Learning of Structurally Unambiguous Probabilistic Grammars

Dolav Nitay*, Dana Fisman, Michal Ziv-Ukelson

Ben Gurion University

dolavn@post.bgu.ac.il, dana@cs.bgu.ac.il, michaluz@bgu.ac.il

Abstract

The problem of identifying a probabilistic context free grammar has two aspects: the first is determining the grammar's topology (the rules of the grammar) and the second is estimating probabilistic weights for each rule. Given the hardness results for learning context-free grammars in general, and probabilistic grammars in particular, most of the literature has concentrated on the second problem. In this work we address the first problem. We restrict attention to structurally unambiguous weighted context-free grammars (SUWCFG) and provide a query learning algorithm for structurally unambiguous probabilistic context-free grammars (SUPCFG). We show that SUWCFG can be represented using co-linear multiplicity tree automata (CMTA), and provide a polynomial learning algorithm that learns CMTAs. We show that the learned CMTA can be converted into a probabilistic grammar, thus providing a complete algorithm for learning a strucutrally unambiguous probabilistic context free grammar (both the grammar topology and the probabilistic weights) using structured membership queries and structured equivalence queries. We demonstrate the usefulness of our algorithm in learning PCFGs over genomic data.

1 Introduction

Probabilistic context free grammars (PCFGs) constitute a computational model suitable for probabilistic systems which observe non-regular (yet context-free) behavior. They are vastly used in computational linguistics (Chomsky 1956), natural language processing (Church 1988) and biological modeling, for instance, in probabilistic modeling of RNA structures (Grate 1995). Methods for learning PCFGs from experimental data have been thought for over half a century. Unfortunately, there are various hardness results regarding learning context-free grammars in general and probabilistic grammars in particular. It follows from (Gold 1978) that context-free grammars (CFGs) cannot be identified in the limit from positive examples, and from (Angluin 1990) that CFGs cannot be identified in polynomial time using equivalence queries only. Both results are not surprising for those familiar with learning regular languages, as they hold for the class of regular languages as well. However, while regular languages can be learned using both membership queries and equivalence queries (Angluin 1987), it was shown that learning CFGs using both membership queries and equivalence queries is computationally as hard as key cryptographic problems for which there is currently no known polynomial-time algorithm (Angluin and Kharitonov 1995). See more on the difficulties of learning context-free grammars in (de la Higuera 2010, Chapter 15). Hardness results for the probabilistic setting have also been established. (Abe and Warmuth 1992) have shown a computational hardness result for the inference of probabilistic automata, in particular, that an exponential blowup with respect to the alphabet size is inevitable unless **RP** = **NP**.

The problem of identifying a probabilistic grammar from examples has two aspects: the first is determining the rules of the grammar up to variable renaming and the second is estimating probabilistic weights for each rule. Given the hardness results mentioned above, most of the literature has concentrated on the second problem. Two dominant approaches for solving the second problem are the forward-backward algorithm for HMMs (Rabiner 1989) and the inside-outside algorithm for PCFGs (Baker 1979; Lari and Young 1990).

In this work we address the first problem. Due to the hardness results regarding learning probabilistic grammars using membership and equivalence queries (MQ and EQ) we use structured membership queries and structured equivalence queries (SMQ and SEQ), as was done by (Sakakibara 1988) for learning context-free grammars. *Structured strings*, proposed by (Levy and Joshi 1978), are strings over the given alphabet that includes parentheses that indicate the structure of a possible derivation tree for the string. One can equivalently think about a structured string as a derivation tree in which all nodes but the leaves are marked with ?, namely an *unlabeled derivation tree*.

It is known that the set of derivation trees of a given CFG constitutes a *regular tree-language*, where a regular tree-language is a tree-language that can be recognized by a *tree automaton*. (Sakakibara 1988) has generalized Angluin's $\mathbf{L}^*$ algorithm (for learning regular languages using MQ and EQ) to learning a tree automaton, and provided a polynomial learning algorithm for CFGs using SMQ and SEQ. Let $\mathsf{T}(\mathcal{G})$ denote the set of derivation trees of a CFG $\mathcal{G}$, and $\mathsf{S}(\mathsf{T}(\mathcal{G}))$ the set of unlabeled derivation trees (namely the structured strings of $\mathcal{G}$). While a membership query (MQ) asks whether

*This research was partially funded by BSF Grant #2016239, ISF Grant #939/18 & Frankel Center for Computer Science, BGU.

a given string w is in the unknown grammar $\mathcal{G}$, a structured membership query (SMQ) asks whether a structured string s is in $\mathsf{S}(\mathsf{T}(\mathcal{G}))$ and a structured equivalence query (SEQ) answers whether the queried CFG $\mathcal{G}'$ is structurally equivalent to the unknown grammar $\mathcal{G}$, and accompanies a negative answer with a structured string s' in the symmetric difference of $\mathsf{S}(\mathsf{T}(\mathcal{G}'))$ and $\mathsf{S}(\mathsf{T}(\mathcal{G}))$.

In our setting, since we are interested in learning probabilistic grammars, an SMQ on a structured string s is answered by a weight $p \in [0,1]$ standing for the probability for $\mathcal{G}$ to generate s, and a negative answer to an SEQ is accompanied by a structured string s such that $\mathcal{G}$ and $\mathcal{G}'$ generate s with different probabilities (up to a predefined error margin) along with the probability p with which the unknown grammar $\mathcal{G}$ generates s.

