A Fast Algorithm for Computing Prefix Probabilities

Franz Nowak **Ryan Cotterell**
{franz.nowak, ryan.cotterell}@inf.ethz.ch
ETH zürich

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, whose original runtime is $\mathcal{O}(N^3|\mathcal{N}|^3 + |\mathcal{N}|^4)$, 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 $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \mathcal{N}$, we have a production rule $\mathrm{X} \rightarrow \mathrm{Y}\ \mathrm{Z} \in \mathcal{R}$.

[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 subcubic time in the input length using fast matrix multiplication (Valiant, 1975; Benedí and Sánchez, 2007). However, they are of limited practical use (Lee, 1997).

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 2: Short Papers, pages 57–67

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:*

- *A finite set of non-terminal symbols $\mathcal{N}$;*
- *A finite set of terminal symbols Σ, $\Sigma \cap \mathcal{N} = \emptyset$;*
- *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 $p \colon \mathcal{R} \to [0, 1]$ 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:*

$$\mathrm{X} \to \mathrm{Y}\,\mathrm{Z} \tag{1}$$

$$\mathrm{X} \to a \tag{2}$$

$$\mathrm{S} \to \varepsilon \tag{3}$$

where $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \mathcal{N}$ such that $\mathrm{Y}, \mathrm{Z} \neq \mathrm{S}$, $a \in \Sigma$, and ε is the empty string.[6]

Definition 3. *A **derivation step** $\boldsymbol{\alpha} \Rightarrow \boldsymbol{\beta}$ is an application of the binary relation $\Rightarrow \colon (\mathcal{N} \cup \Sigma)^* \times (\mathcal{N} \cup \Sigma)^*$, 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 $\boldsymbol{\beta}$. The probability of a derivation step is the probability of the applied rule: $p(\boldsymbol{\alpha}\mathrm{X}\boldsymbol{\gamma} \Rightarrow \boldsymbol{\alpha}\boldsymbol{\beta}\boldsymbol{\gamma}) \stackrel{\text{def}}{=} p(\mathrm{X} \to \boldsymbol{\beta})$.*

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 define $p(\boldsymbol{\alpha} \stackrel{*}{\Rightarrow} \boldsymbol{\beta})$ as the sum of probabilities of all subsequences from $\boldsymbol{\alpha}$ to $\boldsymbol{\beta}$, where each subsequence probability is the product of the individual derivation step probabilities defined in Def. 3.*

[6] Note that any PCFG can be converted to an equivalent PCFG in CNF (Smith and Johnson, 2007).

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:*

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

Definition 6. *$\mathcal{T}_{\mathrm{X}}(w_i \cdots w_k)$ is 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 string $\boldsymbol{w} = w_1 \cdots w_N \in \Sigma^*$, and a non-terminal $\mathrm{X} \in \mathcal{N}$, the **inside probability** of X between indices i and k (where $0 \le i \le k \le N$) is defined as:*

$$\beta(i, \mathrm{X}, k) \stackrel{\text{def}}{=} p(\mathrm{X} \stackrel{*}{\Rightarrow} w_{i+1} \cdots w_k) \tag{5}$$

$$= \sum_{\boldsymbol{\tau} \in \mathcal{T}_{\mathrm{X}}(w_{i+1} \cdots w_k)} p(\boldsymbol{\tau}) \tag{6}$$

That is, the sum of the probabilities 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 string $\boldsymbol{w} = w_1 \cdots w_N \in \Sigma^*$, and a non-terminal $\mathrm{X} \in \mathcal{N}$, we define the **prefix probability** π, i.e., the probability of $\boldsymbol{w}$ being a prefix under $\mathcal{G}$, as:*

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

In words, 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 $\boldsymbol{w} = w_{i+1} \cdots w_k$ from X as $\pi(i, \mathrm{X}, k)$.

