $\boldsymbol{w}$ being a prefix under $\mathcal{G}$, to be:*

$$p_\pi(\boldsymbol{w} \mid \mathrm{X}) \overset{\text{def}}{=} \sum_{\boldsymbol{u} \in \Sigma^*} p(\mathrm{X} \overset{*}{\Rightarrow} \boldsymbol{w}\boldsymbol{u}) \qquad (7)$$

In words, p_π is the probability of deriving $\boldsymbol{w}$ with an arbitrary continuation from X, that is, the sum of probabilities of deriving $\boldsymbol{w}\boldsymbol{u}$ from X over all possible suffixes $\boldsymbol{u} \in \Sigma^*$. In the following, we write the prefix probability of deriving prefix $\boldsymbol{w} = w_i \cdots w_k$ from X as $p_\pi(i, k \mid \mathrm{X})$.

Definition 9. *Let $\mathcal{G}$ be a PCFG in CNF. Then for non-terminals $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \mathcal{N}$, the **left-corner** expectations $E_{\mathrm{lc}}(\mathrm{Y} \mid \mathrm{X})$ and $E_{\mathrm{lc}}(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X})$ are, respectively, defined as:*

$$E_{\mathrm{lc}}(\mathrm{Y} \mid \mathrm{X}) \overset{\text{def}}{=} \sum_{\boldsymbol{\alpha} \in \mathcal{N}^*} p(\mathrm{X} \overset{*}{\Rightarrow} \mathrm{Y}\boldsymbol{\alpha}) \qquad (8)$$

$$E_{\mathrm{lc}}(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X}) \overset{\text{def}}{=} \sum_{\mathrm{X}' \in \mathcal{N}} E_{\mathrm{lc}}(\mathrm{X}' \mid \mathrm{X}) \cdot p(\mathrm{X}' \to \mathrm{Y}\,\mathrm{Z})$$
$$(9)$$

Algorithm 1 CKY

```
1: def CKY(w = w_1 ··· w_N, G):
2:      ▷ Initialize inside probabilities
3:      β(·, · | ·) ← 0
4:      for k ∈ 1, ..., N :
5:          for X → w_k ∈ R :
6:              ▷ Handle single word tokens
7:              β(k, k | X) ← β(k, k | X) + p(X → w_k)
8:      ▷ ℓ is the span size
9:      for ℓ ∈ 2, ..., N :
10:         ▷ i marks the beginning of the span
11:         for i ∈ 1, ..., N − ℓ + 1 :
12:             ▷ k marks the end of the span
13:             k ← i + ℓ − 1
14:             ▷ Recursively compute β
15:             for X → Y Z ∈ R :
16:                 β(i, k | X) ← β(i, k | X) + p(X → 
        Y Z) · Σ_{j=i}^{k−1} β(i, j | Y) · β(j+1, k | Z)
17:     return β
```

Algorithm 2 Jelinek–Lafferty

```
1:  def JL(w = w_1 ··· w_N, G):
2:      p_π(·, · | ·) ← 0          ▷ Initialize prefix probabilities
3:      β ← CKY(w)       ▷ Precompute β with Algorithm 1
4:      for X_i, X_j ∈ N :          ▷ Precompute E_lc(Y | X)
5:          E_lc(X_j | X_i) ← [(I − P)^{−1}]_{ij}
6:      for X′ → Y Z ∈ R :  ▷ Precompute E_lc(Y Z | X)
7:          E_lc(Y Z | X) ← Σ_{X∈N} E_lc(X′|X) · p(X′→Y Z)
8:      for k ∈ 1, ..., N :
9:          for X ∈ N :              ▷ Compute base case
10:             p_π(k, k | X) ← Σ_{Y∈N} E_lc(Y|X) · p(Y→w_k)
11:     for ℓ ∈ 2 ... N :
12:         for i ∈ 1 ... N − ℓ + 1 :
13:             k ← i + ℓ − 1
14:             for X, Y, Z ∈ N :  ▷ Recursively compute p_π
15:                 p_π(i, k | X) ← p_π(i, k | X) +
        E_lc(Y Z|X) · Σ_{j=i}^{k−1} β(i, j | Y) · p_π(j+1, k | Z)
16:     return p_π
```

Figure 1: Pseudocode for the CKY algorithm (left) and Jelinek–Lafferty (right)

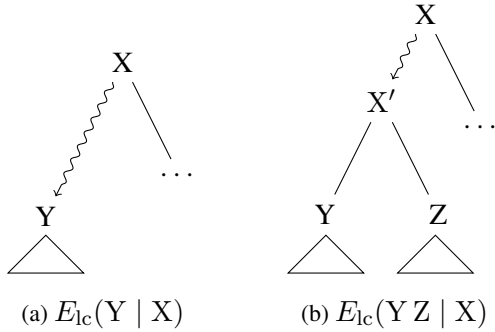(a) $E_{lc}(Y \mid X)$ (b) $E_{lc}(Y Z \mid X)$

