Computation of Distances for Regular and Context-Free Probabilistic Languages ⋆

Mark-Jan Nederhof [a], Giorgio Satta [b,*]

[a] *School of Computer Science, University of St Andrews, North Haugh, St Andrews, KY16 9SX, Scotland*

[b] *Department of Information Engineering, University of Padua, via Gradenigo, 6/A, I-35131 Padova, Italy*

Abstract

Several mathematical distances between probabilistic languages have been investigated in the literature, motivated by applications in language modeling, computational biology, syntactic pattern matching and machine learning. In most cases, only pairs of probabilistic regular languages were considered. In this paper we extend previous results to pairs of languages generated by a probabilistic context-free grammar and a probabilistic finite automaton.

Key words: Probabilistic Context-Free Languages, Probabilistic Finite Automata, Probabilistic Language Distances, Language Entropy, Kullback-Leibler Divergence

1 Introduction

In the last decade, widespread interest in probabilistic formal languages has arisen in the areas of natural language processing [8,27,23], speech recognition [21], computational biology [14] and syntactic pattern matching [13]. In all of these fields, formal languages are used to model the domain of interest, and probabilities are exploited to direct the search in the relevant space.

⋆ A preliminary version of some of the results presented in Section 3 appears in [30], and a preliminary version of some of the results presented in Section 6 appears in [31].

* Corresponding author. Tel.: +39 049 827 7948, fax: +39 049 827 7799.
URLs: `http://www.cs.st-andrews.ac.uk/~mjn` (Mark-Jan Nederhof), `http://www.dei.unipd.it/~satta` (Giorgio Satta).

For instance, in speech recognition, probabilistic regular languages are used to rank different interpretations of a noisy acoustic signal, and in natural language processing, probabilistic context-free grammars are used to rank different parses for an ambiguous input sentence. In this way more probable interpretations of a sentence can be discriminated from less probable ones.

The success of this approach relies on the accuracy of the language model expressed by the probabilistic grammar or automaton, i.e., whether the probabilities assigned to derivations accurately reflect the 'true' probabilities in the domain at hand. To this end, probabilities of the grammar or automaton are usually estimated on the basis of a large corpus, i.e., a multiset of examples annotated with various kinds of information [27]. A probability distribution can be associated with such a corpus. This is called the empirical distribution and describes the observed frequency of the phenomena of interest. Alternatively, the parameters of a model can be estimated from a more expressive language model. An example is the approximation of a source probabilistic context-free language by means of a target probabilistic finite automaton [29]. The main goal of the estimation process is then to optimize some objective function that expresses the 'similarity' between the source distribution and the target distribution.

Several similarity measures between probability distributions have been proposed and investigated in the literature, ranging from mathematical functions that satisfy all of the properties of distances, to pseudo-distances, defined on the basis of information-theoretic concepts. Examples of pseudo-distances are cross-entropy and Kullback-Leibler divergence, which do not satisfy the properties of distances but are still very useful in practice. These measures can be computed and compared to some threshold, in order to establish a criterion for the convergence of estimation algorithms. These measures are also used for the benefit of language modeling, where different target models are to be compared against a source distribution, in order to select the model that best fits the data. Even when such measures cannot be exactly computed, or when their computation requires an exponential amount of time, they may still be applied in practice by relying on efficient approximation algorithms.

Most of the algorithms reported in the literature for the computation of distances between probabilistic languages are stated for pairs of languages generated by probabilistic finite automata. An overview of these results can be found in [39]. More recent work has been presented in [7,**?**,**?**]. The main contribution of the present paper is the extension of some of these results to pairs of languages generated by a probabilistic context-free grammar and a probabilistic finite automaton, under various conditions. In deriving our algorithms for the computation of the distances considered here, we exploit a construction that was originally presented in [4] for the purpose of the computation of the intersection of a context-free language and a regular language. We extend this

construction to the probabilistic case.

The remainder of this article is organized as follows. In Section 2 we briefly recall the definitions of probabilistic and weighted context-free grammars and finite automata. In Section 3 we introduce a probabilistic extension of a construction computing the intersection of a context-free and a regular language. In Section 4 we discuss several methods for the computation of the partition function, which plays an important rôle in our algorithms. In Section 5 we introduce some true distance measures, and provide algorithms for their exact or approximate computation. In Section 6 we also provide algorithms for the computation of some pseudo-distances that are commonly used in language modeling and in machine learning. We close with some discussion in Section 7.

2 Preliminaries

In this section we briefly recall the definitions of probabilistic and weighted context-free grammars and probabilistic and weighted finite automata. All of the results we provide in this paper are stated for probabilistic formalisms, but in some proofs we will need more general weighted formalisms as intermediate models. Many of the definitions related to weighted and probabilistic context-free grammars are based on [34,5] and those related to weighted and probabilistic finite automata are based on [33,37]. Below, the symbol ε denotes the empty string, and string concatenation is represented by operator '$\cdot$' or by empty space. The length of a string x is written $|x|$.

A *context-free grammar* (CFG) is a tuple $G = (\Sigma, N, S, R)$, where Σ and N are two finite disjoint sets of *terminals* and *nonterminals*, respectively, $S \in N$ is the *start symbol*, and R is a finite set of *rules*, each of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\Sigma \cup N)^*$. In what follows, symbol a ranges over the set Σ, symbols w, v range over the set Σ^*, symbols A, B range over the set N, symbol X ranges over the set $\Sigma \cup N$, symbols α, β, γ range over the set $(\Sigma \cup N)^*$, symbol π ranges over the set R, and symbols d, e range over the set R^*. With slight abuse of notation, we treat a rule $\pi = (A \to \alpha) \in R$ as an *atomic* symbol when it occurs within a string $d\pi e \in R^*$.

For a fixed CFG G, we define the *left-most derive* relation $\Rightarrow_G$ on triples consisting of two strings $\alpha, \beta \in (\Sigma \cup N)^*$ and a rule $\pi \in R$. We write $\alpha \overset{\pi}{\Rightarrow}_G \beta$ if and only if α is of the form $wA\delta$ and β is of the form $w\gamma\delta$, for some $w \in \Sigma^*$ and $\delta \in (\Sigma \cup N)^*$, and $\pi = (A \to \gamma)$. A *left-most derivation* (in G) is a string $d = \pi_1 \cdots \pi_m$, $m \geq 0$, such that $\alpha_0 \overset{\pi_1}{\Rightarrow}_G \alpha_1 \overset{\pi_2}{\Rightarrow}_G \cdots \overset{\pi_m}{\Rightarrow}_G \alpha_m$, for some $\alpha_0, \ldots, \alpha_m \in (\Sigma \cup N)^*$; $d = \varepsilon$ is always a left-most derivation. In the remainder of this paper, we will let the term 'derivation' refer to 'left-most derivation', unless specified otherwise. We omit the subscript from the notation $\Rightarrow_G$ when

CFG G is understood.

If $\alpha_0 \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} \alpha_m$ for some $\alpha_0, \ldots, \alpha_m \in (\Sigma \cup N)^*$, then we say that $d = \pi_1 \cdots \pi_m$ *derives* α_m from α_0 and we write $\alpha_0 \stackrel{d}{\Rightarrow} \alpha_m$; $d = \varepsilon$ derives any $\alpha_0 \in (\Sigma \cup N)^*$ from itself. Let $w \in \Sigma^*$. We define

$$\mathcal{D}(w, G) = \{d \mid S \stackrel{d}{\Rightarrow} w\}, \tag{1}$$

that is, $\mathcal{D}(w, G)$ is the set of all derivations (in G) of w from the start symbol S. We also let $\mathcal{D}(G) = \cup_w \mathcal{D}(w, G)$. The language generated by G, written $L(G)$, is the set of all strings $w \in \Sigma^*$ such that $|\mathcal{D}(w, G)| > 0$.

We say that G is *ambiguous* if $|\mathcal{D}(w, G)| > 1$ for at least one string w. We say that G is *linear* if each of its rules has at most one nonterminal in the right-hand side.

A CFG is said to be *reduced* if for each nonterminal A there are $d_1, d_2 \in R^*$, $w_1, w_2 \in \Sigma^*$ and $\beta \in (\Sigma \cup N)^*$ such that $S \stackrel{d_1}{\Rightarrow} w_1 A \beta$ and $w_1 A \beta \stackrel{d_2}{\Rightarrow} w_1 w_2$. In words, if a CFG is reduced, then for each nonterminal A there is at least one derivation $d_1 d_2$ that derives a string $w_1 w_2$ from S and that includes some rule with left-hand side A. Reduction of a CFG consists of removing the offending nonterminals (if any) and the rules in which they occur, and can be carried out in linear time in the size of the grammar [35, Theorem 4.17] (see below for the definition of the size of a CFG).

A *probabilistic* CFG (PCFG) is a tuple $\mathcal{G} = (\Sigma, N, S, R, p_G)$, where Σ, N, S, R are defined as for a CFG and p_G is a function from rules in R to real numbers in the interval $(0, 1]$. Function p_G is extended to instances of the derive relation as follows. For a rule $\pi \in R$ and a derivation $d \in R^*$, we write $f(\pi, d)$ to denote the number of occurrences of π within d. Let $\alpha, \beta \in (\Sigma \cup N)^*$. We define

$$p_G(\alpha \stackrel{d}{\Rightarrow} \beta) = \begin{cases} \prod_{\pi \in R} p_G(\pi)^{f(\pi, d)}, & \text{if } \alpha \stackrel{d}{\Rightarrow} \beta; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The probability of a string $w \in \Sigma^*$ is defined as

$$p_G(w) = \sum_d p_G(S \stackrel{d}{\Rightarrow} w). \tag{3}$$

The *underlying* CFG of a PCFG $\mathcal{G} = (\Sigma, N, S, R, p_G)$ is $G = (\Sigma, N, S, R)$. We let a PCFG $\mathcal{G}$ inherit the properties we defined for CFG G. For example,

$L(\mathcal{G}) = L(G)$, and a PCFG is linear if and only if its underlying CFG is linear. We define $p_G(L) = \sum_{w \in L} p_G(w)$ for each language $L \subseteq \Sigma^*$. Note that $p_G(\Sigma^*) = p_G(L(\mathcal{G}))$.