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 $\xi(\mathrm{Y} \mid \mathrm{X})$ and $\xi(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X})$ are defined as:*

$$\xi(\mathrm{Y} \mid \mathrm{X}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{u} \in \Sigma^*} p(\mathrm{X} \stackrel{*}{\Rightarrow} \mathrm{Y}\boldsymbol{u}) \tag{8}$$

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

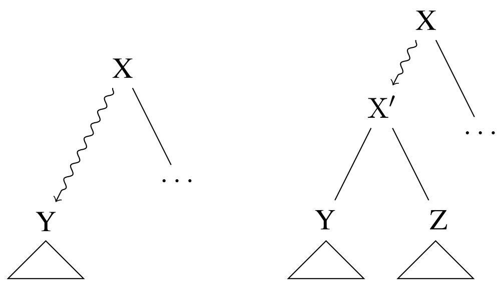
Algorithm 1 CKY

def CKY($\boldsymbol{w} = w_1 \cdots w_N$):
  ▷ *Initialize inside probabilities*
  $\beta(\cdot,\cdot,\cdot) \leftarrow 0$
  ▷ *Handle special rule,* $\mathrm{S} \rightarrow \varepsilon$
  $\beta(0, \mathrm{S}, 0) \leftarrow p(\mathrm{S} \rightarrow \varepsilon)$
  for $k \in 0, \ldots, N-1$:
    for $\mathrm{X} \rightarrow w_{k+1} \in \mathcal{R}$:
      ▷ *Handle single word tokens*
      $\beta(k, \mathrm{X}, k{+}1)$ += $p(\mathrm{X} \rightarrow w_{k+1})$
  ▷ *ℓ is the span size*
  for $\ell \in 2, \ldots, N$:
    ▷ *i marks the beginning of the span*
    for $i \in 0, \ldots, N-\ell$:
      ▷ *k marks the end of the span*
      $k \leftarrow i + \ell$
      ▷ *Recursively compute* β
      for $\mathrm{X} \rightarrow \mathrm{Y}\,\mathrm{Z} \in \mathcal{R}$:
        $\beta(i, \mathrm{X}, k)$ += $p(\mathrm{X} \rightarrow \mathrm{Y}\,\mathrm{Z}) \cdot \sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \cdot \beta(j, \mathrm{Z}, k)$
  return β

Algorithm 2 Jelinek–Lafferty

$P' \leftarrow (I - P)^{-1}$ ▷ *Precompute P^* by Eq. (14)*
for $\mathrm{X}_i, \mathrm{X}_j \in \mathcal{N}$: ▷ *Assign $\xi(\mathrm{Y} \mid X)$*
  $\xi(\mathrm{X}_j \mid \mathrm{X}_i) \leftarrow P'_{ij}$
for $\mathrm{X}' \rightarrow \mathrm{Y}\,\mathrm{Z} \in \mathcal{R}$: ▷ *Precompute $\xi(\mathrm{Y}\,\mathrm{Z} \mid X)$*
  $\xi(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X}) \leftarrow \sum_{\mathrm{X} \in \mathcal{N}} \xi(\mathrm{X}' \mid \mathrm{X}) \cdot p(\mathrm{X}' \rightarrow \mathrm{Y}\,\mathrm{Z})$
def JL($\boldsymbol{w} = w_1 \cdots w_N$):
  $\pi(\cdot,\cdot,\cdot) \leftarrow 0$ ▷ *Initialize prefix probabilities*
  for $k \in 0, \ldots, N$:
    for $\mathrm{X} \in \mathcal{N}$: ▷ *Prefix probability of ε*
      $\pi(k, \mathrm{X}, k) \leftarrow 1$
  $\beta \leftarrow \mathrm{CKY}(\boldsymbol{w})$ ▷ *Compute β with Alg. 1*
  for $k \in 0, \ldots, N-1$:
    for $\mathrm{X} \in \mathcal{N}$: ▷ *Compute base case*
      $\pi(k, \mathrm{X}, k{+}1) \leftarrow \sum_{\mathrm{Y} \in \mathcal{N}} \xi(\mathrm{Y} | \mathrm{X}) \cdot p(\mathrm{Y} \rightarrow w_{k+1})$
  for $\ell \in 2 \ldots N$:
    for $i \in 0 \ldots N - \ell$:
      $k \leftarrow i + \ell$
      for $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \mathcal{N}$: ▷ *Recursively compute π*
        $\pi(i, \mathrm{X}, k)$ += $\xi(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X}) \cdot \sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \cdot \pi(j, \mathrm{Z}, k)$
  return π