Figure 2: Visualization of left-corner expectations

The left-corner expectation $E_{lc}(Y \mid X)$ is hence the sum of the probabilities of partial derivation subtrees rooted in X that have Y as the left-most leaf; see Fig. 2a for a visualization. Similarly, $E_{lc}(Y Z \mid X)$ is the sum of the probabilities of partial derivation subtrees that have Y and Z as the leftmost leaves; see Fig. 2b.

3 Jelinek and Lafferty (1991)

We now give a derivation of the Jelinek–Lafferty algorithm. The first step is to derive an expression for the prefix probability in PCFG terms.

Lemma 1. *Given a tight, trim PCFG in CNF and a string $w = w_1 \cdots w_N$, the prefix probability of a substring $w_i \cdots w_k$ of w, can be defined recursively as follows:*

$$
p_\pi(i, k \mid X) = \sum_{Y,Z \in N} E_{lc}(Y Z \mid X) \\
\cdot \sum_{j=i}^{k-1} \beta(i, j \mid Y) \cdot p_\pi(j+1, k \mid Z)
\tag{10}
$$

Proof. A proof of Lemma 1 is given in App. A.

The above formulation of the prefix probability is closely related to that of the inside probability from Baker's (1979) inside–outside algorithm, which can be efficiently computed using CKY, see Algorithm 1.

Next, the left-corner expectations E_{lc} as defined by Eq. (8) can be computed efficiently as follows. Let P denote the square matrix of dimension $|N|$, with rows and columns indexed by the non-terminals N (in some fixed order), where the entry at the ith row and the jth column corresponds to $p(X_i \to X_j \bullet)$, i.e., the probability of deriving

X_j on the left corner from X_i in one step:

$$p(X_i \to X_j \bullet) \overset{\text{def}}{=} \sum_{Y \in \mathcal{N}} p(X_i \to X_j \, Y) \quad (11)$$

We can find the probability of getting to non-terminal X_j after k derivation steps starting from X_i by multiplying P with itself k times:

$$p(X_i \xrightarrow{k} X_j \bullet) = (P^k)_{i,j} \quad (12)$$

We can hence get the matrix P^*, whose entries correspond to deriving X_j from X_i after *any* number of derivation steps, by summing over all the powers of the matrix P:[6]

$$P^* \overset{\text{def}}{=} I + P + P^2 + P^3 + \cdots = \sum_{n=0}^{\infty} P^n \quad (13)$$

$$= I + P \sum_{n=0}^{\infty} P^n = I + PP^* = (I - P)^{-1}$$

Note that the entry at the ith row and j^{th} column of P^* is exactly the left-corner expectation $E_{\text{lc}}(X_j \mid X_i)$. Finally, we can compute the left-corner expectations $E_{\text{lc}}(Y\,Z \mid X)$ using Eq. (9):

$$E_{\text{lc}}(Y\,Z \mid X) \overset{\text{def}}{=} \sum_{X' \in \mathcal{N}} E_{\text{lc}}(X' \mid X) \cdot p(X' \to Y\,Z)$$

Similarly, we can compute the base case of the recursive Eq. (10), namely $p_\pi(k, k \mid X)$, which is defined as follows.

Definition 10. *The prefix probability of the token at position k being derived from X is defined as:*

$$p_\pi(k, k \mid X) \overset{\text{def}}{=} \sum_{Y \in \mathcal{N}} E_{\text{lc}}(Y \mid X) \cdot p(Y \to w_k) \quad (14)$$

We can now combine the quantities derived above to obtain an efficient algorithm for the computation of prefix probabilities $p_\pi(i, k \mid S)$. For the full algorithm, see Algorithm 2.

Proposition 1. *The time complexity of the CKY algorithm as presented in Algorithm 1 is $\mathcal{O}(N^3 |\mathcal{N}|^3)$.*

Proof. Clearly, the computationally critical part is in lines 9–13, where we iterate over all indices of w for i, j, and k, as well as over the whole set of grammar rules, thus taking $\mathcal{O}(N^3 |\mathcal{R}|)$. In a PCFG in CNF, with the size of the alphabet taken as constant, the number of rules, $|\mathcal{R}|$, is $\mathcal{O}(|\mathcal{N}|^3)$, making the overall complexity of CKY $\mathcal{O}(N^3 |\mathcal{N}|^3)$. ∎

[6]Note that this sum converges if the PCFG is tight and trim since infinite derivation (sub)trees have zero probability mass.