A PCFG is said to be *proper* if, for all $A \in N$,

$$\sum_{\alpha} p_G(A \rightarrow \alpha) = 1, \tag{4}$$

that is, if the probabilities of all rules with left-hand side A sum to 1. A PCFG is said to be *consistent* if $p_G(L(\mathcal{G})) = 1$. Consistency implies that function p_G defines a probability distribution over the set of terminal strings in Σ^*. There is a practical sufficient condition for consistency that is decidable [5].

A *weighted* CFG (WCFG) is a tuple $\mathcal{G} = (\Sigma, N, S, R, p_G)$ defined as for a PCFG, with the only difference that p_G is now a function from rules in R to positive real numbers. A WCFG is said to be *convergent* if $p_G(L(\mathcal{G})) < \infty$.

Throughout this paper, we make the assumption that the weights of the rules of our WCFGs are all rational numbers. We also assume a reasonable (finite) representation of rational numbers satisfying the following condition. Let $\|r\|$ be the number of bits of rational number r under the chosen representation. Then for any two rational numbers r_1 and r_2, we have $\|r_1 \cdot r_2\| \leq \|r_1\| + \|r_2\|$. The size of a rule of the form $A \rightarrow \alpha$ in a PCFG or WCFG $\mathcal{G}$ is defined as $\|A \rightarrow \alpha\| = |A\alpha| + \|p_G(A \rightarrow \alpha)\|$. The size of $\mathcal{G}$ is defined as $\|\mathcal{G}\| = \sum_{A \rightarrow \alpha} \|A \rightarrow \alpha\|$.

A *finite automaton* (FA) is a tuple $M = (\Sigma, Q, I, F, T)$, where Σ and Q are two finite disjoint sets of *terminals* and *states*, respectively, $I \subseteq Q$ is the set of *initial* states, $F \subseteq Q$ is the set of *final* states, and T is a finite set of *transitions*, each of the form $r \overset{a}{\mapsto} s$, where $r, s \in Q$ and $a \in \Sigma$. In what follows, symbols q, r, s range over the set Q, symbol τ ranges over the set T, and symbol c ranges over the set T^*. Again, with slight abuse of notation, we treat a transition $\tau = (r \overset{a}{\mapsto} s) \in T$ as an atomic symbol when it occurs within a string in T^*.

For a fixed FA M, we define a *configuration* to be an element of $Q \times \Sigma^*$. We also define a relation $\vdash_M$ on triples consisting of two configurations and a transition $\tau \in T$. We write $(r, w) \overset{\tau}{\vdash}_M (s, w')$ if and only if w is of the form aw', for some $a \in \Sigma$, and $\tau = (r \overset{a}{\mapsto} s)$. A *computation* (in M) is a string $c = \tau_1 \cdots \tau_m$, $m \geq 0$, such that $(r_0, w_0) \overset{\tau_1}{\vdash}_M (r_1, w_1) \overset{\tau_2}{\vdash}_M \cdots \overset{\tau_m}{\vdash}_M (r_m, w_m)$, for some $(r_0, w_0), \ldots, (r_m, w_m) \in Q \times \Sigma^*$; $c = \varepsilon$ is always a computation. We omit the subscript from the notation $\vdash_M$ when FA M is understood.

If $(r_0, w_0) \overset{\tau_1}{\vdash} \cdots \overset{\tau_m}{\vdash} (r_m, w_m)$ for some $(r_0, w_0), \ldots, (r_m, w_m) \in Q \times \Sigma^*$ and

$c = \tau_1 \cdots \tau_m \in T^*$, then we write $(r_0, w_0) \overset{c}{\vdash} (r_m, w_m)$. Let $w \in \Sigma^*$ and let $r, s \in Q$. We define

$$\mathcal{C}_{r,s}(w, M) = \{c \mid (r, w) \overset{c}{\vdash} (s, \varepsilon)\}, \tag{5}$$

that is, $\mathcal{C}_{r,s}(w, M)$ is the set of all computations (in M) scanning w, starting from r and ending in s. We also define

$$\mathcal{C}(w, M) = \cup_{r \in I, s \in F} \mathcal{C}_{r,s}(w, M), \tag{6}$$

and let $\mathcal{C}(M) = \cup_w \mathcal{C}(w, M)$. The language recognized by M, written $L(M)$, is the set of all strings $w \in \Sigma^*$ such that $|\mathcal{C}(w, M)| > 0$.

A FA M is *deterministic* if $|I| = 1$ and, for each $r \in Q$ and $a \in \Sigma$, there is at most one transition in T of the form $r \overset{a}{\mapsto} s$. It is easy to see that if M is deterministic, then $|\mathcal{C}(w, M)| \leq 1$ for every string $w \in \Sigma^*$.

A FA is said to be *reduced* if, for each state q, there are $r \in I, s \in F, c_1, c_2 \in T^*$ and $w_1, w_2 \in \Sigma^*$ such that $(r, w_1 w_2) \overset{c_1}{\vdash} (q, w_2)$ and $(q, w_2) \overset{c_2}{\vdash} (s, \varepsilon)$. In words, if a FA is reduced, then for each state q there is at least one computation $c_1 c_2$ going through q that recognizes a string $w_1 w_2$. A FA can easily be turned into one that is reduced by omitting the offending states and the transitions that use them. This procedure can be carried out in linear time in the size of the FA, using standard graph algorithms for the vertex-to-vertex reachability problem [12].

A *probabilistic* FA (PFA) is a tuple $\mathcal{M} = (\Sigma, Q, I, F, T, p_I, p_F, p_M)$, where Σ, Q, I, F, T are defined as for a FA, and p_I, p_F, p_M are functions that take values in the interval $(0, 1]$. The domain of p_I is I, the domain of p_F is F, and the domain of p_M is T. Function p_M can be extended to computations as follows. For a transition $\tau \in T$ and a computation $c \in T^*$, we write $f(\tau, c)$ to denote the number of occurrences of τ within c. Let (r, w) and (s, v) be two configurations. We define

$$p_M((r, w) \overset{c}{\vdash} (s, v)) = \begin{cases} \prod_{\tau \in T} p_M(\tau)^{f(\tau, c)}, & \text{if } (r, w) \overset{c}{\vdash} (s, v); \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

The probability of a string $w \in \Sigma^*$ is defined as

$$p_M(w) = \sum_{r \in I, s \in F, c} p_I(r) \cdot p_M((r, w) \overset{c}{\vdash} (s, \varepsilon)) \cdot p_F(s). \tag{8}$$

We define the *underlying* FA of a PFA in the obvious way, and let PFAs inherit the properties we defined for FAs. We define $p_M(L) = \sum_{w \in L} p_M(w)$ for each language $L \subseteq \Sigma^*$. Note that $p_M(\Sigma^*) = p_M(L(\mathcal{M}))$.

A PFA $\mathcal{M}$ is said to be *proper* if

$$\sum_{q \in I} p_I(q) = 1, \tag{9}$$

and if for all $q \in Q$

$$p_F(q) + \sum_{a,r} p_M(q \overset{a}{\mapsto} r) = 1, \tag{10}$$

that is, the probability of ending the computation at q and the probabilities of all transitions from state q sum to 1. (In the above formula, we let $p_F(q) = 0$ for $q \notin F$.) A PFA is said to be *consistent* if $p_M(L(\mathcal{M})) = 1$, that is, function p_M defines a probability distribution over the set of strings in Σ^*. A sufficient condition for the consistency of a PFA $\mathcal{M}$ is that $\mathcal{M}$ be proper and reduced [39]. This condition can be easily decided.

A *weighted* FA (WFA) is a tuple $\mathcal{M} = (\Sigma, Q, I, F, T, p_I, p_F, p_M)$ defined as for a PFA, with the only difference that p_M, p_I, p_F take values in the set of positive real numbers. A WFA is said to be *convergent* if $p_M(L(\mathcal{M})) < \infty$.

Again, we make the assumption that the weights of the transitions of our WFAs are all rational numbers and are represented as in the case of WCFGs. The size of a transition τ of the form $r \overset{a}{\mapsto} s$ in a PFA or WFA $\mathcal{M}$ is defined as $\|r \overset{a}{\mapsto} s\| = |ras| + \|p_M(r \overset{a}{\mapsto} s)\|$. The size of $\mathcal{M}$ is defined as

$$\|\mathcal{M}\| = \sum_{r \overset{a}{\mapsto} s} \|r \overset{a}{\mapsto} s\| + \sum_{q \in I} \|p_I(q)\| + \sum_{q \in F} \|p_F(q)\|. \tag{11}$$

3 Weighted intersection

In this section we investigate an extension of a construction originally presented in [4] that computes the intersection of a context-free language and a regular language. We will refer to this extended construction as *weighted intersection*. The input consists of a WCFG $\mathcal{G} = (\Sigma, N, S, R, p_G)$ and a WFA $\mathcal{M} = (\Sigma, Q, I, F, T, p_I, p_F, p_M)$. Note that we assume, without loss of generality, that $\mathcal{G}$ and $\mathcal{M}$ share the same set of terminals Σ.

The output of the construction is a WCFG $\mathcal{G}_\cap = (\Sigma, N_\cap, S_\cap, R_\cap, p_\cap)$, where

$$N_\cap = (Q \times (\Sigma \cup N) \times Q) \cup \{S_\cap\}, \tag{12}$$

$S_\cap$ is a new symbol, and set $R_\cap$ is the smallest set satisfying the following conditions.

- For each pair of states $r \in I, s \in F$, let the rule

  $$\pi_\cap = (S_\cap \to (r, S, s))$$

  be in $R_\cap$, and let $p_\cap(\pi_\cap) = p_I(r) \cdot p_F(s)$.
- For each rule $\pi = (A \to X_1 \cdots X_m) \in R$, $m \geq 0$, and each sequence of states $r_0, \ldots, r_m \in Q$, let the rule

  $$\pi_\cap = ((r_0, A, r_m) \to (r_0, X_1, r_1) \cdots (r_{m-1}, X_m, r_m))$$

  be in $R_\cap$, and let $p_\cap(\pi_\cap) = p_G(\pi)$; for $m = 0$, $R_\cap$ contains a rule $\pi_\cap = ((r_0, A, r_0) \to \varepsilon)$ for each state r_0.
- For each transition $\tau = (r \overset{a}{\mapsto} s) \in T$, let the rule

  $$\pi_\cap = ((r, a, s) \to a)$$

  be in $R_\cap$, and let $p_\cap(\pi_\cap) = p_M(\tau)$.