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



(a) $\xi(\mathrm{Y} \mid \mathrm{X})$ (b) $\xi(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X})$

Figure 2: Visualization of left-corner expectations

The left-corner expectation $\xi(\mathrm{Y} \mid \mathrm{X})$ 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, $\xi(\mathrm{Y}\,\mathrm{Z} \mid \mathrm{X})$ 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 $\boldsymbol{w} = w_1 \cdots w_N$, the prefix probability of a substring $w_{i+1} \cdots w_k$ of $\boldsymbol{w}$ being derived from* X *can be defined recursively as follows:*

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

Base case (for all $k \in 0 \ldots N-1$ and all $\mathrm{X} \in \mathcal{N}$):

$$\pi(k, \mathrm{X}, k{+}1) = \sum_{\mathrm{Y} \in \mathcal{N}} \xi(\mathrm{Y}|\mathrm{X}) \cdot p(\mathrm{Y} \rightarrow w_{k+1}) \tag{11}$$

Proof. A proof of Lem. 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 Alg. 1. Next, the left-corner expectations ξ as defined by Eq. (8) can be computed efficiently as follows. Let P denote the square matrix of dimension $|\mathcal{N}|$, with rows and columns indexed by the non-terminals $\mathcal{N}$ (in some fixed order), where the entry at the i^{th} row and the j^{th} column corresponds to $p(\mathrm{X}_i \rightarrow \mathrm{X}_j\ \_)$, i.e., the probability of deriving X_j on the left corner from X_i in one step (we use _ as a wildcard):

$$p(\mathrm{X}_i \rightarrow \mathrm{X}_j \,\_) \stackrel{\text{def}}{=} \sum_{\mathrm{Y} \in \mathcal{N}} p(\mathrm{X}_i \rightarrow \mathrm{X}_j \ \mathrm{Y}) \tag{12}$$

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:

$$p(\mathrm{X}_i \xrightarrow{k} \mathrm{X}_j \,\_) = (P^k)_{ij} \tag{13}$$

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:[7]

$$P^* \stackrel{\text{def}}{=} I + P + P^2 + P^3 + \cdots = \sum_{n=0}^{\infty} P^n \tag{14}$$
$$= I + P \sum_{n=0}^{\infty} P^n = I + PP^* = (I - P)^{-1}$$

Note that the entry at the i^{th} row and j^{th} column of P^* is exactly the left-corner expectation $\xi(\mathrm{X}_j \mid \mathrm{X}_i)$. Finally, we can compute the left-corner expectations $\xi(\mathrm{Y\,Z} \mid \mathrm{X})$ using Eq. (9):

$$\xi(\mathrm{Y\,Z} \mid \mathrm{X}) \stackrel{\text{def}}{=} \sum_{\mathrm{X}' \in \mathcal{N}} \xi(\mathrm{X}' \mid \mathrm{X}) \cdot p(\mathrm{X}' \rightarrow \mathrm{Y\,Z})$$

Lastly, for completeness, we also compute the prefix probability of the empty string, ε. For probabilistic PCFGs, this probability is simply 1 because ε is a prefix of any string:[8]

$$\pi(k, \mathrm{X}, k) \stackrel{\text{def}}{=} \pi(\varepsilon \mid \mathrm{X}) = 1 \tag{15}$$

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

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

Proof. Clearly, the computationally critical part is in lines 11–18, where we iterate over all indices of $\boldsymbol{w}$ 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 Σ taken as constant, the number of rules, $|\mathcal{R}|$, is $\mathcal{O}(|\mathcal{N}|^3)$, making the overall complexity of CKY $\mathcal{O}(N^3|\mathcal{N}|^3)$. ■

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

[8]For a generalization of the algorithm that does not require locally normalized rule weights, see App. B.