Proposition 2. *The total time complexity of Jelinek–Lafferty is $\mathcal{O}(N^3 |\mathcal{N}|^3 + |\mathcal{N}|^4)$:*

Proof.   1. We begin by pre-computing all the inside probabilities β in line 2 of Algorithm 2, which takes $\mathcal{O}(N^3 |\mathcal{N}|^3)$ by Proposition 1.

2. Next, in lines 3–4, we pre-compute all the left-corner expectations $E_{\text{lc}}(Y \mid X)$ using Eq. (13), which has the complexity of inverting the matrix P, i.e., $\mathcal{O}(|\mathcal{N}|^3)$.

3. In lines 5–7, we then use Eq. (9) to compute $E_{\text{lc}}(Y\,Z \mid X)$, iterating once over all non-terminals X for each rule, which takes $\mathcal{O}(|\mathcal{R}||\mathcal{N}|)$, that is, $\mathcal{O}(|\mathcal{N}|^4)$.

4. Computing $p_\pi(k, k \mid X)$ for all $X \in \mathcal{N}$ by Eq. (14) in lines 8–10 takes $\mathcal{O}(N|\mathcal{N}|^2)$ as we iterate over all positions $k \in N$ and over all $Y \in \mathcal{N}$ for each $X \in \mathcal{N}$.

5. And finally, computing the p_π chart in lines 11–14 takes $\mathcal{O}(N^3 |\mathcal{N}|^3)$ since we iterate over all $\ell, i, j \leq N$ and $X, Y, Z \in \mathcal{N}$.

6. This yields an overall time complexity of $\mathcal{O}(N^3 |\mathcal{N}|^3 + |\mathcal{N}|^4)$.

∎

4   Our Speed-up

We now turn to our development of a faster dynamic program to compute all prefix probabilities. The speed-up comes from a different way to factorize $p_\pi(i, k \mid X)$, which allows additional memoization. Starting with the definition of the prefix probability in Eq. (15a), we first expand $E_{\text{lc}}(Y\,Z \mid X)$ by Eq. (9), as seen in Eq. (15b). Then, we factor out all terms that depend on the left-corner non-terminal Y in Eq. (15c), which we store in a chart γ, see Eq. (15e). We then do the same for all terms depending on X', factoring them out in Eq. (15d) and storing them in another chart δ, see Eq. (15f).

Our improved algorithm for computing all prefix probabilities is shown in Algorithm 3.

Proposition 3. *The complexity of our improved algorithm is $\mathcal{O}(N^2 |\mathcal{N}|^3 + N^3 |\mathcal{N}|^2)$.*

Proof.   1. As before, computing $E_{\text{lc}}(Y \mid X)$ and $p_\pi(k, k \mid X)$ takes $\mathcal{O}(|\mathcal{N}|^3)$ and $\mathcal{O}(N|\mathcal{N}|^2)$, respectively.

$$p_\pi(i, k \mid \mathrm{X}) = \sum_{\mathrm{Y},\mathrm{Z}\in\mathcal{N}} E_{\mathrm{lc}}(\mathrm{Y},\mathrm{Z}\mid\mathrm{X}) \cdot \sum_{j=i}^{k-1} \beta(i,j\mid\mathrm{Y}) \cdot p_\pi(j{+}1, k\mid\mathrm{Z}) \tag{15a}$$

$$= \sum_{\mathrm{Y},\mathrm{Z}\in\mathcal{N}} \sum_{\mathrm{X}'\in\mathcal{N}} E_{\mathrm{lc}}(\mathrm{X}'\mid\mathrm{X}) \cdot p(\mathrm{X}'\to\mathrm{Y}\,\mathrm{Z}) \cdot \sum_{j=i}^{k-1} \beta(i,j\mid\mathrm{Y}) \cdot p_\pi(j{+}1, k\mid\mathrm{Z}) \tag{15b}$$

$$= \sum_{\mathrm{X}',\mathrm{Z}\in\mathcal{N}} E_{\mathrm{lc}}(\mathrm{X}'\mid\mathrm{X}) \cdot \sum_{j=i}^{k-1} \gamma_{ij}(\mathrm{X}',\mathrm{Z}) \cdot p_\pi(j{+}1, k\mid\mathrm{Z}) \tag{15c}$$

$$= \sum_{\mathrm{Z}\in\mathcal{N}} \sum_{j=i}^{k-1} \delta_{ij}(\mathrm{X},\mathrm{Z}) \cdot p_\pi(j{+}1, k\mid\mathrm{Z}) \tag{15d}$$

$$\text{where } \gamma_{ij}(\mathrm{X}',\mathrm{Z}) \stackrel{\text{def}}{=} \sum_{\mathrm{Y}\in\mathcal{N}} p(\mathrm{X}'\to\mathrm{Y}\,\mathrm{Z}) \cdot \beta(i,j\mid\mathrm{Y}) \tag{15e}$$

$$\text{and } \delta_{ij}(\mathrm{X},\mathrm{Z}) \stackrel{\text{def}}{=} \sum_{\mathrm{X}'\in\mathcal{N}} E_{\mathrm{lc}}(\mathrm{X}'\mid\mathrm{X}) \cdot \gamma_{ij}(\mathrm{X}',\mathrm{Z}) \tag{15f}$$