Note that each derivation $S_\cap \overset{d_\cap}{\Rightarrow} w$ in $\mathcal{G}_\cap$ must have the form $d_\cap = (S_\cap \to (r, S, s)) \cdot d'_\cap$, for some $r \in I$, $s \in F$. Further note that when we provide a PCFG and a PFA as input to the weighted intersection, the resulting WCFG $\mathcal{G}_\cap$ is also a PCFG, although in general $\mathcal{G}_\cap$ may not be proper and may not be consistent. The above facts will be used implicitly on a number of occasions below.

From the definition of $\mathcal{G}_\cap$ it directly follows that, for each rule $(r_0, A, r_m) \to (r_0, X_1, r_1) \cdots (r_{m-1}, X_m, r_m)$ in $R_\cap$, there is a unique rule $A \to X_1 \cdots X_m$ in R from which the former rule was constructed. Similarly, each rule $(r, a, s) \to a$ uniquely identifies a transition $r \overset{a}{\mapsto} s$. This means that a derivation $d_\cap$ in $\mathcal{G}_\cap$ can be divided into a sequence $h_1(d_\cap)$ of rules from $\mathcal{G}$ and a sequence $h_2(d_\cap)$ of transitions from $\mathcal{M}$, where h_1 and h_2 are string homomorphisms that we define point-wise as

$$h_1(\pi_\cap) = \begin{cases} \pi, & \text{if } \pi_\cap = ((r_0, A, r_m) \to (r_0, X_1, r_1) \cdots (r_{m-1}, X_m, r_m)), \\ & \text{and } \pi = (A \to X_1 \cdots X_m); \\ \varepsilon, & \text{if } \pi_\cap = ((r, a, s) \to a) \text{ or } \pi_\cap = (S_\cap \to (r, S, s)). \end{cases}$$

$$h_2(\pi_\cap) = \begin{cases} \tau, & \text{if } \pi_\cap = ((r, a, s) \to a) \text{ and } \tau = (r \overset{a}{\mapsto} s); \\ \varepsilon, & \text{if } \pi_\cap = ((r_0, A, r_m) \to (r_0, X_1, r_1) \cdots (r_{m-1}, X_m, r_m)) \\ & \text{or } \pi_\cap = (S_\cap \to (r, S, s)). \end{cases}$$

We also define $h(d_\cap) = (h_1(d_\cap), h_2(d_\cap))$.

Fix some $w \in \Sigma^*$. It can be easily seen that, if $S_\cap \overset{d_\cap}{\Rightarrow} w$ for some $d_\cap = (S_\cap \to (r, S, s)) \cdot d'_\cap$ with $h(d_\cap) = (d, c)$, then $S \overset{d}{\Rightarrow} w$ and $(r, w) \overset{c}{\vdash} (s, \varepsilon)$. Conversely, if $S \overset{d}{\Rightarrow} w$ for some d and $(r, w) \overset{c}{\vdash} (s, \varepsilon)$ for some $r \in I$, $s \in F$ and c, then there must be a derivation $d_\cap = (S_\cap \to (r, S, s)) \cdot d'_\cap$ such that $h(d_\cap) = (d, c)$ and $S_\cap \overset{d_\cap}{\Rightarrow} w$.

The following lemma can now be stated without further proof.

Lemma 1 *By restricting its domain, h becomes a bijection from $\mathcal{D}(w, \mathcal{G}_\cap)$ to $\mathcal{D}(w, \mathcal{G}) \times (\cup_{r \in I, s \in F} \mathcal{C}_{r,s}(w, \mathcal{M}))$, for each $w \in \Sigma^*$. Furthermore, for each $w \in \Sigma^*$ and for each $d_\cap \in \mathcal{D}(w, \mathcal{G}_\cap)$ with $d_\cap = (S_\cap \to (r, S, s)) \cdot d'_\cap$, we have*

$$p_\cap(S_\cap \overset{d_\cap}{\Rightarrow} w) = p_G(S \overset{h_1(d_\cap)}{\Rightarrow} w) \cdot p_I(r) \cdot p_M((r, w) \overset{h_2(d_\cap)}{\vdash} (s, \varepsilon)) \cdot p_F(s).$$

If we are only interested in the weights assigned to strings by our models, then we can use the following result.

Lemma 2 *For each $w \in \Sigma^*$, we have $p_\cap(w) = p_G(w) \cdot p_M(w)$.*

Proof. Due to the existence of h, with the properties stated in Lemma 1, we can write

$$
\begin{aligned}
p_\cap(w) &= \sum_{d_\cap} p_\cap(S_\cap \overset{d_\cap}{\Rightarrow} w) \\
&= \sum_{(S_\cap \to (r,S,s)) \cdot d'_\cap} p_\cap(S_\cap \overset{(S_\cap \to (r,S,s)) \cdot d'_\cap}{\Rightarrow} w) \\
&= \sum_{(S_\cap \to (r,S,s)) \cdot d'_\cap \in \mathcal{D}(w, \mathcal{G}_\cap)} p_G(S \overset{h_1(d'_\cap)}{\Rightarrow} w) \cdot \\
&\qquad \cdot p_I(r) \cdot p_M((r,w) \overset{h_2(d'_\cap)}{\vdash} (s,\varepsilon)) \cdot p_F(s) \\
&= \sum_d p_G(S \overset{d}{\Rightarrow} w) \cdot \sum_{r \in I, s \in F, c} p_I(r) \cdot p_M((r,w) \overset{c}{\vdash} (s,\varepsilon)) \cdot p_F(s) \\
&= p_G(w) \cdot p_M(w). \quad \blacksquare
\end{aligned}
$$

From Lemma 2 we immediately have that $L(\mathcal{G}_\cap) = L(\mathcal{G}) \cap L(\mathcal{M})$.

In later sections we will also need the following result.

Lemma 3 *If $\mathcal{G}$ and $\mathcal{M}$ are convergent, then so is $\mathcal{G}_\cap$.*

Proof. Using Lemma 2 we can write

$$
\begin{aligned}
p_\cap(L(\mathcal{G}_\cap)) &= \sum_w p_\cap(w) \\
&= \sum_w p_G(w) \cdot p_M(w) \\
&\leq \sum_w p_G(w) \cdot \sum_w p_M(w) \\
&= p_G(L(\mathcal{G})) \cdot p_M(L(\mathcal{M})). \quad \blacksquare
\end{aligned}
$$

All of the above results hold for any grammar $\mathcal{G}_\cap$ constructed by weighted intersection. Note however that, in the general case, $\mathcal{G}_\cap$ may not be a reduced grammar. In what follows, we always assume that $\mathcal{G}_\cap$ has been reduced. Since we have already observed in Section 2 that reduction of a (W)CFG can be carried out in linear time, this assumption does not change the asymptotic complexity of the algorithms that will be presented later.

We conclude the present section with a discussion of the computational complexity of weighted intersection. The most expensive step in the construction of $\mathcal{G}_\cap$ is the construction of rules of the form $(r_0, A, r_m) \to (r_0, X_1, r_1) \cdots (r_{m-1}, X_m, r_m)$ in $R_\cap$. Let $\rho(\mathcal{G})$ be the length of the longest right-hand side of a rule in the input grammar $\mathcal{G}$. Then in the worst case there could be $\Theta(|R| \cdot |Q|^{\rho(\mathcal{G})+1})$ different rules of the above form.

One way to avoid this exponential growth is to preprocess $\mathcal{G}$ by casting it into a normal form that imposes a bound on the length of right-hand sides,

in such a way that a bijection between the derivations in the two grammars is established that preserves the associated weights. One such form is the well-known Chomsky normal form [19], which was extended in [1] to WCFGs. However, such a transformation is problematic in the treatment of so-called empty rules, that is, rules of the form $A \to \varepsilon$. We apply below an alternative normal form for CFGs and extend it to WCFGs.

Given a WCFG $\mathcal{G} = (\Sigma, N, S, R, p_G)$, we define a new WCFG $\mathcal{G}' = (\Sigma, N', S, R', p')$, where

$$N' = N \cup \{[\alpha] \mid (A \to \alpha\beta) \in R,\ |\alpha\beta| \geq 3, |\alpha| \geq 2\}, \tag{13}$$

and set R' satisfies the following conditions.

- For each rule $\pi = (A \to X_1 \cdots X_m) \in R$, $0 \leq m \leq 2$, let π be also in R', and let $p'(\pi) = p_G(\pi)$.
- For each rule $\pi = (A \to X_1 \cdots X_m) \in R$, $m \geq 3$, let the rule

  $$\pi' = (A \to [X_1 \cdots X_m])$$

  be in R', and let $p'(\pi') = p_G(\pi)$.
- For each nonterminal $[X_1 \cdots X_m] \in N'$, $m > 2$, let the rule

  $$\pi' = ([X_1 \cdots X_m] \to [X_1 \cdots X_{m-1}]X_m)$$

  be in R', and let $p'(\pi') = 1$.
- For each nonterminal $[X_1 X_2] \in N'$, let the rule

  $$\pi' = ([X_1 X_2] \to X_1 X_2)$$

  be in R', and let $p'(\pi') = 1$.

Note that $\rho(\mathcal{G}') \leq 2$. Furthermore, it is not difficult to show that, for each $w \in \Sigma^*$, there is a bijection from $\mathcal{D}(w, \mathcal{G})$ to $\mathcal{D}(w, \mathcal{G}')$ that preserves the weight of each derivation. Finally, note that the above transformation linearly expands the number of rules in R and copies their weights to the new rules, so the construction can be easily carried out in time proportional to $\|\mathcal{G}\|$, and therefore $\|\mathcal{G}'\| = \mathcal{O}(\|\mathcal{G}\|)$.

We conclude that the time and space complexity of weighted intersection becomes $\mathcal{O}(\|\mathcal{G}\| \cdot |Q|^3)$. If we take $|Q|$ to be the length of an input string, this is also the time complexity of several practical parsing algorithms for PCFGs. A discussion of the relation between weighted intersection and parsing can be found in [28].