(Sakakibara 1988) works with tree automata to model the derivation trees of the unknown grammars. In our case the automaton needs to associate a weight with every tree (representing a structured string). We choose to work with the model of *multiplicity tree automata*. A multiplicity tree automaton (MTA) associates with every tree a value from a given field $\mathbb{K}$. An algorithm for learning multiplicity tree automata, to which we refer as $\mathbf{M}^*$, was developed in (Habrard and Oncina 2006; Drewes and Högberg 2007).[1]

A probabilistic grammar is a special case of a weighted grammar and (Abney, McAllester, and Pereira 1999; Smith and Johnson 2007) have shown that convergent weighted CFGs (WCFG) where all weights are non-negative and probabilistic CFGs (PCFGs) are equally expressive.[2] We thus might expect to be able to use the learning algorithm $\mathbf{M}^*$ to learn an MTA corresponding to a WCFG, and apply this conversion to the result, in order to obtain the desired PCFG. However, as we show in Proposition 4.1, there are probabilistic languages for which applying the $\mathbf{M}^*$ algorithm results in an MTA with negative weights. Trying to adjust the algorithm to learn a positive basis may encounter the issue that for some PCFGs, no finite subset of the infinite Hankel Matrix spans the entire space of the function, as we show in Proposition 4.2.[3] To overcome these issues we restrict attention to structurally unambiguous grammars (SUCFG, see section 4.1), which as we show, can be modeled using co-linear multiplicity automata (defined next).

We develop a polynomial learning algorithm, which we term $\mathbf{C}^*$, that learns a restriction of MTA, which we term *co-linear multiplicity tree automata* (CMTA). We then show that a CMTA for a probabilistic language can be converted into a PCFG, thus yielding a complete algorithm for learning SUPCFGs using SMQs and SEQs as desired.

As a proof-of-concept, in Section 6 we exemplify our algorithm by applying it to a small data-set of genomic data.

Due to lack of space, all the proofs, a complete running example, and the supplementary material for the demonstration section are available in the full version of this paper (Nitay, Fisman, and Ziv-Ukelson 2021).

[1]Following a learning algorithm developed for multiplicity word automata (Beimel et al. 2000).

[2]The definition of *convergent* is deferred to the preliminaries.

[3]The definition of the Hankel Matrix and its role in learning algorithms appears in the sequel.

2 Preliminaries

This section provides the definitions required for *probabilistic grammars* – the object we design a learning algorithm for, and *multiplicity tree automata*, the object we use in the learning algorithm.

2.1 Probabilistic Grammars

Probabilistic grammars are a special case of context free grammars where each production rule has a weight in the range $[0,1]$ and for each non-terminal, the sum of weights of its productions is one. A *context free grammar* (CFG) is a quadruple $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$, where $\mathcal{V}$ is a finite non-empty set of symbols called *variables* or *non-terminals*, Σ is a finite non-empty set of symbols called the *alphabet* or the *terminals*, $R \subseteq \mathcal{V} \times (\mathcal{V} \cup \Sigma)^*$ is a relation between variables and strings over $\mathcal{V} \cup \Sigma$, called the *production rules*, and $S \in \mathcal{V}$ is a special variable called the *start variable*. We assume the reader is familiar with the standard definition of CFGs and of derivation trees. We say that $S \Rightarrow w$ for a string $w \in \Sigma^*$ if there exists a derivation tree t such that all leaves are in Σ and when concatenated from left to right they form w. That is, w is the *yield* of the tree t. In this case we also use the notation $S \Rightarrow_t w$. A CFG $\mathcal{G}$ defines a set of words over Σ, the *language generated by* $\mathcal{G}$, which is the set of words $w \in \Sigma^*$ such that $S \Rightarrow w$, and is denoted $[\![\mathcal{G}]\!]$. For simplicity, we assume the grammar does not derive the empty word.

Weighted grammars A *weighted grammar* (WCFG) is a pair $\langle \mathcal{G}, \theta \rangle$ where $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$ is a CFG and $\theta : R \to \mathbb{R}$ is a function mapping each production rule to a weight in $\mathbb{R}$. A WCFG $\mathcal{W} = \langle \mathcal{G}, \theta \rangle$ defines a function from words over Σ to weights in $\mathbb{R}$. The WCFG associates with a derivation tree t its weight, which is defined as $\mathcal{W}(t) = \prod_{(V \to \alpha) \in R} \theta(V \to \alpha)^{\sharp_t(V \to \alpha)}$ where $\sharp_t(V \to \alpha)$ is the number of occurrences of the production $V \to \alpha$ in the derivation tree t. We abuse notation and treat $\mathcal{W}$ also as a function from Σ^* to $\mathbb{R}$ defined as $\mathcal{W}(w) = \sum_{S \Rightarrow_t w} \mathcal{W}(t)$. That is, the weight of w is the sum of weights of the derivation trees yielding w, and if $w \notin [\![\mathcal{G}]\!]$ then $\mathcal{W}(w) = 0$. If the sum of all derivation trees in $[\![\mathcal{G}]\!]$, namely $\sum_{w \in [\![\mathcal{G}]\!]} \mathcal{W}(w)$, is finite we say that $\mathcal{W}$ is *convergent*.