Algorithm 3 Faster prefix probability algorithm

1: **def** FastJL($\boldsymbol{w} = w_1 \cdots w_N, \mathcal{G}$):
2:    $p_\pi(\cdot, \cdot \mid \cdot) \leftarrow 0$ ▷ *Initialize prefix probabilities*
3:    $\beta \leftarrow \mathrm{CKY}(\boldsymbol{w})$ ▷ *Precompute β with Algorithm 1*
4:    **for** $\mathrm{X}_i, \mathrm{X}_j \in \mathcal{N}$:   ▷ *Precompute $E_{\mathrm{lc}}(\mathrm{Y}\mid\mathrm{X})$*
5:      $E_{\mathrm{lc}}(\mathrm{X}_j \mid \mathrm{X}_i) \leftarrow \big[(I-P)^{-1}\big]_{ij}$
6:    **for** $i, j = 1, \ldots, N$:
7:      **for** $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$:   ▷ *Precompute γ by Eq. (15e)*
8:        $\gamma_{ij}(\mathrm{X},\mathrm{Z}) \leftarrow \sum_{\mathrm{Y}\in\mathcal{N}} p(\mathrm{X}{\to}\mathrm{YZ}) \cdot \beta(i,j\mid\mathrm{Y})$
9:      **for** $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$:   ▷ *Precompute δ by Eq. (15f)*
10:       $\delta_{ij}(\mathrm{X},\mathrm{Z}) \leftarrow \sum_{\mathrm{Y}\in\mathcal{N}} E_{\mathrm{lc}}(\mathrm{Y}\mid\mathrm{X}) \cdot \gamma_{ij}(\mathrm{Y},\mathrm{Z})$
11:    **for** $k \in 1, \ldots, N$, **for** $\mathrm{X} \in \mathcal{N}$:   ▷ *Base case*
12:      $p_\pi(k, k \mid \mathrm{X}) \leftarrow \sum_{\mathrm{Y}\in\mathcal{N}} E_{\mathrm{lc}}(\mathrm{Y}\mid\mathrm{X}) \cdot p(\mathrm{Y}{\to}w_k)$
13:    **for** $\ell \in 2 \ldots N$:
14:      **for** $i \in 1 \ldots N - \ell + 1$:
15:       $k \leftarrow i + \ell - 1$
16:       **for** $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$:   ▷ *Recursively compute p_π*
17:        $p_\pi(i, k \mid \mathrm{X}) \leftarrow p_\pi(i, k \mid \mathrm{X}) + \sum_{j=i}^{k-1} \delta_{ij}(\mathrm{X},\mathrm{Z}) \cdot p_\pi(j{+}1, k \mid \mathrm{Z})$
18:    **return** p_π

---

2. As Eisner and Blatz (2007) show, one can compute β in $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$, thus improving the runtime of Algorithm 1 for dense grammars.

3. Pre-computing γ and δ in lines 5–9 takes $\mathcal{O}(N^2|\mathcal{N}|^3)$, as we sum over non-terminals, and both charts each have two dimensions indexing N and two indexing $\mathcal{N}$.

4. The loops computing p_π in lines 13–17 take $\mathcal{O}(N^3|\mathcal{N}|^2)$, as we are now iterating over $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$ and $\ell, i, j \leq N$.

5. Hence, our new overall time complexity is $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$. ∎

5 Generalization to Semirings

It turns out that Jelinek–Lafferty, and, by extension, our improved algorithm, can be generalized to work for semiring-weighted CFGs, with the same time complexity, under the condition that the weights are locally normalized and the semiring has a well-defined Kleene closure. This follows from the fact that the only operations used by the algorithm are addition and multiplication if we use Lehmann's (1977) algorithm for the computation of left-corner expectations, E_{lc}. The definitions, derivation, and proof of this statement can be found in App. B.

6 Conclusion

In this paper, we have shown how to efficiently compute prefix probabilities for PCFGs in CNF, adapting Jelinek–Lafferty to use additional memoization, thereby reducing the time complexity from $\mathcal{O}(N^3|\mathcal{N}|^3 + |\mathcal{N}|^4)$ to $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$. We thereby addressed one of the main limitations of the original formulation, of being slow for large grammar sizes.