4 Partition functions

In this section we introduce the so-called partition function of a PCFG, which will be used in later sections. We also discuss several methods that have been used in the literature for the computation of this function. Let $\mathcal{G} = (\Sigma, N, S, R, p_G)$ be a PCFG. We define the *partition function* of $\mathcal{G}$ as the function Z that assigns to each $A \in N$ the value

$$Z(A) = \sum_{d,w} p_G(A \stackrel{d}{\Rightarrow} w). \tag{14}$$

Note that $Z(S) = 1$ means that $\mathcal{G}$ is consistent. More generally, one may want to compute the partition function for non-consistent PCFGs. As an example, let $\mathcal{G}$ be a consistent PCFG representing our language model, and consider some property $\mathcal{P}$ of strings that can be characterized by means of a regular expression or, equivalently, by a deterministic FA. We can turn this FA into a PFA such that every generated string is assigned a probability of one (we will see an example of such a construction later in Section 6). We then apply the weighted intersection of Section 3 to $\mathcal{G}$ and our PFA, resulting in a possibly non-consistent PCFG $\mathcal{G}_{\mathcal{P}}$. It is not difficult to see that $Z(S) \leq 1$ in $\mathcal{G}_{\mathcal{P}}$ provides the probability that a string generated by the original language model $\mathcal{G}$ satisfies $\mathcal{P}$. Examples of properties $\mathcal{P}$ of interest are the set of strings that have a prefix or infix u for some fixed $u \in \Sigma^*$ [22,38,30], and the set of strings of length k for some fixed $k \geq 0$ [10].

We can characterize the partition function of a PCFG as a solution of a specific system of equations. Following the approach in [18,9], we introduce generating functions associated with the nonterminals of the grammar. Let $N = \{A_1, A_2, \ldots, A_{|N|}\}$. For each $A_k \in N$, let m_k be the number of rules in R with left-hand side A_k, and assume some fixed order for these rules. For each i with $1 \leq i \leq m_k$, let $A_k \rightarrow \alpha_{k,i}$ be the i-th rule with left-hand side A_k. Recall also that $f(A, \alpha)$ denotes the number of occurrences of symbol A within string α. For each k with $1 \leq k \leq |N|$, the *generating function* associated with A_k is defined as

$$g_{A_k}(z_1, z_2, \ldots, z_{|N|}) = \sum_{i=1}^{m_k} \left(p_G(A_k \rightarrow \alpha_{k,i}) \cdot \prod_{j=1}^{|N|} z_j^{f(A_j, \alpha_{k,i})} \right). \tag{15}$$

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

$$g^{(1)}_{A_k}(z_1, z_2, \ldots, z_{|N|}) = g_{A_k}(z_1, z_2, \ldots, z_{|N|}), \tag{16}$$

and, for $i \geq 2$, by

$$g_{A_k}^{(i)}(z_1, z_2, \ldots, z_{|N|}) = g_{A_k}(g_{A_1}^{(i-1)}(z_1, z_2, \ldots, z_{|N|}), g_{A_2}^{(i-1)}(z_1, z_2, \ldots, z_{|N|}), \ldots, g_{A_{|N|}}^{(i-1)}(z_1, z_2, \ldots, z_{|N|})). \quad (17)$$

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

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

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

The above shows that the values of the partition function provide a solution of the system of $|N|$ equations

$$z_k = g_{A_k}(z_1, z_2, \ldots, z_{|N|}). \quad (18)$$

If $\mathcal{G}$ is a linear PCFG, the equations in (18) are all linear. Then the system has at most one solution. The solution can be obtained in cubic time in $|N|$, and in less than cubic time by using asymptotically faster algorithms for matrix multiplication; see for instance [2,12]. When we take into account the size of the representation of (rational) probabilities of the rules in the input PCFG, these algorithms can still be made to run in polynomial time; see for instance [16]. We can thus state the following lemma without further proof.

Lemma 4 *The partition function associated with a linear PCFG $\mathcal{G}$ with rational probabilities for each rule can be exactly computed in polynomial time in $\|\mathcal{G}\|$.*

In the case of a general PCFG, the equations in (18) are non-linear, and we cannot hope to obtain closed-form expressions for the sought solution, that is, the values of the partition function. Even worse, despite the fact that we have assumed that all the probabilities of the rules in the input PCFG are rational numbers, the sought solution of the system can be composed of irrational numbers, as observed in [15].

Nonetheless, the partition function can still be approximated to any degree of precision by iterative computation of the relation in (17), as done for instance in [38,1]. This corresponds to the so-called fixed-point iteration method, which is well-known in the numerical calculus literature and is frequently applied to systems of non-linear equations because it can be easily implemented. When a number of standard conditions are met, each iteration of (17) adds a fixed number of bits to the precision of the approximated solution [24, Chapter 4]. Since each iteration can easily be implemented to run in polynomial time, this means that we can approximate the partition function of a PCFG in polynomial time in the size of the PCFG itself and in the number of bits of the desired precision.

In practical applications where large PCFGs are empirically estimated from data sets, the standard conditions mentioned above for the polynomial time approximation of the partition function are usually met. However, there are some degenerate cases for which these standard conditions do not hold, resulting in exponential time behaviour of the fixed-point iteration method. This has been firstly observed in [15], where an alternative algorithm is proposed for the approximation of the partition function, based on Newton's method for the solution of non-linear systems.

Experiments with Newton's method for the approximation of partition functions of PCFGs have been carried out by [40,32], showing a considerable improvement over the fixed-point iteration method. As far as we know, it has not yet been proved or refuted whether Newton's method can approximate the partition function in polynomial time in the size of the PCFG itself and in the number of bits of the desired precision. Some of the algorithms we propose in the next sections for the computation of distances between language distributions make use of suitable approximations of the partition function. As will be discussed later, whether these algorithms run in polynomial time or not depends on the above-mentioned open problem.

5 Distances between probabilistic languages

In this section we give an overview of some of the most commonly used distances between pairs of probabilistic languages. We also develop algorithms for the computation of such distances, in case the two languages are generated by a non-ambiguous PCFG and a PFA. As already discussed in Section 1, this extends results in the literature that were stated for pairs of deterministic PFAs. The distances d we treat in this section satisfy the well-known properties of metrics, that is, for probability distributions p, p' and p'' defined on sets of strings, we have

(i) $d(p, p') = 0$ if and only if $p = p'$;
(ii) $d(p, p') = d(p', p)$;
(iii) $d(p, p') + d(p', p'') \geq d(p, p'')$.

We start with the L_2 norm. Let p and p' be probability distributions defined on sets of strings. Without loss of generality, we assume a common alphabet for the two distributions. We define

$$d_2(p, p') = \sqrt{\sum_w (p(w) - p'(w))^2}. \tag{19}$$

We also introduce the notion of *coemission* for distributions p and p', defined as

$$C(p, p') = \sum_w p(w) \cdot p'(w). \tag{20}$$

Note that if p and p' are defined by means of some grammar or automaton model, then $C(p, p')$ is the probability that the two models independently generate the same string.

We can rewrite the definition of d_2 as

$$\begin{aligned} d_2(p, p') &= \sqrt{\sum_w p(w)^2 - 2 \sum_w p(w) \cdot p'(w) + \sum_w p'(w)^2} \\ &= \sqrt{C(p, p) - 2 \cdot C(p, p') + C(p', p')}. \end{aligned} \tag{21}$$

We have thus reduced the problem of computing d_2 to the problem of computing coemission probabilities for the involved distributions.

We now discuss the problem of computing (21) for different combinations of models. Let $\mathcal{G} = (\Sigma, N, S, R, p_G)$ be a PCFG, and let $\mathcal{M} = (\Sigma, Q, I, F, T, p_I, p_F, p_M)$ be a PFA. Using the associated distributions p_G and p_M, equality (21) is instantiated to

$$d_2(p_G, p_M) = \sqrt{C(p_G, p_G) - 2 \cdot C(p_G, p_M) + C(p_M, p_M)}. \tag{22}$$

We first consider term $C(p_G, p_M)$. Let $\mathcal{G}_\cap = (\Sigma, N_\cap, S_\cap, R_\cap, p_\cap)$ be the WCFG defined by the weighted intersection of $\mathcal{G}$ and $\mathcal{M}$ as in Section 3. By Lemma 2 we have $p_\cap(w) = p_G(w) \cdot p_M(w)$ for every $w \in \Sigma^*$. Therefore we can conclude that

$$p_\cap(L(\mathcal{G}_\cap)) = C(p_G, p_M). \tag{23}$$

We can then compute or approximate the coemission $C(p_G, p_M)$ through the value $Z(S_\cap)$ of the partition function Z of $\mathcal{G}_\cap$, as discussed in Section 4.

Consider now term $C(p_G, p_G)$ in (22). From this point onward, we assume that $\mathcal{G}$ is non-ambiguous. We define a second PCFG $\mathcal{G}' = (\Sigma, N, S, R, p'_G)$ with the same underlying CFG, such that $p'_G(\pi) = p_G(\pi)^2$ for every $\pi \in R$. We can now write

$$\begin{aligned} C(p_G, p_G) &= \sum_w p_G(w)^2 \\ &= \sum_{d,w} p_G(S \stackrel{d}{\Rightarrow} w)^2 \\ &= \sum_{d \in \mathcal{D}(\mathcal{G})} \Big(\prod_{A \to \alpha} p_G(A \to \alpha)^{f(A \to \alpha, d)} \Big)^2 \\ &= \sum_{d \in \mathcal{D}(\mathcal{G})} \prod_{A \to \alpha} p'_G(A \to \alpha)^{f(A \to \alpha, d)} \\ &= \sum_w p'_G(w) \quad = \quad p'_G(L(\mathcal{G}')). \end{aligned} \tag{24}$$

Again, we can compute or approximate $C(p_G, p_G)$ through the value $Z(S)$ of the partition function Z of $\mathcal{G}'$, using (24).

We are left with term $C(p_M, p_M)$ from (22). In principle, we could derive equalities for the computation of $C(p_M, p_M)$ from the equalities developed above for the case of $C(p_G, p_M)$. This is because any PFA can be transformed into an equivalent PCFG, deriving the same language and with the same distribution. A more efficient computation of $C(p_M, p_M)$ can be obtained using the algorithm reported in [39] for the computation of the coemission $C(p_M, p'_M)$ for possibly distinct distributions p_M and p'_M, defined by means of PFAs. More precisely, the algorithm proposed in [39] is based on the so-called cross-product construction of two FAs [19], which is then generalized to the probabilistic case. (Such a construction has an obvious similarity to the weighted intersection reported in Section 3.) While the algorithm is defined for deterministic PFAs, it can be easily generalized to the nondeterministic case.