Probabilistic grammars A *probabilistic grammar* (PCFG) is a WCFG $\mathcal{P} = \langle \mathcal{G}, \theta \rangle$ where $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$ is a CFG and $\theta : R \to [0,1]$ is a function mapping each production rule of $\mathcal{G}$ to a weight in the range $[0,1]$ that satisfies $1 = \sum_{(V \to \alpha_i) \in R} \theta(V \to \alpha_i)$ for every $V \in \mathcal{V}$.[4] One can see that if $\mathcal{P}$ is a PCFG then the sum of all derivations equals 1, thus $\mathcal{P}$ is convergent.

2.2 Word/Tree Series and Multiplicity Automata

While words are defined as sequences over a given alphabet, trees are defined using a *ranked alphabet*, an alphabet $\Sigma =$

[4]Probabilistic grammars are sometimes called *stochastic grammars (SCFGs)*.

$$M_\eta = \begin{pmatrix} c^1_{111} & c^1_{112} & c^1_{121} & c^1_{122} & c^1_{211} & c^1_{212} & c^1_{221} & c^1_{222} \\ c^2_{111} & c^2_{112} & c^2_{121} & c^2_{122} & c^2_{211} & c^2_{212} & c^2_{221} & c^2_{222} \end{pmatrix} \quad P_{xyz} = \begin{pmatrix} x_1y_1z_1 \\ x_1y_1z_2 \\ x_1y_2z_1 \\ \dots \\ x_2y_2z_2 \end{pmatrix}$$

Figure 1: A matrix M_η for a multi-linear function η and a vector P_{xyz} for the respective 3 parameters.

$\{\Sigma_0, \Sigma_1, \dots, \Sigma_p\}$ which is a tuple of alphabets Σ_k where Σ_0 is non-empty. Let *Trees*(Σ) be the set of trees over Σ, where a node labeled $\sigma \in \Sigma_k$ for $0 \leq k \leq p$ has exactly k children. While a *word language* is a function mapping all possible words (elements of Σ^*) to $\{0, 1\}$, a *tree language* is a function from all possible trees (elements of *Trees*(Σ)) to $\{0, 1\}$. We are interested in assigning each word or tree a non-Boolean value, usually a weight $p \in [0, 1]$. Let $\mathbb{K}$ be a field. We are interested in functions mapping words or trees to values in $\mathbb{K}$. A function from Σ^* to $\mathbb{K}$ is called a *word series*, and a function from *Trees*(Σ) to $\mathbb{K}$ is referred to as a *tree series*.

Word automata are machines that recognize word languages, i.e. they define a function from Σ^* to $\{0, 1\}$. *Tree automata* are machines that recognize tree languages, i.e. they define a function from *Trees*(Σ) to $\{0, 1\}$. *Multiplicity word automata* (MA) are machines to implement word series, i.e. they define a function from Σ^* to $\mathbb{K}$. *Multiplicity tree automata* (MTA) are machines to implement tree series, i.e. they define a function from *Trees*(Σ) to $\mathbb{K}$. Multiplicity automata can be thought of as an algebraic extension of automata, in which reading an input letter is implemented by matrix multiplication. In a multiplicity word automaton with dimension m over alphabet Σ, for each $\sigma \in \Sigma$ there is an m by m matrix, μ_σ, whose entries are values in $\mathbb{K}$ where intuitively the value of entry $\mu_\sigma(i, j)$ is the weight of the passage from state i to state j. The definition of multiplicity tree automata is a bit more involved; it makes use of multilinear functions as defined next.

Multilinear functions Let $\mathbb{V} = \mathbb{K}^d$ be the d dimensional vector space over $\mathbb{K}$. Let $\eta : \mathbb{V}^k \to \mathbb{V}$ be a k-linear function. We can represent η by a d by d^k matrix over $\mathbb{K}$. For instance, if $\eta : \mathbb{V}^3 \to \mathbb{V}$ and $d = 2$ (i.e. $\mathbb{V} = \mathbb{K}^2$) then η can be represented by the 2×2^3 matrix M_η provided in Fig 1 where $c^i_{j_1 j_2 j_3} \in \mathbb{K}$ for $i, j_1, j_2, j_3 \in \{1, 2\}$. Then η, a function taking k parameters in $\mathbb{V} = \mathbb{K}^d$, can be computed by multiplying the matrix M_η with a vector for the parameters for η. Continuing this example, given the parameters $\mathbf{x} = (x_1\ x_2)$, $\mathbf{y} = (y_1\ y_2)$, $\mathbf{z} = (z_1\ z_2)$ the value $\eta(\mathbf{x}, \mathbf{y}, \mathbf{z})$ can be calculated using the multiplication $M_\eta P_{xyz}$ where the vector P_{xyz} of size 2^3 is provided in Fig 1. In general, if $\eta : \mathbb{V}^k \to \mathbb{V}$ is such that $\eta(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k) = \mathbf{y}$ and M_η, the matrix representation of η, is defined using the constants $c^i_{j_1 j_2 \dots j_k}$ then

$$\mathbf{y}[i] = \sum_{\{(j_1, j_2, \dots, j_k) \in \{1, 2, \dots, d\}^k\}} c^i_{j_1 j_2 \dots j_k} \mathbf{x}_1[j_1] \mathbf{x}_2[j_2] \cdots \mathbf{x}_k[j_k]$$