7   Limitations

While we have improved the asymptotic running time of a classic algorithm with regard to grammar size, the time complexity of our algorithm is still cubic in the length of the input. Our result follows the tradition of dynamic programming algorithms that trade time for space by memoizing and reusing pre-computed intermediate results. The usefulness of this trade-off in practice depends on the specifics of the grammar, and while the complexity is strictly better in terms of non-terminals, it will be most noticeable for denser grammars with many non-terminals.

8   Ethics Statement

We do not foresee any ethical issues arising from this work.

9   Acknowledgements

References

J. K. Baker. 1979. Trainable grammars for speech recognition. In *Speech Communication Papers presented at the 97th Meeting of the Acoustical Society of America*, pages 547–550, MIT, Cambridge, Massachusetts.

José-Miguel Benedí and Joan-Andreu Sánchez. 2007. Fast stochastic context-free parsing: A stochastic version of the valiant algorithm. In *Pattern Recognition and Image Analysis*, pages 80–88, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

John Cocke and J.T. Schwartz. 1970. *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University.

Shay B. Cohen, Giorgio Satta, and Michael Collins. 2013. Approximate PCFG parsing using tensor decomposition. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 487–496, Atlanta, Georgia. Association for Computational Linguistics.

A. Corazza, R. De Mori, R. Gretter, and G. Satta. 1994. Optimal probabilistic evaluation functions for search controlled by stochastic context-free grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10):1018–1027.

Chris Dyer. 2017. Should neural network architecture reflect linguistic structure? In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, page 1, Vancouver, Canada. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Jacob Eisenstein. 2019. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning series. MIT Press.

Jason Eisner and John Blatz. 2007. Program transformations for optimization of parsing algorithms and other weighted logic programs. In *Proceedings of FG 2006: The 11th Conference on Formal Grammar*, pages 45–85. CSLI Publications.

Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345.

Susan L. Graham, Michael Harrison, and Walter L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462.

Frederick Jelinek and John D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–353.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. In *Technical Report, Air Force Cambridge Research Lab, Bedford, MA*.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1105–1117, Minneapolis, Minnesota. Association for Computational Linguistics.

Lillian Lee. 1997. Fast context-free parsing requires fast Boolean matrix multiplication. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 9–15, Madrid, Spain. Association for Computational Linguistics.

M. C. J. Leermakers, A. Augusteijn, and F.E.J. Kruseman Aretz. 1992. A functional LR parser. *Theoretical Computer Science*, 104(2):313–323.

Daniel J. Lehmann. 1977. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76.

Robert C. Moore. 2000. Time as a measure of parsing efficiency. In *Proceedings of the COLING-2000 Workshop on Efficiency In Large-Scale Parsing Systems*, pages 23–28, Centre Universitaire, Luxembourg. International Committee on Computational Linguistics.

Andreas Opedal, Ran Zmigrod, Tim Vieira, Ryan Cotterell, and Jason Eisner. 2023. Efficient semiring-weighted Earley parsing. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, Toronto, Canada.

Bernard Roy. 1959. Transitivité et connexité. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 249:216–218.

Grzegorz Rozenberg and Arto Salomaa, editors. 1997. *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer-Verlag, Berlin, Heidelberg.

Joan-Andreu Sánchez and José-Miguel Benedí. 1997. Computation of the probability of the best derivation of an initial substring from a stochastic context-free grammar. *Proceedings of the VII Spanish Symposium on Pattern Recognition and Image Analysis*, pages 181–186.

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

Leslie G. Valiant. 1975. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315.

Stephen Warshall. 1962. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

A   Proof of Lemma 1

Lemma 1. *Given a tight, trim PCFG in CNF and a string $\boldsymbol{w} = w_1 \cdots w_N$, the prefix probability of a substring $w_i \cdots w_k$ of $\boldsymbol{w}$, can be defined recursively as follows:*

$$p_\pi(i, k \mid \mathrm{X}) = \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} E_{\mathrm{lc}}(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X}) \tag{10}$$
$$\cdot \sum_{j=i}^{k-1} \beta(i, j \mid \mathrm{Y}) \cdot p_\pi(j{+}1, k \mid \mathrm{Z})$$