From all of the arguments above, and from the computational analysis of the intersection construction in Section 3, we conclude that the computation of distance d_2 for a non-ambiguous PCFG and a PFA can be reduced in polynomial time to the computation of the partition function of a PCFG. Using Lemma 4 we can then state the following result without further proof.

Theorem 5 *Let $\mathcal{G}$ be a linear and non-ambiguous PCFG, and let $\mathcal{M}$ be a PFA. Let p_G and p_M be the associated distributions. Quantity $d_2(p_G, p_M)^2$ can be exactly computed in polynomial time in $\|\mathcal{G}\|$ and $\|\mathcal{M}\|$.*

The desired distance $d_2(p_G, p_M)$ can be approximated on the basis of $d_2(p_G, p_M)^2$, by applying standard algorithms from numerical analysis that work in polynomial time in the number of bits of the desired precision. Note that this is not needed in many practical applications, as the square of distance $d_2(p_G, p_M)$ is sufficient to compare distances for different models and to evaluate learning curves for convergence.

For the more general case of a distribution p_G associated with a non-ambiguous PCFG, we can use our reduction above to approximate distance $d_2(p_G, p_M)$ through the approximation of the partition function. If the latter problem can be solved in polynomial time, as discussed in Section 4, then distance $d_2(p_G, p_M)$ can be approximated in polynomial time in the size of the input models and in the number of bits of the desired precision.

One might also wonder whether our assumption on non-ambiguity of the PCFG can be dropped. If we do so, we observe a drastic change in the complexity of the problem, as discussed in what follows. Consider a linear PCFG $\mathcal{G}$ and a PFA $\mathcal{M}$, with associated distributions p_G and p_M, respectively. Under these more general conditions we can still exactly compute terms $C(p_G, p_M)$ and $C(p_M, p_M)$ through (23), as discussed above. If we were able to compute $d_2(p_G, p_M)$, then we could easily derive term $C(p_G, p_G)$ from (22). Let us define the *derivational coemission* for $\mathcal{G}$ as

$$C_d(p_G, p_G) = \sum_{d,w} (p(S \stackrel{d}{\Rightarrow} w))^2. \tag{25}$$

We observe that the method implied by (24) also provides a method for the exact computation of $C_d(p_G, p_G)$, this time without the requirement that the grammar be non-ambiguous. Finally, notice that $C_d(p_G, p_G) = C(p_G, p_G)$ if and only if $\mathcal{G}$ is non-ambiguous. However, the problem of testing whether a linear (P)CFG is non-ambiguous is undecidable [25]. This precludes the exact computation of $C(p_G, p_G)$ and thereby of $d_2(p_G, p_M)$ for a linear, possibly ambiguous PCFG.

We now introduce a second distance, which uses the logarithm of probabilities. This is especially convenient in practical applications, since for infinite languages the probabilities of sentences can be arbitrarily small. Let p and p' be probability distributions defined over sets of strings. Below we assume that $\infty - \infty = 0$ and that $\frac{0}{0} = 1$. We define

$$d_{\log}(p, p') = \max_w \left| \log(p(w)) - \log(p'(w)) \right|. \tag{26}$$

We discuss below how to compute (26) in the case of a non-ambiguous PCFG $\mathcal{G} = (\Sigma, N, S, R, p_G)$ and a deterministic PFA $\mathcal{M} = (\Sigma, Q, I, F, T, p_I,$

p_F, p_M), under the assumption that $L(\mathcal{G}) = L(\mathcal{M})$. In order to simplify the notation, we assume F is a singleton $\{q_F\}$, for some $q_F \in Q$, and $p_F(q_F) = 1$. Every deterministic PFA can be transformed into a deterministic PFA of this form, by adding a new final state q_F with $p_F(q_F) = 1$, and introducing a special end-marker not in Σ. The end-marker is appended to each input string, and transitions reading the end-marker are added from each former final state q of the source PFA to q_F, moving the probability mass $p_F(q)$ to the new transition. This preserves the original language distribution, modulo the addition of the end-marker.

We start by rewriting (26) as

$$\begin{aligned} d_{\log}(p_G, p_M) &= \max_w \left| \log(\frac{p_G(w)}{p_M(w)}) \right| \\ &= \max\{\max_w \log(\frac{p_G(w)}{p_M(w)}), -1 \cdot \min_w \log(\frac{p_G(w)}{p_M(w)})\}. \end{aligned} \tag{27}$$

We focus below on the computation of quantity $\max_w \log(\frac{p_G(w)}{p_M(w)})$; the other term in (27) can be computed in a similar way.

We define a WFA $\mathcal{M}' = (\Sigma, Q, I, F, T, p_I, p_F, p'_M)$, such that $p'_M(\tau) = \frac{1}{p_M(\tau)}$ for each transition $\tau \in T$. Note that $\mathcal{M}'$ is not a PFA in general, since the weights $\frac{1}{p_M(\tau)}$ can be strictly greater than 1. Let $\mathcal{G}_\cap = (\Sigma, N_\cap, S_\cap, R_\cap, p_\cap)$ be the WCFG obtained by applying the weighted intersection to $\mathcal{G}$ and $\mathcal{M}'$. By Lemma 2 we have $p_\cap(w) = p_G(w) \cdot p'_M(w)$, for every $w \in \Sigma^*$.

Let us fix some string $w \in L(\mathcal{G})$. By our assumptions, there is a unique derivation d_w for w in $\mathcal{G}$, and a unique computation c_w scanning w in $\mathcal{M}$. We can write

$$\begin{aligned} \frac{p_G(w)}{p_M(w)} &= \frac{\prod_{\pi \in R} p_G(\pi)^{f(\pi, d_w)}}{\prod_{\tau \in T} p_M(\tau)^{f(\tau, c_w)}} \\ &= \prod_{\pi \in R} p_G(\pi)^{f(\pi, d_w)} \cdot \prod_{\tau \in T} \left(\frac{1}{p_M(\tau)}\right)^{f(\tau, c_w)} \\ &= p_\cap(w). \end{aligned} \tag{28}$$

Thus we have

$$\max_w \frac{p_G(w)}{p_M(w)} = \max_{d_\cap, w} p_\cap(S_\cap \overset{d_\cap}{\Rightarrow} w). \tag{29}$$

In words, the above quantity represents the highest weight of a derivation in $\mathcal{G}_\cap$. In the case of a PCFG, several algorithms for its computation can be found

in the literature, running in polynomial time in the size of the input grammar. All of these algorithms also work within the same time bound for WCFGs with derivations of bounded weight. For instance, [11] discusses a polynomial time method, based on dynamic programming, that is an adaptation of Dijkstra's algorithm for the search of the shortest path in a graph with non-negative weights [12]. From the above, we derive the following result.

Theorem 6 *Let $\mathcal{G}$ be a non-ambiguous PCFG, and let $\mathcal{M}$ be a deterministic PFA such that $L(\mathcal{G}) = L(\mathcal{M})$. Let p_G and p_M be the associated distributions. Quantity $2^{d_{\log}(p_G,\, p_M)}$ can be exactly computed in polynomial time in $\|\mathcal{G}\|$ and $\|\mathcal{M}\|$.*

Similarly to the case of distance d_2, we can approximate distance $d_{\log}(p_G, p_M)$ from quantity $2^{d_{\log}(p_G,\, p_M)}$ by applying standard numerical methods for computation of logarithms that work in polynomial time in the number of bits of the desired precision.

If we allow $L(\mathcal{G}) \neq L(\mathcal{M})$, then $d_{\log}(p_G, p_M)$ is undefined, but it may still be useful in practice to compute the max function in (26) restricted to strings w in $L(\mathcal{G}) \cap L(\mathcal{M})$. Property (iii) of metrics, as discussed at the beginning of this section, then no longer holds, but this is not a problem for many applications. For example, one may want to determine a probability assignment p_G to rules of a fixed CFG M that minimizes the distance between distribution p_G and a fixed distribution p_M, among several such assignments. One may compare different choices on the basis of the distance $d_{\log}$ between p_G and p_M, restricted to $L(\mathcal{G}) \cap L(\mathcal{M})$ as discussed above.

We close the present section with a discussion of some other distance measures from the literature on probabilistic language models. Let p and p' be probability distributions defined over sets of strings. We define

$$d_\infty(p, p') = \max_w \; |p(w) - p'(w)| \,. \tag{30}$$

In [26] it is shown that computation of d_∞ is NP-hard for two PFAs, and it remains so even if these automata are acyclic. Another distance related to d_2 is defined as

$$d_1(p, p') = \sum_w \; |p(w) - p'(w)| \,. \tag{31}$$

Again, in [26] it is shown that computation of d_1 is NP-hard for two acyclic PFAs. See also [**?**] for related results. To the best of our knowledge, it is not known whether d_∞ and d_1 can be computed in polynomial time for a linear, non-ambiguous PCFG and a deterministic PFA.

6 Pseudo-distances for probabilistic languages

In this section we consider the Kullback-Leibler (KL) divergence, also called relative entropy, between the string distributions induced by a PCFG and a PFA. As discussed in more detail below, the KL divergence is not a true distance, since it does not satisfy all of the properties of metrics. Therefore we call this measure a 'pseudo-distance'.

The KL divergence is an important concept in information theory, where it is defined between probability distributions having the same domain, and it is commonly used in language modeling to evaluate how well an empirically estimated model fits a reference model; see for instance [27]. Using the weighted intersection from Section 3, we now present a method for the computation of the KL divergence under a number of assumptions to be discussed below.