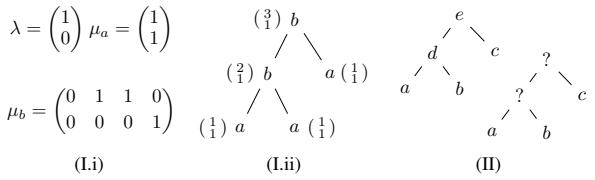

Figure 2: (I.i) An MTA $\mathcal{M} = ((\Sigma_0, \Sigma_2), \mathbb{R}, 2, \mu, \lambda)$ where $\Sigma_0 = \{a\}$ and $\Sigma_2 = \{b\}$ implementing a tree series that returns the number of leaves in the tree. (I.ii) a tree where a node t is annotated by $\mu(t)$. Since $\mu(t_\epsilon) = \binom{3}{1}$, where t_ϵ is the root, the value of the entire tree is $\lambda \cdot \binom{3}{1} = 3$. (II) a derivation tree and its corresponding skeletal tree, which can be written as the structured string $((ab)c)$.

Multiplicity tree automata A *multiplicity tree automaton* (MTA) is a tuple $\mathcal{M} = (\Sigma, \mathbb{K}, d, \mu, \lambda)$ where $\Sigma = \{\Sigma_0, \Sigma_1, \dots, \Sigma_p\}$ is the given ranked alphabet, $\mathbb{K}$ is a field corresponding to the range of the tree-series, d is a non-negative integer called the automaton *dimension*, μ and λ are the transition and output function, respectively, whose types are defined next. Let $\mathbb{V} = \mathbb{K}^d$. Then λ is an element of $\mathbb{V}$, namely a d-vector over $\mathbb{K}$. Intuitively, λ corresponds to the final values of the "states" of $\mathcal{M}$. The transition function μ maps each element σ of Σ to a dedicated transition function μ_σ such that given $\sigma \in \Sigma_k$ for $0 \leq k \leq p$ then μ_σ is a k-linear function from $\mathbb{V}^k$ to $\mathbb{V}$. The transition function μ induces a function from *Trees*(Σ) to $\mathbb{V}$, defined as follows. If $t = \sigma$ for some $\sigma \in \Sigma_0$, namely t is a tree with one node which is a leaf, then $\mu(t) = \mu_\sigma$ (note that μ_σ is a vector in $\mathbb{K}^d$ when $\sigma \in \Sigma_0$). If $t = \sigma(t_1, \dots, t_k)$, namely t is a tree with root $\sigma \in \Sigma_k$ and children $t_1, \dots, t_k$ then $\mu(t) = \mu_\sigma(\mu(t_1), \dots, \mu(t_k))$. The automaton $\mathcal{M}$ induces a total function from *Trees*(Σ) to $\mathbb{K}$ defined as follows: $\mathcal{M}(t) = \lambda \cdot \mu(t)$. Fig. 2(I.i) provides an example of an MTA and the value for a computed tree Fig. 2(I.ii).

Contexts In the course of our learning algorithm we need a way to compose trees, more accurately we compose trees with *contexts* as defined next. Let $\Sigma = \{\Sigma_0, \Sigma_1, \dots, \Sigma_p\}$ be a ranked alphabet. Let $\diamond$ be a symbol not in Σ. We use *Trees*$_\diamond(\Sigma)$ to denote all non-empty trees over $\Sigma' = \{\Sigma_0 \cup \{\diamond\}, \Sigma_1, \dots, \Sigma_p\}$ in which $\diamond$ appears exactly once. We refer to an element of *Trees*$_\diamond(\Sigma)$ as a *context*. Note that at most one child of any node in a context c is a context; the other ones are pure trees (i.e. elements of *Trees*(Σ)). Given a tree $t \in$ *Trees*(Σ) and context $c \in$ *Trees*$_\diamond(\Sigma)$ we use $c[\![t]\!]$ for the tree $t' \in$ *Trees*(Σ) obtained from c by replacing $\diamond$ with t.

Structured tree languages/series Recall that our motivation is to learn a word (string) series rather than a tree series, and due to hardness results on learning CFGs and PCFGs we resort to using *structured strings*. A *structured string* is a string with parentheses exposing the structure of a derivation tree for the corresponding trees, as exemplified in Fig. 2 (II). A *skeletal alphabet* is a ranked alphabet in which we use a special symbol $? \notin \Sigma_0$ and for every $0 < k \leq p$ the set Σ_k consists only of the symbol $?$. For $t \in$ *Trees*(Σ), the skeletal description of t, denoted by $\mathsf{S}(t)$, is a tree with

the same topology as t, in which the symbol in all internal nodes is ?, and the symbols in all leaves are the same as in t. Let T be a set of trees. The corresponding skeletal set, denoted $\mathsf{S}(T)$ is $\{\mathsf{S}(t) \mid t \in T\}$. Going from the other direction, given a skeletal tree s we use $\mathsf{T}(s)$ for the set $\{t \in \mathit{Trees}(\Sigma) \mid \mathsf{S}(t) = s\}$.