Proof. Given the PCFG is in CNF and $k > i$, in order to derive the prefix $w_i \cdots w_k$ we must first apply some rule $\mathrm{X} \to \mathrm{Y}\,\mathrm{Z}$, where the first part of the substring is then derived from Y and the remainder (and potentially more) from Z:

$$p_\pi(i, k \mid \mathrm{X}) = \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{X} \to \mathrm{Y}\,\mathrm{Z}) \cdot \left[\sum_{j=i}^{k-1} \beta(i, j \mid \mathrm{Y}) \cdot p_\pi(j{+}1, k \mid \mathrm{Z}) + p_\pi(i, k \mid \mathrm{Y}) \right] \tag{16}$$

where the last term, $p_\pi(i, k \mid \mathrm{Y})$, handles the case where the whole prefix is derived from Y alone. This term is clearly recursively defined through Eq. (16), with X replaced by Y. Defining $R(\mathrm{Y}, \mathrm{Z}) \overset{\text{def}}{=} \sum_{j=i}^{k-1} \beta(i, j \mid \mathrm{Y}) \cdot p_\pi(j{+}1, k \mid \mathrm{Z})$, we can rewrite Eq. (16) as:

$$p_\pi(i, k \mid \mathrm{X}) = \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{X} \to \mathrm{Y}\,\mathrm{Z}) \cdot R(\mathrm{Y}, \mathrm{Z}) + \sum_{\mathrm{A},\mathrm{B} \in \mathcal{N}} p(\mathrm{X} \to \mathrm{A}\,\mathrm{B}) \cdot p_\pi(i, k \mid \mathrm{A}) \tag{17}$$

After repeated substitutions ad infinitum, we get:

$$p_\pi(i, k \mid \mathrm{X}) = \sum_{\mathrm{A},\mathrm{B} \in \mathcal{N}} p(\mathrm{X} \overset{*}{\Rightarrow} \mathrm{A}\,\mathrm{B}) \cdot \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{A} \to \mathrm{Y}\,\mathrm{Z}) \cdot R(\mathrm{Y}, \mathrm{Z}) \tag{18}$$

Note that, in the last step, infinite derivations do not carry any probability mass since we assumed the PCFG to be tight and trim. Hence, the final form of the equation is:

$$p_\pi(i, k \mid \mathrm{X}) = \sum_{\mathrm{A},\mathrm{B} \in \mathcal{N}} p(\mathrm{X} \overset{*}{\Rightarrow} \mathrm{A}\,\mathrm{B}) \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{A} \to \mathrm{Y}\,\mathrm{Z}) \cdot R(\mathrm{Y}, \mathrm{Z})$$
$$= \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} E_{\mathrm{lc}}(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X}) \cdot \sum_{j=i}^{k-1} \beta(i, j \mid \mathrm{Y}) \cdot p_\pi(j{+}1, k \mid \mathrm{Z})$$

∎

B Extension of Algorithm 3 to Semirings

In the following, we give the necessary background on semirings and then show how the algorithms introduced above can be framed in terms of semirings. We start by introducing the necessary definitions and notation.

Definition 11. *A **monoid** is a 3-tuple $\langle \mathcal{A}, \circ, \mathbf{1} \rangle$ where:*

(i) $\mathcal{A}$ is a non-empty set;

(ii) $\circ$ is a binary operation which is associative: $\forall a, b, c \in \mathcal{A}, (a \circ b) \circ c = a \circ (b \circ c)$;

(iii) $\mathbf{1}$ is a left and right identity element: $\forall a \in \mathcal{A}, \mathbf{1} \circ a = a \circ \mathbf{1} = a$

(iv) $\mathcal{A}$ is closed under the operation $\circ$: $\forall a, b \in \mathcal{A}, a \circ b \in \mathcal{A}$

*A monoid is **commutative** if $\forall a, b \in \mathcal{A} : a \circ b = b \circ a$.*

Definition 12. *A **semiring** is a 5-tuple $\mathcal{W} = \langle \mathcal{A}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$, where*

*(i) $\langle \mathcal{A}, \oplus, \mathbf{0} \rangle$ is a **commutative monoid** over $\mathcal{A}$ with identity element $\mathbf{0}$ under the addition operation $\oplus$;*

*(ii) $\langle \mathcal{A}, \otimes, \mathbf{1} \rangle$ is a **monoid** over $\mathcal{A}$ with identity element $\mathbf{1}$ under the multiplication operation $\otimes$;*

*(iii) Multiplication is **distributive** over addition, that is, $\forall a, b, c \in \mathcal{A}$:*

- *$a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$;*
- *$(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$.*

*(iv) $\mathbf{0}$ is an **annihilator** for $\mathcal{A}$, that is, $\forall a \in \mathcal{A}, \mathbf{0} \otimes a = a \otimes \mathbf{0} = \mathbf{0}$.*