We start with the necessary definitions. In what follows we view a random variable as a denumerable set X associated with a probability distribution p_X. Let f be a real-valued function defined over X. The expected value of f with respect to distribution p_X is defined as

$$E_{p_X}\ f(x) = \sum_{x \in X}\ p_X(x) \cdot f(x).$$

We write log to represent logarithms in base 2, and we assume that $0 \cdot \log 0 = 0$. The *entropy* of p_X is defined as the expected value of the so-called information function defined over elements of X:

$$\begin{aligned} H(p_X) &= E_{p_X}\ \log \frac{1}{p_X(x)} \\ &= -\sum_{x \in X}\ p_X(x) \cdot \log p_X(x). \end{aligned}$$

If we consider the probability distributions associated with PCFGs, the above notion of entropy can be carried over to the domain of languages and derivations generated by these grammars, as explained below. Throughout this section we let $\mathcal{G} = (\Sigma,\ N,\ S,\ R,\ p_G)$ be a proper and consistent PCFG and $\mathcal{M} = (\Sigma,\ Q,\ I,\ F,\ T,\ p_I,\ p_F,\ p_M)$ be a proper and consistent PFA. For a nonterminal $A \in N$, let us define the *rule entropy* relative to A as the entropy of the distribution p_G on all rules with A in their left-hand sides:

$$H_r(A, p_G) = E_{p_G}\ \log \frac{1}{p_G(A \to \alpha)}$$
$$= -\sum_{\alpha}\ p_G(A \to \alpha) \cdot \log p_G(A \to \alpha). \tag{32}$$

The *derivational entropy* of $\mathcal{G}$ is defined as the entropy of p_G, viewed as a distribution over the set $\mathcal{D}(\mathcal{G})$ of all derivations of $\mathcal{G}$:

$$H_d(p_G) = E_{p_G}\ \log \frac{1}{p_G(S \overset{d}{\Rightarrow} w)}$$
$$= -\sum_{d,w}\ p_G(S \overset{d}{\Rightarrow} w) \cdot \log p_G(S \overset{d}{\Rightarrow} w). \tag{33}$$

Similarly to the derivational entropy, the *sentential entropy* of $\mathcal{G}$ is defined as the entropy of p_G, viewed as a distribution over $L(\mathcal{G})$:

$$H_s(p_G) = E_{p_G}\ \log \frac{1}{p_G(w)}$$
$$= -\sum_{w}\ p_G(w) \cdot \log p_G(w). \tag{34}$$

It is not difficult to see that $H_d(p_G) \geq H_s(p_G)$. Equality holds if and only if $\mathcal{G}$ is non-ambiguous, as shown for instance in [36, Theorem 2.2].

Let us assume that $L(\mathcal{G}) \subseteq L(\mathcal{M})$; we will later drop this constraint. The *Kullback-Leibler divergence* between $\mathcal{G}$ and $\mathcal{M}$ is defined as

$$KL(p_G \,||\, p_M) = E_{p_G}\ \log \frac{p_G(w)}{p_M(w)}$$
$$= \sum_{w}\ p_G(w) \cdot \log \frac{p_G(w)}{p_M(w)}. \tag{35}$$

One can show that $KL(p_G \,||\, p_M) \geq 0$, and that equality holds if and only if p_G and p_M are point-wise equal. However, properties (ii) and (iii) in Section 5 do not hold here.

The KL divergence is commonly related to the notion of *cross-entropy* of $\mathcal{G}$ and $\mathcal{M}$, defined as the expectation under distribution p_G of the information of strings generated by $\mathcal{M}$:

$$H(p_G \,||\, p_M) = E_{p_G}\ \log \frac{1}{p_M(w)}$$
$$= -\sum_{w}\ p_G(w) \cdot \log p_M(w). \tag{36}$$

We have

$$\begin{aligned} KL(p_G \,||\, p_M) &= \sum_w \; p_G(w) \cdot \log \frac{p_G(w)}{p_M(w)} \\ &= \sum_w \; p_G(w) \cdot \log \frac{1}{p_M(w)} + \sum_w \; p_G(w) \cdot \log p_G(w) \\ &= H(p_G \,||\, p_M) - H_s(p_G). \end{aligned} \tag{37}$$

If distribution p_G is fixed and we need to optimize the choice of p_M, then quantity $H(p_G \,||\, p_M)$ can also be used as a measure to evaluate the tightness of the model. More discussion on this will be provided later. In what follows, we develop an algorithm for the computation of the KL divergence through equality (37). We first need to develop methods for the computation of the expectation of the frequency of a rule or a nonterminal over all derivations of a PCFG. These quantities will be used later by our algorithms.

Recall that $f(A \to \alpha, d)$ is the number of occurrences of rule $A \to \alpha$ in a derivation d of $\mathcal{G}$. We similarly define $f(A, d)$ as the number of occurrences of a nonterminal A in left-hand sides of rules in d. We thus have $f(A, d) = \sum_\alpha \; f(A \to \alpha, d)$. Below we consider two related quantities:

$$E_{p_G} \; f(A \to \alpha, d) = \sum_{d,w} \; p_G(S \stackrel{d}{\Rightarrow} w) \cdot f(A \to \alpha, d), \tag{38}$$

$$\begin{aligned} E_{p_G} \; f(A, d) &= \sum_{d,w} \; p_G(S \stackrel{d}{\Rightarrow} w) \cdot f(A, d) \\ &= \sum_\alpha E_{p_G} \; f(A \to \alpha, d). \end{aligned} \tag{39}$$

Different methods for the computation of the above expectations have been proposed in the literature. A method based on the so-called *momentum matrix* is reported in [20]. In [10], the same quantities are computed using a generalization of recursive equalities originally presented in [22]. Both of the above methods assume that the PCFG is proper and consistent. Following [29], we adopt here an alternative approach, based on the notion of outside probabilities, which is related to the inside-outside algorithm [3,8] for the unsupervised estimation of PCFGs from sentence samples by the criterion of maximum likelihood. Such a method is applicable even if a PCFG is not consistent or is not proper. In such cases, $p_G(L(\mathcal{G})) \neq 1$, and thus the definition of expectation should be extended in the obvious way to a deficient distribution.

Consider a rule $A \to \alpha$ and a derivation $S \stackrel{d}{\Rightarrow} w$, $w \in \Sigma^*$, such that $f(A \to \alpha, d) > 0$. We start by observing that we can factorize d at each occurrence of

$A \rightarrow \alpha$. More precisely, assuming $d = \pi_1 \pi_2 \cdots \pi_m$, $m \geq 1$, we fix m_1 such that $\pi_{m_1} = A \rightarrow \alpha$. We can now write d as

$$\begin{aligned} S &\overset{\pi_1 \cdots \pi_{m_1 - 1}}{\Rightarrow} uA\beta \\ &\overset{\pi_{m_1}}{\Rightarrow} u\alpha\beta \\ &\overset{\pi_{m_1+1} \cdots \pi_{m_2}}{\Rightarrow} ux\beta \\ &\overset{\pi_{m_2+1} \cdots \pi_m}{\Rightarrow} uxv, \end{aligned}$$

with $uxv = w$. We call derivation $\pi_{m_1+1} \cdots \pi_{m_2}$ the *inner* part of d, and we call derivations $\pi_1 \cdots \pi_{m_1 - 1}$ and $\pi_{m_2+1} \cdots \pi_m$ the *outer* parts of d, both with respect to occurrence π_{m_1}. Based on this, we can write

$$E_{p_G}\ f(A \rightarrow \alpha, d) = \sum_{\substack{d = \pi_1 \cdots \pi_m, m_1, m_2, \beta, u, v, x: \\ S \overset{d_1}{\Rightarrow} uA\beta, \text{ with } d_1 = \pi_1 \cdots \pi_{m_1 - 1}, \\ (A \rightarrow \alpha) = \pi_{m_1}, \\ \alpha \overset{d_2}{\Rightarrow} x, \text{ with } d_2 = \pi_{m_1+1} \cdots \pi_{m_2}, \\ \beta \overset{d_3}{\Rightarrow} v, \text{ with } d_3 = \pi_{m_2+1} \cdots \pi_m}} \prod_{i=1}^{m} p_G(\pi_i). \tag{40}$$

For each nonterminal A, we can define quantities that are obtained as the sum of the probabilities of all outer parts of derivations at some rule with left-hand side A:

$$out(A) = \sum_{d_1, u, \beta, d_3, v} p_G(S \overset{d_1}{\Rightarrow} uA\beta) \cdot p_G(\beta \overset{d_3}{\Rightarrow} v). \tag{41}$$

It is not difficult to see that the above quantities are not probabilities themselves. More precisely, we could have $out(A) > 1$, since outer parts of derivations for A do not represent disjoint events. In a similar way, we can define quantities that are obtained as the sum of the probabilities of all inner parts of derivations for some string $\alpha \in (\Sigma \cup N)^*$:

$$in(\alpha) = \sum_{d_2, x} p_G(\alpha \overset{d_2}{\Rightarrow} x). \tag{42}$$

We can rewrite (40) by grouping together all of the outer and all of the inner parts of derivations, resulting in

$$E_{p_G}\ f(A \rightarrow \alpha, d) = out(A) \cdot p_G(A \rightarrow \alpha) \cdot in(\alpha). \tag{43}$$

We are left with the computation of quantities $out(A)$ and $in(\alpha)$. We observe that for each $\alpha = X\alpha'$ with $X \in N \cup \Sigma$ and $\alpha' \neq \varepsilon$, we have

$$in(\alpha) = in(X) \cdot in(\alpha'). \tag{44}$$

From (42), we can easily derive

$$in(X) = \begin{cases} 1, & X \in \Sigma; \\ Z(X), & X \in N. \end{cases} \tag{45}$$

We have thus reduced the computation of quantities $in(\alpha)$ to the computation of the partition function of the grammar. These latter quantities can be computed or approximated as discussed in Section 4.

Once quantities $in(\alpha)$ have been computed or approximated, we can derive a system of equations for the computation of quantities $out(A)$. For any $A, B \in N$, we let $\delta(A, B) = 1$ if $A = B$ and $\delta(A, B) = 0$ otherwise. Without loss of generality, we assume that the start symbol S does not occur in the right-hand side of any rule of $\mathcal{G}$. (We can always cast a PCFG in a form that satisfies this assumption, preserving the probability distribution on the generated strings and derivations). We can write

$$out(A) = \delta(A, S) + \sum_{B,\alpha,\beta} out(B) \cdot p_G(B \rightarrow \alpha A \beta) \cdot in(\alpha) \cdot in(\beta). \tag{46}$$

We can now state our first intermediate result.