A tree language over a skeletal alphabet is called a *skeletal tree language*. And a mapping from skeletal trees to $\mathbb{K}$ is called a *skeletal tree series*. Let $\mathcal{T}$ denote a tree series mapping trees in $\mathit{Trees}(\Sigma)$ to $\mathbb{K}$. By abuse of notations, given a skeletal tree s, we use $\mathcal{T}(s)$ for the sum of values $\mathcal{T}(t)$ for every tree t of which $s = \mathsf{S}(t)$. That is, $\mathcal{T}(s) = \sum_{t \in \mathsf{T}(s)} \mathcal{T}(t)$. Thus, given a tree series $\mathcal{T}$ (possibly generated by a WCFG or an MTA) we can treat $\mathcal{T}$ as a skeletal tree series.

3 From Positive MTAs to PCFGs

Our learning algorithm for probabilistic grammars builds on the relation between WCFGs with positive weights (henceforth PWCFGs) and PCFGs (Abney, McAllester, and Pereira; Smith and Johnson). In particular, we first establish that a *positive multiplicity tree automaton* (PMTA), which is a multiplicity tree automaton (MTA) where all weights of both μ and λ are positive, can be transformed into an equivalent WCFG $\mathcal{W}$. That is, we show that a given PMTA $\mathcal{A}$ over a skeletal alphabet can be converted into a WCFG $\mathcal{W}$ such that for every structured string s we have that $\mathcal{A}(s) = \mathcal{W}(s)$. If the PMTA defines a convergent tree series (namely the sum of weights of all trees is finite) then so will the constructed WCFG. Therefore, given that the WCFG describes a probability distribution, we can apply the transformation of WCFG to a PCFG (Abney, McAllester, and Pereira; Smith and Johnson) to yield a PCFG $\mathcal{P}$ such that $\mathcal{W}(s) = \mathcal{P}(s)$, obtaining the desired PCFG for the unknown tree series.

Transforming a PMTA into a PWCFG Let $\mathcal{A} = (\Sigma, \mathbb{R}_+, d, \mu, \lambda)$ be a PMTA over the skeletal alphabet $\Sigma = \{\Sigma_0, \Sigma_1, \ldots, \Sigma_p\}$. We define a PWCFG $\mathcal{W}_\mathcal{A} = (\mathcal{G}_\mathcal{A}, \theta)$ for $\mathcal{G}_\mathcal{A} = (\mathcal{V}, \Sigma_0, R, S)$ as provided in Fig. 3 where $c^i_{i_1,i_2,\ldots,i_p}$ is the respective coefficient in the matrix corresponding to $\mu_?$ for $? \in \Sigma_k$, $1 \le k \le p$. Proposition 3.1 states that the transformation preserves the weights.

Proposition 3.1. $\mathcal{W}(t) = \mathcal{A}(t)$ *for every* $t \in \mathit{Trees}(\Sigma)$.

$$\mathcal{V} = \{S\} \cup \{V_i \mid 1 \le i \le d\}$$

$$R = \{S \to V_i \mid 1 \le i \le d\} \cup \{V_i \to \sigma \mid 1 \le i \le d,\ \sigma \in \Sigma_0\} \cup \left\{ V_i \to V_{i_1} V_{i_2} \ldots V_{i_k} \;\middle|\; \begin{array}{c} 1 \le i, i_j \le d, \\ \text{for } 1 \le j \le k \end{array} \right\}$$

$$\theta(S \to V_i) = \lambda[i]$$

$$\theta(V_i \to \sigma) = \mu_\sigma[i]$$

$$\theta(V_i \to V_{i_1} V_{i_2} \ldots V_{i_k}) = c^i_{i_1,i_2,\ldots,i_k}$$

Figure 3: Transforming a PMTA into a PCFG

In this paper we consider *structurally unambiguous* WCFGs and PCFGs (in short SUWCFGs and SUPCFGs, resp.) as defined in §4. In Thm. 5.1, given in §5, we show that we can learn a PMTA for a SUWCFG in polynomial time using a polynomial number of queries (see exact bounds there), thus obtaining the following result.

Corollary 3.2. *SUWCFGs can be learned in polynomial time using* SMQ*s and* SEQ*s, where the number of* SEQ*s is bounded by the number of non-terminal symbols.*

The overall learning time for SUPCFG relies, on top of Corollary 3.2, on the complexity of converting a WCFG into a PCFG (Abney, McAllester, and Pereira 1999), for which an exact bound is not provided, but the method is reported to converge quickly (Smith and Johnson 2007, §2.1).

4 Learning Struc. Unamb. PCFGs

In this section we discuss the setting of the algorithm, the ideas behind Angluin-style learning algorithms, and the issues with using current algorithms to learn PCFGs. As in $\mathbf{G}^*$(the algorithm for CFGs (Sakakibara 1988)), we assume an oracle that can answer two types of queries: *structured membership queries* (SMQ) and *structured equivalence queries* (SEQ) regarding the unknown regular tree series $\mathcal{T}$ (over a given ranked alphabet Σ). Given a structured string s, the query $\text{SMQ}(s)$ is answered with the value $\mathcal{T}(s)$. Given an automaton $\mathcal{A}$ the query $\text{SEQ}(\mathcal{A})$ is answered "yes" if $\mathcal{A}$ implements the skeletal tree series $\mathcal{T}$ and otherwise the answer is a pair $(s, \mathcal{T}(s))$ where s is a structured string for which $\mathcal{T}(s) \neq \mathcal{A}(s)$ (up to a predefined error).