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

Proof. First, we precompute any values that are independent of the input. In lines 1–3, we precompute all the left-corner expectations $\xi(\mathrm{Y} \mid \mathrm{X})$ using Eq. (14), which has the complexity of inverting the matrix P, i.e., $\mathcal{O}(|\mathcal{N}|^3)$, and move the values into a map of left-corner expectations in $\mathcal{O}(|\mathcal{N}|^2)$ (this is just for readability). In lines 4–5, we then use Eq. (9) to compute $\xi(\mathrm{Y\,Z} \mid \mathrm{X})$, iterating once over all non-terminals X for each rule, which takes $\mathcal{O}(|\mathcal{R}||\mathcal{N}|)$, that is, $\mathcal{O}(|\mathcal{N}|^4)$. After initializing the probabilities in lines 7–10 in $\mathcal{O}(N^2|\mathcal{N}|)$, we begin by pre-computing all the inside probabilities β in line 11 of Alg. 2, which takes $\mathcal{O}(N^3|\mathcal{N}|^3)$ by Prop. 1. Computing $\pi(k, \mathrm{X}, k+1)$ for all $\mathrm{X} \in \mathcal{N}$ by Eq. (11) in lines 12–14 takes $\mathcal{O}(N|\mathcal{N}|^2)$ as we iterate over all positions $k \in N$ and over all $\mathrm{Y} \in \mathcal{N}$ for each $\mathrm{X} \in \mathcal{N}$. And finally, computing the π chart in lines 15–19 takes $\mathcal{O}(N^3|\mathcal{N}|^3)$ since we iterate over all $\ell, i, j \leq N$ and $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \mathcal{N}$. 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 $\pi(i, \mathrm{X}, k)$, which allows additional memoization. Starting with the definition of the prefix probability in Eq. (16a), we first expand $\xi(\mathrm{Y\,Z} \mid \mathrm{X})$ by Eq. (9), as seen in Eq. (16b). Then, we factor out all terms that depend on the left-corner non-terminal Y in Eq. (16c), which we store in a chart γ, see Eq. (16e). We then do the same for all terms depending on X', factoring them out in Eq. (16d) and storing them in another chart δ, see Eq. (16f).

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

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

Proof. As before, Alg. 3 starts by precomputing and assigning the left-corner expectations in lines 1–3, which takes $\mathcal{O}(|\mathcal{N}|^3)$ and $\mathcal{O}(|\mathcal{N}|^2)$, respectively. We then initialize the prefix probabilities and compute the inside probabilities in lines 5–8, taking $\mathcal{O}(N^2|\mathcal{N}|)$. As Eisner and Blatz (2007) show, one can compute β (line 9) in $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$, thus improving the runtime of Alg. 1 for dense grammars. Pre-computing γ and δ in lines 10–14

$$\pi(i, \mathrm{X}, k) = \sum_{\mathrm{Y},\mathrm{Z}\in\mathcal{N}} \xi(\mathrm{Y}, \mathrm{Z} \mid \mathrm{X}) \cdot \sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \cdot \pi(j, \mathrm{Z}, k) \tag{16a}$$

$$= \sum_{\mathrm{Y},\mathrm{Z}\in\mathcal{N}} \sum_{\mathrm{X}'\in\mathcal{N}} \xi(\mathrm{X}' \mid \mathrm{X}) \cdot p(\mathrm{X}' \to \mathrm{Y}\,\mathrm{Z}) \cdot \sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \cdot \pi(j, \mathrm{Z}, k) \tag{16b}$$