*A semiring is **idempotent** if $\forall a \in \mathcal{A} : a \oplus a = a$.*

Definition 13. *A semiring $\mathcal{W} = \langle \mathcal{A}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ is **complete** if it is possible to extend the addition operator $\oplus$ to infinite sums, maintaining the properties of associativity, commutativity, and distributivity from the finite case (Rozenberg and Salomaa, 1997, Chapter 9). In this case, we can define the unary operation of the **Kleene star** denoted by a superscript * as the infinite sum over its operand, that is, $\forall a \in \mathcal{A}$:*

$$a^* \overset{\text{def}}{=} \bigoplus_{i=0}^{\infty} a^i \tag{19}$$

Analogously to Eq. (13), it then follows that:

$$a^* = \bigoplus_{i=0}^{\infty} a^i = a^0 \oplus \bigoplus_{i=1}^{\infty} a^i = \mathbf{1} \oplus a \otimes \bigoplus_{i=0}^{\infty} a^i = \mathbf{1} \oplus a \otimes a^* \tag{20}$$

and, similarly:

$$a^* = a^0 \oplus \bigoplus_{i=1}^{\infty} a^i = \mathbf{1} \oplus \bigoplus_{i=a}^{\infty} a^i \otimes a = \mathbf{1} \oplus a^* \otimes a \tag{21}$$

We now discuss how complete semirings can be lifted to matrices. The definitions follow analogously to matrices over the reals.

Definition 14. *We define **semiring matrix addition** as follows. Let A and B be $d \times d$ matrices whose entries are elements from a complete semiring $\mathcal{W} = \langle \mathcal{A}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$. Then the sum ("+") of A and B is defined as:*

$$(A + B)_{ij} \overset{\text{def}}{=} A_{ij} \oplus B_{ij} \qquad i, j \in 1, \dots, d \tag{22}$$

Definition 15. *We define **semiring matrix multiplication** as follows. Let A and B be $d \times d$ matrices whose entries are elements from a complete semiring $\mathcal{W} = \langle \mathcal{A}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$. Then the product ("$\cdot$") of A and B is defined as:*

$$(A \cdot B)_{ij} \stackrel{\text{def}}{=} \bigoplus_{k=1}^{d} A_{ik} \otimes B_{kj} \qquad i, j \in 1, \ldots, d \tag{23}$$

We also define the **zero matrix**, $\mathbf{O}$, over the complete semiring $\mathcal{W} = \langle \mathcal{A}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$, such that all entries are $\mathbf{0}$, and the **unit matrix I** as $(\mathbf{I})_{ij} = \mathbf{1}$ iff $i = j$ and $\mathbf{0}$ otherwise for all indices $i, j \in 0, \ldots, d$. It is then straightforward to show that matrix addition is associative and commutative while matrix multiplication is associative and distributive over matrix addition. Hence, $\langle \mathcal{W}^{d \times d}, +, \cdot, \mathbf{O}, \mathbf{I} \rangle$ is a semiring. Furthermore, by the element-wise definition of its addition operation, it is also complete.

We now consider a semiring-weighted CFG $\mathcal{G} = \langle \mathcal{N}, \Sigma, S, \mathcal{R}, p, \mathcal{W} \rangle$, where $\mathcal{N}, \Sigma, S, \mathcal{R}$ are defined as before, except the (locally normalized) weighting function p is now semiring-valued:

$$p : \mathcal{R} \to \mathcal{W} \text{ such that } \bigoplus_{X \to \boldsymbol{\alpha} \in \mathcal{R}} p(X \to \boldsymbol{\alpha}) = \mathbf{1}$$

As before, we define the matrix P as the square matrix of dimension $|\mathcal{N}|$ whose rows and columns are indexed by the non-terminals $\mathcal{N}$ in some fixed order so that the entry P_{ij} corresponds to $p(X_i \to X_j \bullet) = \bigoplus_{Y \in \mathcal{N}} p(X_i \to X_j Y)$. We can then calculate the probability of getting X_j from X_i at the leftmost non-terminal after exactly k derivation steps as $(P^k)_{ij} = \left(\bigotimes_{i=0}^{k} P \right)_{ij}$. Note that this holds because the production rule weights are locally normalized, meaning that we only need to consider the left-most rule applications instead of having to explicitly calculate the full treesum.

Finally, to get the left-corner expectations, we then need to calculate the Kleene closure over the matrix P,[7] that is, we want to find $P^* = \bigoplus_{k=0}^{\infty} P^k$. To compute the Kleene closure over the transition matrix we can use an efficient algorithm by Lehmann (1977) which is a generalization of the well-known shortest-path algorithm usually attributed to Floyd (1962) and Warshall (1962), but introduced previously by Roy (1959). The algorithm works under the condition that the Kleene closure of all individual matrix entries from semiring $\mathcal{W}$ exists, which is true for our case since we assumed $\mathcal{W}$ to be complete. The algorithm is shown in Algorithm 4.