Our starting point is the learning algorithm $\mathbf{M}^*$ (Habrard and Oncina 2006) which learns MTA using SMQs and SEQs. First we explain the idea behind this and similar algorithms, next the issues with applying it as is for learning PCFGs, then the idea behind restricting attention to strucutrally unambiguous grammars, and finally our algorithm itself.

Hankel Matrix All the generalizations of $\mathbf{L}^*$ (the algorithm for learning regular languages using MQs and EQs, that introduced this learning paradigm (Angluin 1987)) share a general idea that can be explained as follows. A word or tree language as well as a word or tree series can be represented by its Hankel Matrix. The Hankel Matrix has infinitely many rows and infinitely many columns. In the case of word series both rows and columns correspond to an infinite enumeration $w_0, w_1, w_2, \ldots$ of words over the given alphabet. In the case of tree series, the rows correspond to an infinite enumeration of trees $t_0, t_1, t_2, \ldots$ (where $t_i \in \mathit{Trees}(\Sigma)$) and the columns to an infinite enumeration of contexts $c_0, c_1, c_2, \ldots$ (where $c_i \in \mathit{Trees}_\diamond(\Sigma)$). In the case of words, the entry $H(i,j)$ holds the value for the word $w_i \cdot w_j$, and in the case of trees it holds the value of the tree $c_j[\![t_i]\!]$. If the series is *regular* there should exists a finite number of rows in this infinite matrix, which we term *basis*, such that all other rows can be represented using rows in the basis. In the case of $\mathbf{L}^*$, and $\mathbf{G}^*$(that learn word-languages and tree-languages, resp.) rows that are not in the basis should be equal to rows in the basis. The rows of the basis correspond to the automaton states, and the equalities to other rows determines the transition relation. In the case of $\mathbf{M}^*$(that learns tree-series) rows not in the basis should be expressible as a linear combination of rows in the basis, and the linear combinations determines the weights of the extracted automaton. In our case, in order to apply the PMTA to PCFG conversion we need the algorithm to find a *positive linear combination* of rows to act as

the basis, so that the extracted automaton will be a PMTA. In all cases we would like the basis to be *minimal* in the sense that no row in the basis is expressible using other rows in the basis. This is since the size of the basis derives the dimension of the automaton, and obviously we prefer smaller automata.

Positive linear spans An interest in *positive linear combinations* occurs also in the research community studying convex cones and derivative-free optimizations and a theory of positive linear combinations has been developed (Cohen and Rothblum 1993; Regis 2016).[5] We need the following definitions and results.

The *positive span* of a finite set of vectors $S = \{v_1, v_2, ..., v_k\} \subseteq \mathbb{R}^n$ is defined as follows:

$$\textit{span}_+(S) = \{\lambda_1 \cdot v_1 + \lambda_2 \cdot v_2 + ... + \lambda_k \cdot v_k \mid \lambda_i \geq 0, \forall 1 \leq i \leq k\}$$

A set of vectors $S = \{v_1, v_2, ..., v_k\} \subseteq \mathbb{R}^n$ is *positively dependent* if some v_i is a positive combination of the other vectors; otherwise, it is *positively independent*. Let $A \in \mathbb{R}^{m \times n}$. We say that A is *nonnegative* if all of its elements are nonnegative. The *nonnegative column (resp. row) rank* of A, denoted *c-rank*$_+(A)$ (resp. *r-rank*$_+(A)$), is defined as the smallest nonnegative integer q for which there exist a set of column- (resp. row-) vectors $V = \{v_1, v_2, ..., v_q\}$ in $\mathbb{R}^m$ such that every column (resp. row) of A can be represented as a positive combination of V. It is known that *c-rank*$_+(A)$ = *r-rank*$_+(A)$ for any matrix A (Cohen and Rothblum 1993). Thus one can freely use *rank*$_+(A)$ for *positive rank*, to refer to either one of these.

Issues with positive spans The first question that comes to mind, is whether we can use the $\mathbf{M}^*$ algorithm as is – to learn a positive tree series. We show that this is not the case. In particular, there are positive tree series for which applying the $\mathbf{M}^*$ algorithm results in an MTA with negative weights. Moreover, this holds also if we consider word (rather than tree) series, and if we restrict the weights to be probabilistic (rather than simply positive).

Proposition 4.1. *There exists a probabilistic word series for which the $\mathbf{M}^*$ alg. may return an MTA with negative weights.*

The proof shows this is the case for the word series over alphabet $\Sigma = \{a, b, c\}$ which assigns the following six strings: aa, ab, ac, ba, cb, cc probability of $\frac{1}{6}$ each, and probability 0 to all other strings.

Hence, we turn to ask whether we can adjust the algorithm $\mathbf{M}^*$ to learn a positive basis. We note first that working with positive spans is much trickier than working with general spans, since for $d \geq 3$ there is no bound on the size of a positively independent set in $\mathbb{R}^d_+$ (Regis 2016). To apply the ideas of the Angluin-style query learning algorihtms we need the Hankel Matrix (which is infinite) to contain a finite sub-matrix with the same rank. Unfortunately, as we show next, there exists a probabilistic (thus positive) tree series $\mathcal{T}$ that can be recognized by a PMTA, but none of its finite-sub-matrices span the entire space of $H_{\mathcal{T}}$.

[5]Throughout the paper we use the terms *positive* and *nonnegative* interchangeably.