$$= \sum_{\mathrm{X}',\mathrm{Z}\in\mathcal{N}} \xi(\mathrm{X}' \mid \mathrm{X}) \cdot \sum_{j=i+1}^{k-1} \gamma_{ij}(\mathrm{X}', \mathrm{Z}) \cdot \pi(j, \mathrm{Z}, k) \tag{16c}$$

$$= \sum_{\mathrm{Z}\in\mathcal{N}} \sum_{j=i+1}^{k-1} \delta_{ij}(\mathrm{X}, \mathrm{Z}) \cdot \pi(j, \mathrm{Z}, k) \tag{16d}$$

$$\text{where } \gamma_{ij}(\mathrm{X}', \mathrm{Z}) \stackrel{\text{def}}{=} \sum_{\mathrm{Y}\in\mathcal{N}} p(\mathrm{X}' \to \mathrm{Y}\,\mathrm{Z}) \cdot \beta(i, \mathrm{Y}, j) \tag{16e}$$

$$\text{and } \delta_{ij}(\mathrm{X}, \mathrm{Z}) \stackrel{\text{def}}{=} \sum_{\mathrm{X}'\in\mathcal{N}} \xi(\mathrm{X}' \mid \mathrm{X}) \cdot \gamma_{ij}(\mathrm{X}', \mathrm{Z}) \tag{16f}$$

Algorithm 3 Faster prefix probability algorithm

```
1:  P' ← (I − P)^{−1}                         ▷ Precompute P* with Eq. (14)
2:  for X_i, X_j ∈ 𝒩 :                          ▷ Assign ξ(Y | X)
3:      ξ(X_j | X_i) ← P'_{ij}
4:  def FastJL(w = w_1 ··· w_N):
5:      π(·,·,·) ← 0                           ▷ Initialize prefix probabilities
6:      for k ∈ 0, . . . , N :
7:          for X ∈ 𝒩 :                         ▷ Prefix probability of ε
8:              π(k, X, k) ← 1
9:      β ← CKY(w)                             ▷ Compute β with Alg. 1
10:     for i, j = 0, . . . , N :
11:         for X, Z ∈ 𝒩 :                      ▷ Compute γ by Eq. (16e)
12:             γ_{ij}(X,Z) ← Σ_{Y∈𝒩} p(X→YZ)·β(i, Y, j)
13:         for X, Z ∈ 𝒩 :                      ▷ Compute δ by Eq. (16f)
14:             δ_{ij}(X,Z) ← Σ_{Y∈𝒩} ξ(Y | X) · γ_{ij}(Y, Z)
15:     for k ∈ 0, . . . , N − 1 :
16:         for X ∈ 𝒩 :                         ▷ Compute base case
17:             π(k, X, k+1) ← Σ_{Y∈𝒩} ξ(Y | X)
                    ·p(Y → w_{k+1})
18:     for ℓ ∈ 2 . . . N :
19:         for i ∈ 1 . . . N − ℓ :
20:             k ← i + ℓ
21:             for X, Z ∈ 𝒩 :                  ▷ Recursively compute π
22:                 π(i, X, k) += Σ_{j=i+1}^{k−1} δ_{ij}(X, Z)·π(j, Z, k)
23:     return π
```

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 $\mathcal{N}$. Computing the base case $\pi(k, \mathrm{X}, k+1)$ for all non-terminals X and positions k in lines 15–17 takes $\mathcal{O}(N|\mathcal{N}|^2)$, as before. Finally, the loops computing π in lines 18–22 take $\mathcal{O}(N^3|\mathcal{N}|^2)$, as we are now iterating over $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$ and $\ell, i, j \leq N$. 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 semiring is closed, i.e., it has a Kleene star. 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, ξ. We can even relax the condition of local normalization by changing how left-corner expectations and the weight of the prefix ε are computed. The relevant definitions and derivation of the adapted algorithm 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.

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 precomputed 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.