Lemma 7 *Let $\mathcal{G}$ be a linear and consistent PCFG. For each rule $A \rightarrow \alpha$, we can exactly compute the expectations $E_{p_G} f(A \rightarrow \alpha, d)$ and $E_{p_G} f(A, d)$ in polynomial time in $\|\mathcal{G}\|$.*

Proof. Since $\mathcal{G}$ is linear, we can exactly compute in polynomial time the partition function as stated in Lemma 4. This provides all quantities $in(\alpha)$. The equalities in (46) represent a linear system of $|N|$ equations whose solution can be exactly computed in polynomial time in the size of $\mathcal{G}$ (see the discussion on linear systems in Section 4). Finally, the desired expectations can be computed through equations (43) and (39). ∎

Our next step is to provide a characterization of the derivational entropy of $\mathcal{G}$. Starting from the definition in (33), we use (2) and write

$$
\begin{aligned}
H_d(p_G) &= -\sum_{d,w} p_G(S \overset{d}{\Rightarrow} w) \cdot \log p_G(S \overset{d}{\Rightarrow} w) \\
&= -\sum_{d,w} p_G(S \overset{d}{\Rightarrow} w) \cdot \log \prod_{A \to \alpha} p_G(A \to \alpha)^{f(A \to \alpha, d)} \\
&= -\sum_{d,w} p_G(S \overset{d}{\Rightarrow} w) \cdot \sum_{A \to \alpha} f(A \to \alpha, d) \cdot \log p_G(A \to \alpha) \\
&= -\sum_{A \to \alpha} \log p_G(A \to \alpha) \cdot \sum_{d,w} p_G(S \overset{d}{\Rightarrow} w) \cdot f(A \to \alpha, d) \\
&= -\sum_{A \to \alpha} \log p_G(A \to \alpha) \cdot E_{p_G}\, f(A \to \alpha, d). \qquad (47)
\end{aligned}
$$

We can now state a second intermediate result.

Lemma 8 *Let $\mathcal{G}$ be a linear, consistent PCFG. We can approximate the derivational entropy $H_d(p_G)$ in polynomial time in $\|\mathcal{G}\|$ and in the number of bits of the desired precision.*

Proof. Since $\mathcal{G}$ is linear and consistent, this follows directly from Lemma 7 and equation (47), and from the fact that we can approximate each quantity $\log p_G(A \to \alpha)$. ∎

Under the restrictive assumption that the consistent PCFG $\mathcal{G}$ is also proper, we have $in(\alpha) = 1$ for every α. Using (43), the derivational entropy in (47) can be written as

$$
\begin{aligned}
H_d(p_G) &= -\sum_{A} \sum_{\alpha} \log p_G(A \to \alpha) \cdot out(A) \cdot p_G(A \to \alpha) \cdot in(\alpha) \\
&= -\sum_{A} out(A) \cdot \sum_{\alpha} p_G(A \to \alpha) \cdot \log p_G(A \to \alpha) \\
&= \sum_{A} out(A) \cdot H_r(A, p_G), \qquad (48)
\end{aligned}
$$

where we have used the definition of rule entropy in (32). The characterization in (48) was already known from [17, Theorem 10.7, pp. 90–92]. The proof reported in that work is different from ours and uses the already mentioned momentum matrix. Our characterization in (47) is more general, since we do not assume that $\mathcal{G}$ is a proper PCFG.

We now turn to the main result of this section, developing some equalities for the computation of the cross-entropy of $\mathcal{G}$ and $\mathcal{M}$, defined as in (36). We assume that $\mathcal{M}$ is a deterministic PFA. Furthermore, we assume that F is a singleton $\{q_F\}$, and $p_F(q_F) = 1$. As discussed in Section 5, this is without loss of generality.

Since $\mathcal{M}$ is deterministic, for each sentence $w \in L(\mathcal{M})$ there is a single com-

putation in $\mathcal{C}(w, \mathcal{M})$, which we denote as c_w below. Since $p_I(q_I) = p_F(q_F) = 1$, we can also write

$$p_M(c_w) = \prod_{\tau \in T} p_M(\tau)^{f(\tau, c_w)}. \tag{49}$$

Recall that we are working under the assumption that $L(\mathcal{G}) \subseteq L(\mathcal{M})$. For $d \in \mathcal{D}(\mathcal{G})$, we write $y(d)$ to denote the (unique) string w such that $S \stackrel{d}{\Rightarrow} w$. Using (49), the cross-entropy of $\mathcal{G}$ and $\mathcal{M}$ can now be rewritten as

$$\begin{aligned}
H(p_G \,||\, p_M) &= -\sum_w p_G(w) \cdot \log p_M(w) \\
&= -\sum_w \left(\sum_d p_G(S \stackrel{d}{\Rightarrow} w) \right) \cdot \log p_M(w) \\
&= -\sum_{d,w} p_G(S \stackrel{d}{\Rightarrow} w) \cdot \log \prod_{s \stackrel{a}{\mapsto} t} p_M(s \stackrel{a}{\mapsto} t)^{f(s \stackrel{a}{\mapsto} t, c_w)} \\
&= -\sum_{d,w} p_G(S \stackrel{d}{\Rightarrow} w) \cdot \sum_{s \stackrel{a}{\mapsto} t} f(s \stackrel{a}{\mapsto} t, c_w) \cdot \log p_M(s \stackrel{a}{\mapsto} t) \\
&= -\sum_{s \stackrel{a}{\mapsto} t} \log p_M(s \stackrel{a}{\mapsto} t) \cdot \sum_{d,w} p_G(S \stackrel{d}{\Rightarrow} w) \cdot f(s \stackrel{a}{\mapsto} t, c_w) \\
&= -\sum_{s \stackrel{a}{\mapsto} t} \log p_M(s \stackrel{a}{\mapsto} t) \cdot E_{p_G}\, f(s \stackrel{a}{\mapsto} t, c_{y(d)}).
\end{aligned} \tag{50}$$

Below we elaborate on the computation of quantities $E_{p_G} f(s \stackrel{a}{\mapsto} t, c_{y(d)})$.

Let us introduce a new PFA $\mathcal{M}' = (\Sigma, Q, I, F, T, p_I, p_F, p'_M)$, derived from PFA $\mathcal{M}$ by setting $p'_M(s \stackrel{a}{\mapsto} t) = 1$ for each $s \stackrel{a}{\mapsto} t \in T$. Note that $\mathcal{M}'$ is still a deterministic PFA, and we can therefore associate each string $w \in L(\mathcal{M}')$ with a unique computation c_w of $\mathcal{M}'$. Furthermore, we have $L(\mathcal{M}) = L(\mathcal{M}')$ and, for each $w \in L(\mathcal{M}')$, we have $p'_M(w) = 1$.

Consider now the WCFG $\mathcal{G}_\cap = (\Sigma, N_\cap, S_\cap, R_\cap, p_\cap)$ obtained by applying the weighted intersection of Section 3 to $\mathcal{G}$ and $\mathcal{M}'$. Since $p'_M(w) = 1$ for each $w \in L(\mathcal{M}')$, we can derive from Lemma 1 that, for each $d_\cap \in \mathcal{D}(\mathcal{G}_\cap)$, we have

$$p_\cap(S_\cap \stackrel{d_\cap}{\Rightarrow} w) = p_G(S \stackrel{h_1(d_\cap)}{\Rightarrow} w).$$

From the same lemma we can also establish that, for each $w \in L(\mathcal{G}) \cap L(\mathcal{M}')$ and $d_\cap \in \mathcal{D}(\mathcal{G}_\cap, w)$, we have

$$f(s \stackrel{a}{\mapsto} t, c_w) = f((s, a, t), d_\cap). \tag{51}$$

Finally, observe that since $\mathcal{G}$ is consistent, $\mathcal{G}_\cap$ is also consistent. However, $\mathcal{G}_\cap$ might not be proper. The lack of properness does not affect our algorithm below, since we have nowhere used this property in the intermediate results developed in this section.

Using the above observations, we can now write

$$\begin{aligned} E_{p_G}\ f(s \overset{a}{\mapsto} t, c_{y(d)}) &= \sum_{d,w}\ p_G(S \overset{d}{\Rightarrow} w) \cdot f(s \overset{a}{\mapsto} t, c_w) \\ &= \sum_{d_\cap,w}\ p_\cap(S_\cap \overset{d_\cap}{\Rightarrow} w) \cdot f((s,a,t), d_\cap) \\ &= E_{p_\cap}\ f((s,a,t), d_\cap). \end{aligned} \tag{52}$$

Finally, we combine (52) and (50), resulting in

$$\begin{aligned} H(p_G \,||\, p_M) &= -\sum_{s \overset{a}{\mapsto} t}\ \log p_M(s \overset{a}{\mapsto} t) \cdot E_{p_G}\ f(s \overset{a}{\mapsto} t, c_{y(d)}) \\ &= -\sum_{s \overset{a}{\mapsto} t}\ \log p_M(s \overset{a}{\mapsto} t) \cdot E_{p_\cap}\ f((s,a,t), d_\cap). \end{aligned} \tag{53}$$

We can now state our two main results about the computation of the cross-entropy and of the KL divergence.

Lemma 9 *Let $\mathcal{G}$ be a linear, consistent PCFG and let $\mathcal{M}$ be a deterministic PFA, with associated distributions p_G and p_M, respectively. Assume that $L(\mathcal{G}) \subseteq L(\mathcal{M})$. Then the cross-entropy $H(p_G \,||\, p_M)$ can be approximated in polynomial time in $\|\mathcal{G}\|$ and $\|\mathcal{M}\|$, and in the number of bits of the desired precision.*

Proof. The intersection grammar $\mathcal{G}_\cap$ defined as in the previous discussion is a linear, consistent PCFG, and can be constructed in polynomial time. The statement now follows from Lemma 7 and equation (53), and from the fact that we can approximate quantities $\log p_M(s \overset{a}{\mapsto} t)$. ∎

Theorem 10 *Let $\mathcal{G}$ be a linear, consistent and non-ambiguous PCFG and let $\mathcal{M}$ be a deterministic PFA, with associated distributions p_G and p_M, respectively. Assume that $L(\mathcal{G}) \subseteq L(\mathcal{M})$. Then the KL divergence $KL(p_G \,||\, p_M)$ can be approximated in polynomial time in $\|\mathcal{G}\|$ and $\|\mathcal{M}\|$, and in the number of bits of the desired precision.*

Proof. We have already observed that, for non-ambiguous PCFGs, the sentential entropy and the derivational entropy are the same. From (37) we can therefore write

$$KL(p_G \,||\, p_M) = H(p_G \,||\, p_M) - H_d(p_G). \tag{54}$$

The theorem now follows from Lemma 8 and Lemma 9. ■

A method for the approximated computation of the KL divergence between two distributions associated with deterministic PFAs has been presented in [7]. Our Theorem 10 is a generalization of that result. In [7] a different technique from ours is used, based on a specific factorization of computations of deterministic PFAs. This technique is also applied in [6] to the approximated computation of the KL divergence for certain probabilistic tree languages. The result in [7] has also been extended in [**?**] to distributions associated with non-ambiguous PFAs, that is, PFAs that accept each string by means of exactly one computation.