Proposition 4.2. *There exists a PCFG $\mathcal{G}$ s.t. for the Hankel Matrix $H_{\mathcal{G}}$ corresponding to its tree-series $\mathcal{T}_G$ no finite number of rows positively spans the entire matrix.*

Thus, running $\mathbf{M}^*$on $\mathcal{T}_G$ would yield an MTA with negative weights. The proof shows this is the case for the following PCFG:

$$N_1 \longrightarrow aN_1\ [\tfrac{1}{2}]\ \mid\ aN_2\ [\tfrac{1}{3}]\ \mid\ aa\ [\tfrac{1}{6}]$$
$$N_2 \longrightarrow aN_1\ [\tfrac{1}{4}]\ \mid\ aN_2\ [\tfrac{1}{4}]\ \mid\ aa\ [\tfrac{1}{2}]$$

4.1 Focusing on Structurally Unambiguous CFGs

To overcome these obstacles we restrict attention to structurally unambiguous CFGs (SUCFGs) and their weighted/probabilistic versions (SUWCFGs/SUPCFGs). A context-free grammar is termed *ambiguous* if there exists more than one derivation tree for the same word. We term a CFG *structurally ambiguous* if there exists more than one derivation tree with the same structure for the same word. A context-free language is termed *inherently ambiguous* if it cannot be derived by an unambiguous CFG. Note that a CFG which is unambiguous is also structurally unambiguous, while the other direction is not necessarily true. For instance, the language $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$ which is inherently ambiguous (Hopcroft and Ullman 1979, Thm. 4.7) is not inherently structurally ambiguous. Therefore we have relaxed the classical unambiguity requirement.

The Hankel Matrix and MTA for SUPCFG Recall that the Hankel Matrix considers skeletal trees. Therefore if a word has more than one derivation tree with the same structure, the respective entry in the matrix holds the sum of weights for all derivations. This makes it harder for the learning algorithm to infer the weight of each tree separately. By choosing to work with structurally unambiguous grammars, we overcome this difficulty as an entry corresponds to a single derivation tree.

To discuss properties of the Hankel Matrix for an SUPCFG we need the following definitions. Let H be a matrix, t a tree (or row index) c a context (or column index), T a set of trees (or row indices) and C a set of contexts (or column indices). We use $H[t]$ (resp. $H[c]$) for the row (resp. column) of H corresponding to t (resp. c). Similarly we use $H[T]$ and $H[C]$ for the corresponding sets of rows or columns. Finally, we use $H[t][C]$ for the restriction of H to row t and columns $[C]$.

Two vectors, $v_1, v_2 \in \mathbb{R}^n$ are co-linear with a scalar $\alpha \in \mathbb{R}$ for some $\alpha \neq 0$ iff $v_1 = \alpha \cdot v_2$. Given a matrix H, and two trees t_1 and t_2, we say that $t_1 \ltimes^{\alpha}_H t_2$ iff $H[t_1]$ and $H[t_2]$ are co-linear, with scalar $\alpha \neq 0$. That is, $H[t_1] = \alpha \cdot H[t_2]$. Note that if $H[t_1] = H[t_2] = \bar{0}$, then $t_1 \ltimes^{\alpha}_H t_2 \ltimes^{\alpha}_H t_1$ for every $\alpha > 0$. We say that $t_1 \equiv_H t_2$ if $t_1 \ltimes^{\alpha}_H t_2$ for some $\alpha \neq 0$. It is not hard to see that $\equiv_H$ is an equivalence relation.

The following proposition states that in the Hankel Matrix of an SUPCFG, the rows of trees that are rooted by the same non-terminal are co-linear.

Proposition 4.3. *Let H be the Hankel Matrix of an SUPCFG. Let t_1, t_2 be derivation trees rooted by the same*

non-terminal. Assume $\mathbb{P}(t_1), \mathbb{P}(t_2) > 0$. *Then* $t_1 \ltimes_H^\alpha t_2$ *for some* $\alpha \neq 0$.

We can thus conclude that the number of equivalence classes of $\equiv_H$ for an SUPCFG is finite and bounded by the number of non-terminals plus one (for the zero vector).

Corollary 4.4. *The skeletal tree-set for an SUPCFG has a finite number of equivalence classes under* $\equiv_H$.

Next we would like to reveal the restrictions that can be imposed on a PMTA that corresponds to an SUPCFG. We term an MTA *co-linear* (and denote it CMTA) if in every column of every transition matrix μ_σ there is at most one entry which is non-negative.

Proposition 4.5. *A CMTA can represent an SUPCFG.*

The proof relies on showing that a WCFG is structurally unambiguous iff it is invertible and converting an invertible WCFG into a PMTA yields a CMTA.[6]