Algorithm 4 Lehmann's algorithm for computing the Kleene closure over a transition matrix

---

1: **def** Lehmann(M):
2:    $d \leftarrow \dim(M)$ *▷ M is a $d \times d$ matrix over a complete semiring*
3:    $M^{(0)} \leftarrow M$
4:    **for** $j = 1, \ldots, d$:
5:      **for** $i = 1, \ldots, d$:
6:        **for** $k = 1, \ldots, d$:
7:          $M_{ik}^{(j)} \leftarrow M_{ik}^{(j-1)} \oplus M_{ij}^{(j-1)} \otimes \left(M_{jj}^{(j-1)} \right)^* \otimes M_{jk}^{(j-1)}$
8:    **return** $\mathbf{I} \oplus M^{(d)}$

With this, we can now generalize our prefix probability algorithm to semirings, as shown in Algorithm 5.

Proposition 4. *The semiring-weighted version of our algorithm runs in $\mathcal{O}(N^2 |\mathcal{N}|^3 + N^3 |\mathcal{N}|^2)$.*

Proof. Lehmann's algorithm, as presented in Algorithm 4, has three nested for loops of d iterations each, where d is the dimension of the input matrix. In our case, d is the number of non-terminals, $|\mathcal{N}|$. Assuming

---

[7]Note that the Kleene closure exists since matrices with elements from a complete semiring are complete.

Algorithm 5 Faster prefix probability algorithm over semirings

---

1: **def** FastSemiringJL($\boldsymbol{w} = w_1 \cdots w_N, \mathcal{G}$):
2:     $\beta \leftarrow \text{CKY}(\boldsymbol{w})$ *▷ Precompute β with Algorithm 1*
3:     $P^* \leftarrow \text{Lehmann}(P)$ *▷ Precompute P^* with Algorithm 4*
4:     **for** $X_i, X_j \in \mathcal{N}$:   *▷ Precompute $E_{\text{lc}}(X_j \mid X_i)$*
5:         $E_{\text{lc}}(X_j \mid X_i) \leftarrow (P^*)_{ij}$
6:     **for** $i, j = 1, \ldots, N$:
7:         **for** $X, Z \in \mathcal{N}$:   *▷ Precompute γ by Eq. (15e)*
8:             $\gamma_{ij}(X, Z) \leftarrow \bigoplus_{Y \in \mathcal{N}} p(X \rightarrow YZ) \otimes \beta(i, j \mid Y)$
9:         **for** $X, Z \in \mathcal{N}$:   *▷ Precompute δ by Eq. (15f)*
10:             $\delta_{ij}(X, Z) \leftarrow \bigoplus_{Y \in \mathcal{N}} E_{\text{lc}}(Y \mid X) \otimes \gamma_{ij}(Y, Z)$
11:     **for** $k \in 1, \ldots, N$:
12:         **for** $X \in \mathcal{N}$:   *▷ Base case*
13:             $p_\pi(k, k \mid X) \leftarrow \bigoplus_{Y \in \mathcal{N}} E_{\text{lc}}(Y \mid X) \otimes p(Y \rightarrow w_k)$
14:     **for** $\ell \in 2 \ldots N$:
15:         **for** $i \in 1 \ldots N - \ell + 1$:
16:             $k \leftarrow i + \ell - 1$
17:             **for** $X, Z \in \mathcal{N}$:   *▷ Recursively compute p_π*
18:                 $p_\pi(i, k \mid X) \leftarrow p_\pi(i, k \mid X) \oplus \bigoplus_{j=i}^{k-1} \delta_{ij}(X, Z) \otimes p_\pi(j+1, k \mid Z)$
19:     **return** p_π

---

the Kleene closure of elements in $\mathcal{W}$ can be evaluated in $\mathcal{O}(1)$, this means that computing the left corner expectations in lines 3-5 of Algorithm 5 takes $\mathcal{O}(|\mathcal{N}|^3)$, as before. Hence, the complexity of the overall algorithm remains unchanged, that is, we can compute the prefix probabilities under a semiring-weighted, locally normalized CFG $\mathcal{G}$ in $\mathcal{O}(N^2 |\mathcal{N}|^3 + N^3 |\mathcal{N}|^2)$.   ∎

A For every submission:

☑ A1. Did you describe the limitations of your work?
7

☒ A2. Did you discuss any potential risks of your work?
As this is a theoretical result about a runtime improvement of an algorithm, we were unable to identify any risks from this work.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☒ Did you run computational experiments?

Left blank.

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
No response.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☐ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
No response.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
No response.

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
No response.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.