Ethics Statement

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

Acknowledgements

We thank the anonymous reviewers for their helpful comments and suggestions. We also extend our gratitude to Abra Ganz, Andreas Opedal, Anej Svete, and Tim Vieira for their valuable feedback on various versions of this paper.

References

Steven C. Althoen and Renate McLaughlin. 1987. Gauss-jordan reduction: A brief history. *The American Mathematical Monthly*, 94(2):130–142.

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.

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2007. On fixed point equations over commutative semirings. In *Proceedings of the 24th Annual Conference on Theoretical Aspects of Computer Science*, page 296–307, Berlin, Heidelberg. Springer-Verlag.

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.

S. C. Kleene. 1956. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–42. Princeton University Press, Princeton.

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*, 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.

Noah A. Smith and Mark Johnson. 2007. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491.

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 Lem. 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 $w_{i+1} \cdots w_k$ of $\boldsymbol{w}$ being derived from* X *can be defined recursively as follows:*

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

Base case (for all $k \in 0 \ldots N-1$ and all $\mathrm{X} \in \mathcal{N}$):

$$\pi(k, \mathrm{X}, k{+}1) = \sum_{\mathrm{Y} \in \mathcal{N}} \xi(\mathrm{Y} \mid \mathrm{X}) \cdot p(\mathrm{Y} \to w_{k+1}) \tag{11}$$

Proof. Eq. (11): The base case, $\pi(k, \mathrm{X}, k{+}1)$, is the probability of w_{k+1} being the left-most terminal in the parse subtree under X. It is, therefore, simply the sum of probabilities of any non-terminal Y being on the left corner of the parse subtree under X multiplied by the corresponding probability of Y directly deriving w_{k+1}.

Eq. (10): Given the PCFG is in CNF and assuming $k > i + 1$, in order to derive the prefix $w_{i+1} \cdots w_k$ we must first apply some rule $\mathrm{X} \to \mathrm{Y\,Z}$, where the first part of the substring is then derived from Y and the remainder (and potentially more) from Z:

$$\pi(i, \mathrm{X}, k) = \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{X} \to \mathrm{Y\,Z}) \left[\sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \cdot \pi(j, \mathrm{Z}, k) + \pi(i, \mathrm{Y}, k) \right] \tag{17}$$

where the last term, $\pi(i, \mathrm{Y}, k)$, handles the case where the whole prefix is derived from Y alone. This term is clearly recursively defined through Eq. (17), with X replaced by Y. Defining $R(\mathrm{Y}, \mathrm{Z}) \stackrel{\text{def}}{=} \sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \pi(j, \mathrm{Z}, k)$, we can rewrite Eq. (17) as:

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

After repeated substitutions ad infinitum, we get:

$$\pi(i, \mathrm{X}, k) = \sum_{\mathrm{A},\mathrm{B} \in \mathcal{N}} p(\mathrm{X} \stackrel{*}{\Rightarrow} \mathrm{A\,B}) \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{A} \to \mathrm{Y\,Z}) \cdot R(\mathrm{Y}, \mathrm{Z}) \tag{19}$$

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:

$$\begin{aligned} \pi(i, \mathrm{X}, k) &= \sum_{\mathrm{A},\mathrm{B} \in \mathcal{N}} p(\mathrm{X} \stackrel{*}{\Rightarrow} \mathrm{A\,B}) \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} p(\mathrm{A} \to \mathrm{Y\,Z}) \cdot R(\mathrm{Y}, \mathrm{Z}) \\ &= \sum_{\mathrm{Y},\mathrm{Z} \in \mathcal{N}} \xi(\mathrm{Y\,Z} \mid \mathrm{X}) \sum_{j=i+1}^{k-1} \beta(i, \mathrm{Y}, j) \cdot \pi(j, \mathrm{Z}, k) \end{aligned} \tag{20}$$

■

B Extension of Alg. 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.

B.1 Semirings

We start by introducing the necessary definitions and notation.