To conclude the present section, we consider extensions to the above results for more general conditions on $\mathcal{G}$ and $\mathcal{M}$. If we drop the linearity assumption for $\mathcal{G}$, then we can no longer compute the expectations of rule frequencies as in Lemma 7, since we cannot hope to exactly compute the partition function of the grammar. In this case a result on polynomial time approximation of the cross-entropy and the KL divergence depends on the open problem discussed in Section 4 about the polynomial time approximation of the partition function. If $\mathcal{G}$ is linear and possibly ambiguous, equation (54) in the proof of Theorem 10 no longer holds, and we would need to compute instead the sentential entropy $H_s(p_G)$, which seems to be problematic for the following reason. Recall that the sentential entropy and the derivational entropy are the same if and only if the PCFG at hand is non-ambiguous. Under the assumption of linearity of the grammar, we can compute $H_d(p_G)$ on the basis of Lemma 9. However, we cannot hope to obtain a closed-form solution for $H_s(p_G)$, as determining its equality to $H_d(p_G)$ would allow us to decide ambiguity of CFGs, in conflict with the undecidability of this problem [25]. Whether the condition of determinism for $\mathcal{M}$ can be dropped, without altering the polynomial time results above, is still an open problem, but we conjecture that the answer is negative.

As a final remark we point out that, for many purposes, computation of $H_d(p_G)$ is not needed. For example, assume we are given a FA M and are seeking possible functions p_M that extend M to a PFA $\mathcal{M}$, with the goal of minimizing the distance between the distributions induced by $\mathcal{G}$ and $\mathcal{M}$. Then the choice that minimizes $H(p_G \,||\, p_M)$ determines the choice that minimizes $KL(p_G \,||\, p_M)$, irrespective of $H_s(p_G)$. Formally, we can use the above characterization to compute

$$\begin{aligned} p_M^* &= \operatorname*{argmax}_{p_M} \; KL(p_G \,||\, p_M) \\ &= \operatorname*{argmax}_{p_M} H(p_G \,||\, p_M). \end{aligned}$$

Similar approximation problems frequently occur in language modeling applications, where the statistical parameters are sought that result in the tightest model.

When $L(\mathcal{G}) - L(\mathcal{M})$ is non-empty, both $KL(p_G \,||\, p_M)$ and $H(p_G \,||\, p_M)$ are undefined, as their definitions imply a division by $p_M(w) = 0$ for $w \in L(\mathcal{G}) - L(\mathcal{M})$. If an underlying FA M is given, and our task is to compare the relative distances between p_G and p_M for different choices of p_M, we may resort to the following. We exclude strings in $L(\mathcal{G}) - L(\mathcal{M})$ and renormalize p_G to compensate for the restricted domain. In effect, this means we define

$$p_{G|M}(w) = \begin{cases} \frac{p_G(w)}{Z}, & \text{if } w \in L(M); \\ 0, & \text{otherwise,} \end{cases}$$

where Z is the normalization constant $\sum_{w \in L(M)} p_G(w)$. As $p_{G|M}$ is clearly a probability distribution on strings, we can now apply the KL divergence to $p_{G|M}$ and p_M, which we rewrite as

$$\begin{aligned} KL(p_{G|M} \,||\, p_M) &= \sum_w p_{G|M}(w) \cdot \log \frac{p_{G|M}(w)}{p_M(w)} \\ &= \sum_{w \in L(M)} p_{G|M}(w) \cdot \log \frac{p_{G|M}(w)}{p_M(w)} \\ &= \sum_{w \in L(M)} \frac{1}{Z} \cdot p_G(w) \cdot \log \left(\frac{1}{Z} \cdot \frac{p_G(w)}{p_M(w)} \right) \\ &= \sum_{w \in L(M)} \frac{1}{Z} \cdot p_G(w) \cdot \log \frac{1}{Z} \; + \sum_{w \in L(M)} \frac{1}{Z} \cdot p_G(w) \cdot \log \frac{p_G(w)}{p_M(w)} \\ &= \log \frac{1}{Z} \; + \; \frac{1}{Z} \; \cdot \sum_{w \in L(M)} p_G(w) \cdot \log \frac{p_G(w)}{p_M(w)}. \end{aligned} \tag{55}$$

As Z depends only on p_G and M, which we assumed to be fixed, we can identify the choice of p_M that minimizes $KL(p_{G|M} \,||\, p_M)$ on the basis of $\overline{KL}(p_G \,||\, p_M)$, defined as

$$\overline{KL}(p_G \,||\, p_M) = \sum_{w \in L(M)} p_G(w) \cdot \log \frac{p_G(w)}{p_M(w)}. \tag{56}$$

Because strings in $L(\mathcal{M}) - L(\mathcal{G})$ contribute zero values to the above expression, the only strings we need to consider are those in $L(\mathcal{G}) \cap L(\mathcal{M})$. As weighted intersection is by its nature restricted to strings in $L(\mathcal{G}) \cap L(\mathcal{M})$, it follows that the algorithms we have discussed earlier in this section can be straightforwardly applied to approximate $\overline{KL}(p_G \,||\, p_M)$. A related observation is that the

normalization constant Z equals $Z(S_\cap)$, where $S_\cap$ is the start symbol of the PCFG resulting from weighted intersection, as in the construction discussed before.

7 Discussion

Computation of distances between probabilistic languages has important applications in areas such as natural language processing, speech recognition, computational biology and syntactic pattern matching. Most of the algorithms reported in the literature are concerned with the computation of distances between pairs of languages generated by probabilistic finite automata. In this paper we have extended some of these results to pairs of languages generated by a probabilistic context-free grammar and a probabilistic finite automaton, under various conditions.

There are no essential differences between probabilistic finite automata and Hidden Markov Models (HMMs); see [39] for discussion. Our results easily extend to the class of HMMs, which is used more frequently than the class of probabilistic finite automata in such areas as speech recognition and computational biology.

References

[1] S. Abney, D. McAllester, F. Pereira, Relating probabilistic grammars and automata, in: 37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Maryland, USA, 1999.

[2] A. Aho, J. Hopcroft, J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[3] J. Baker, Trainable grammars for speech recognition, in: J. Wolf, D. Klatt (eds.), Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America, 1979.

[4] Y. Bar-Hillel, M. Perles, E. Shamir, On formal properties of simple phrase structure grammars, in: Y. Bar-Hillel (ed.), Language and Information: Selected Essays on their Theory and Application, chap. 9, Addison-Wesley, Reading, Massachusetts, 1964, pp. 116–150.

[5] T. Booth, R. Thompson, Applying probabilistic measures to abstract languages, IEEE Transactions on Computers C-22 (1973) 442–450.

[6] J. Calera-Rubio, R. Carrasco, Computing the relative entropy between regular tree languages, Information Processing Letters 68 (6) (1998) 283–289.

[7] R. Carrasco, Accurate computation of the relative entropy between stochastic regular grammars, RAIRO (Theoretical Informatics and Applications) 31 (5) (1997) 437–444.

[8] E. Charniak, Statistical Language Learning, MIT Press, 1993.

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

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

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

[12] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, The MIT Press, 1990.

[13] R. Duda, P. Hart, D. Stork, Pattern Classification, John Wiley and Sons, New York, NY, 2001.

[14] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, Cambridge, UK, 1999.

[15] K. Etessami, M. Yannakakis, Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations, in: 22nd International Symposium on Theoretical Aspects of Computer Science, vol. 3404 of Lecture Notes in Computer Science, Springer-Verlag, Stuttgart, Germany, 2005.

[16] B. Gregory, E. Kaltofen, Analysis of the binary complexity of asymptotically fast algorithms for linear system solving, SIGSAM Bull. 22 (2) (1988) 41–49.

[17] U. Grenander, Lectures in Pattern Theory, Vol. I: Pattern Synthesis, Springer-Verlag, 1976.

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

[19] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[20] S. Hutchins, Moments of strings and derivation lengths of stochastic context-free grammars, Information Sciences 4 (1972) 179–191.

[21] F. Jelinek, Statistical Methods for Speech Recognition, MIT Press, 1997.

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

[23] D. Jurafsky, J. Martin, Speech and Language Processing, Prentice-Hall, 2000.

[24] C. Kelley, Iterative Methods for Linear and Nonlinear Equations, SIAM, Philadelphia, PA, 1995.

[25] H. Lewis, C. Papadimitriou, Elements of the Theory of Computation, Prentice-Hall, 1981.

[26] R. Lyngso, C. S. Pedersen, The consensus string problem and the complexity of comparing hidden Markov models, Journal of Computing and System Science 65 (2002) 545–569.

[27] C. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, Massachusetts Institute of Technology, 1999.

[28] M.-J. Nederhof, Weighted deductive parsing and Knuth's algorithm, Computational Linguistics 29 (1) (2003) 135–143.

[29] M.-J. Nederhof, A general technique to train language models on language models, Computational Linguistics 31 (2) (2005) 173–185.

[30] M.-J. Nederhof, G. Satta, Probabilistic parsing as intersection, in: 8th International Workshop on Parsing Technologies, LORIA, Nancy, France, 2003.

[31] M.-J. Nederhof, G. Satta, Kullback-Leibler distance between probabilistic context-free grammars and probabilistic finite automata, in: Proc. of the 20^{th} COLING, vol. 1, Geneva, Switzerland, 2004.

[32] M.-J. Nederhof, G. Satta, Using newton's method to compute the partition function of a PCFG, submitted article (2006).

[33] A. Paz, Introduction to Probabilistic Automata, Academic Press, New York, 1971.

[34] E. Santos, Probabilistic grammars and automata, Information and Control 21 (1972) 27–47.

[35] S. Sippu, E. Soisalon-Soininen, Parsing Theory, Vol. I: Languages and Parsing, vol. 15 of EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1988.

[36] S. Soule, Entropies of probabilistic grammars, Information and Control 25 (1974) 57–74.

[37] P. Starke, Abstract Automata, North-Holland Publishing Company, Amsterdam, 1972.

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

[39] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, R. C. Carrasco, Probabilistic finite-state machines – Part I, IEEE Trans. on Pattern analysis and Machine Intelligence 27 (7) (2005) 1013–1025.

[40] D. Wojtczak, K. Etessami, Premo: an analyzer for Probabilistic Recursive Models, to appear in TACAS'07 (2007).