Definition 10. *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 an associative binary operation: $\forall a, b, c \in \mathcal{A}, (a \circ b) \circ c = a \circ (b \circ c)$;

(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 11. *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 **commutative** if $\langle \mathcal{A}, \otimes, \mathbf{1} \rangle$ is a commutative monoid. A semiring is **idempotent** if $\forall a \in \mathcal{A} : a \oplus a = a$.*

Definition 12. *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 powers of its operand, that is, $\forall a \in \mathcal{A}$:*

$$a^* \stackrel{\text{def}}{=} \bigoplus_{i=0}^{\infty} a^i \tag{21}$$

Analogously to Eq. (14), it then follows that:

$$a^* = \bigoplus_{i=0}^{\infty} a^i = a^0 \oplus \bigoplus_{i=1}^{\infty} a^i = \mathbf{1} \oplus a \otimes \bigoplus_{i=0}^{\infty} a^i = \mathbf{1} \oplus a \otimes a^* \tag{22}$$

and, similarly:

$$a^* = a^0 \oplus \bigoplus_{i=1}^{\infty} a^i = \mathbf{1} \oplus \bigoplus_{i=0}^{\infty} a^i \otimes a = \mathbf{1} \oplus a^* \otimes a \tag{23}$$

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

Definition 13. *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:*

$$(A + B)_{ij} \stackrel{\text{def}}{=} A_{ij} \oplus B_{ij} \qquad i, j \in 1, \ldots, d \tag{24}$$

Definition 14. *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 of A and B is defined as:*

$$(AB)_{ij} \stackrel{\text{def}}{=} \bigoplus_{k=1}^{d} A_{ik} \otimes B_{kj} \qquad i, j \in 1, \ldots, d \tag{25}$$

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, the set of square matrices of dimension d over a semiring, with addition and multiplication as defined above, is itself a semiring. Furthermore, by the element-wise definition of its addition operation, it is also complete.

B.2 Semiring-weighted prefix algorithm

We now consider a semiring-weighted CFG $\mathcal{G} = \langle \mathcal{N}, \Sigma, \mathrm{S}, \mathcal{R}, p, \mathcal{W} \rangle$, where $\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{R}$ are defined as before but the weighting function $p \colon \mathcal{R} \to \mathcal{W}$ now maps rules to elements of a commutative semiring $\mathcal{W}$.[9] Note that we no longer require the rule weights to sum to one. 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 the weight of getting the non-terminal $\mathrm{X_j}$ on the left after one rule application to $\mathrm{X_i}$. Since the rule weights are no longer locally normalized, however, we need to include the treesum under the right non-terminal of each rule as an additional term:[10]

$$P_{ij} = p(\mathrm{X}_i \to \mathrm{X}_j \ \_) \stackrel{\text{def}}{=} \bigoplus_{\mathrm{Y} \in \mathcal{N}} p(\mathrm{X}_i \to \mathrm{X}_j \mathrm{Y}) \cdot \mathrm{treesum}(\mathrm{Y}) \tag{26}$$

We can then calculate the weight of getting X_j from X_i at the leftmost non-terminal after exactly k derivation steps as $(P^k)_{ij}$, where $P^k \stackrel{\text{def}}{=} \underbrace{P \cdots P}_{k \text{ times}}$. Finally, to get the left-corner expectations, we then need to calculate the Kleene closure over the matrix P,[11] that is, we want to find $P^* = \sum_{k=0}^{\infty} P^k$. 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).[12] 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 Alg. 4.

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

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

[9] We require that $\mathcal{W}$ be *commutative* because the order of rule applications does not affect string weight in a weighted CFG.

[10] For locally normalized semirings, the treesum of any non-terminal is $\mathbf{1}$. In the general case when the weights are not normalized, the treesum of a semiring-weighted WCFG can be computed through fixed point iteration in a similar way to Newton's method (Esparza et al., 2007).

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

[12] The generalization is by way of choosing the appropriate semiring for the given problem. By the same token, Lehmann's algorithm can also be seen as a generalization of Kleene's (1956) algorithm for converting finite-state automata to regular expressions and the Gauss–Jordan algorithm for computing matrix inversion (see e.g. Althoen and McLaughlin (1987)).

Lastly, since we no longer have normalized rule weights, we need to set the prefix weight of ε under any non-terminal $\mathrm{X} \in \mathcal{N}$ to the treesum under X. With this, we can now generalize our prefix weight algorithm to semirings, as shown in Alg. 5.

Algorithm 5 Faster prefix algorithm over semirings

1: $P' \leftarrow \mathrm{Lehmann}(P)$ ▷ *Precompute P^* with Alg. 4*
2: **for** $\mathrm{X}_i, \mathrm{X}_j \in \mathcal{N}$: ▷ *Assign $\xi(\mathrm{X}_j \mid \mathrm{X}_i)$*
3:     $\xi(\mathrm{X}_j \mid \mathrm{X}_i) \leftarrow {P'}_{ij}$
4: **def** FastSemiringJL($\boldsymbol{w} = w_1 \cdots w_N, \mathcal{G}$):
5:     $\pi(\cdot,\cdot,\cdot) \leftarrow \mathbf{0}$ ▷ *Initialize prefix probabilities*
6:     **for** $k \in 0, \ldots, N$:
7:         **for** $\mathrm{X} \in \mathcal{N}$: ▷ *Prefix weight of ε*
8:             $\pi(k, \mathrm{X}, k) \leftarrow \mathrm{treesum}(\mathrm{X})$
9:     $\beta \leftarrow \mathrm{CKY}(\boldsymbol{w})$ ▷ *Compute β with Alg. 1*
10:     **for** $i, j = 0, \ldots, N$:
11:         **for** $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$: ▷ *Compute γ by Eq. (16e)*
12:             $\gamma_{ij}(\mathrm{X}, \mathrm{Z}) \leftarrow \bigoplus_{\mathrm{Y} \in \mathcal{N}} p(\mathrm{X} \rightarrow \mathrm{YZ}) \otimes \beta(i, \mathrm{Y}, j)$
13:         **for** $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$: ▷ *Compute δ by Eq. (16f)*
14:             $\delta_{ij}(\mathrm{X}, \mathrm{Z}) \leftarrow \bigoplus_{\mathrm{Y} \in \mathcal{N}} \xi(\mathrm{Y} \mid \mathrm{X}) \otimes \gamma_{ij}(\mathrm{Y}, \mathrm{Z})$
15:     **for** $k \in 0, \ldots, N-1$:
16:         **for** $\mathrm{X} \in \mathcal{N}$: ▷ *Compute base case*
17:             $\pi(k, \mathrm{X}, k{+}1) \leftarrow \bigoplus_{\mathrm{Y} \in \mathcal{N}} \xi(\mathrm{Y} \mid \mathrm{X}) \otimes p(\mathrm{Y} \rightarrow w_{k+1})$
18:     **for** $\ell \in 2 \ldots N$:
19:         **for** $i \in 1 \ldots N - \ell$:
20:             $k \leftarrow i + \ell$
21:             **for** $\mathrm{X}, \mathrm{Z} \in \mathcal{N}$: ▷ *Recursively compute π*
22:                 $\pi(i, \mathrm{X}, k) \mathrel{+}= \bigoplus_{j=i+1}^{k-1} \delta_{ij}(\mathrm{X}, \mathrm{Z}) \otimes \pi(j, \mathrm{Z}, k)$
23:     **return** π

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 Alg. 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 the Kleene closure of elements in $\mathcal{W}$ can be evaluated in $\mathcal{O}(1)$, this means that computing the left corner expectations in line 1 of Alg. 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 $\mathcal{O}(N^2|\mathcal{N}|^3 + N^3|\mathcal{N}|^2)$. ■