5 The Learning Algorithm

Let $\mathcal{T} : \mathit{Trees}(\Sigma) \to \mathbb{R}$ be an unknown tree series, and let H be its Hankel Matrix. The learning algorithm *LearnCMTA* (or $\mathbf{C}^*$, for short), provided in Alg. 1, maintains a data structure called an *observation table*. An observation table for $\mathcal{T}$ is a quadruple (T, C, H, B). Where $T \subseteq \mathit{Trees}(\Sigma)$ is a set of row titles, $C \subseteq \mathit{Trees}_\diamond(\Sigma)$ is a set of column titles, $H : T \times C \to \mathbb{R}$ is a sub-matrix of H, and $B \subset T$, the so called *basis*, is a set of row titles corresponding to rows of H that are co-linearly independent. The algorithm starts with an almost empty observation table, where $T = \emptyset$, $C = \diamond$, $B = \emptyset$ and uses procedure *Complete*(T, C, H, B, Σ_0) to add the nullary symbols of the alphabet to the row titles, uses SMQ queries to fill in the table until certain criteria hold on the observation, namely it is *closed* and *consistent*, as defined in the sequel. Once the table is closed and consistent, it is possible to extract from it a CMTA $\mathcal{A}$ (as we shortly explain). The algorithm then issues the query SEQ$(\mathcal{A})$. If the result is "yes" the algorithm returns $\mathcal{A}$ which was determined to be structurally equivalent to the unknown series. Otherwise, the algorithm gets in return a counterexample $(s, \mathcal{T}(s))$, a structured string in the symmetric difference of $\mathcal{A}$ and $\mathcal{T}$, and its value. It then uses *Complete* to add all prefixes of t to T and uses SMQs to fill in the entries of the table until the table is once again closed and consistent.

Algorithm 1 *LearnCMTA*(T, C, H, B).

```
Initialize B ← ∅, T ← ∅, C ← {◇}
Complete(T, C, H, B, Σ_0)
while true do
    A ← ExtractCMTA(T, C, H, B)
    t ← SEQ(A)
    if t is null then
        return A
    Complete(T, C, H, B, Pref(t))
```

[6] A CFG $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$ is said to be invertible if and only if $A \to \alpha$ and $B \to \alpha$ in R implies $A = B$ (Sakakibara 1992).

Algorithm 2 *Close*(T, C, H, B).

```
while ∃t ∈ Σ(T) s.t. H[t] is co-linearly independent from T do
    B ← B ∪ {t}
    T ← T ∪ {t}
```

Algorithm 3 *Consistent*(T, C, H, B)

```
for t ∈ T s.t. H[t] = 0̄ do
    for c ∈ Σ(T, ◇), c' ∈ C do
        if H[c'[[c[t]]]] ≠ 0̄ then
            C ← C ∪ {c[[c']]}
for t_1, t_2 ∈ T s.t. t_1 ⋉_H^α t_2 do
    for c ∈ Σ(T, ◇), c' ∈ C do
        if H[c'][c[[t_1]]] ≠ αH[c'][c[[t_2]]] then
            C ← C ∪ {c[[c']]}
```

Algorithm 4 *Complete*(T, C, H, B, S).

```
T ← T ∪ S
while (T, C, H, B) is not closed or not consistent do
    Close(T, C, H, B)
    Consistent(T, C, H, B)
```

Given a set of trees T we use $\Sigma(T)$ for the set of trees $\{\sigma(t_1, \ldots, t_k) \mid \exists \Sigma_k \in \Sigma,\ \sigma \in \Sigma_k,\ t_i \in T,\ \forall 1 \leq i \leq k\}$. The procedure *Close*$(T, C, H, B)$, Alg. 2, checks if $H[t][C]$ is co-linearly independent from T for some tree $t \in \Sigma(T)$. If so it adds t to both T and B and loops back until no such trees are found, in which case the table is termed *closed*.

We use $\Sigma(T, t)$ for the set of trees in $\Sigma(T)$ satisfying that one of the children is the tree t. We use $\Sigma(T, \diamond)$ for the set of contexts all of whose children are in T. An observation table (T, C, H, B) is said to be *zero-consistent* if for every tree $t \in T$ for which $H[t] = \overline{0}$ it holds that $H[c[[t']]] = \overline{0}$ for every $t' \in \Sigma(T, t)$ and $c \in C$. It is said to be *co-linear consistent* if for every $t_1, t_2 \in T$ s.t. $t_1 \ltimes_H^\alpha t_2$ and every context $c \in \Sigma(T, \diamond)$ we have that $c[[t_1]] \ltimes_H^\alpha c[[t_2]]$. The procedure *Consistent*, given in Alg. 3, looks for trees which violate the zero-consistency or co-linear consistency requirement, and for every violation, the respective context is added to C.

The procedure *Complete*(T, C, H, B, S), given in Alg. 4, first adds the trees in S to T, then runs procedures *Close* and *Consistent* iteratively until the table is both closed and consistent. When the table is closed and consistent the algorithm extracts from it a CMTA as detailed in Alg. 5.

Overall we can show that the algorithm always terminates, returning a correct CMTA whose dimension is minimal, namely it equals the rank n of Hankel matrix for the target language. It does so while asking at most n equivalence queries, and the number of membership queries is polynomial in n, and in the size of the largest counterexample m, but of course exponential in p, the highest rank of the a symbol in Σ. Hence for a grammar in Chomsky Normal Form, where $p = 2$, it is polynomial in all parameters.

Theorem 5.1. *Let* n *be the rank of the target language, let* m *be the size of the largest counterexample given by the teacher, and let* p *be the highest rank of a symbol in* Σ. *Then the algorithm makes at most* $n \cdot (n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^p)$ SMQ*s and at most* n SEQ*s.*

Algorithm 5 Extract CMTA

$d \leftarrow |B|$
for $0 \leq k \leq p$ **do**
  for $\sigma \in \Sigma_{(k)}$ **do**
    for $(i_1, i_2, \ldots, i_k) \in \{1, 2, \ldots, d\}^k$ **do**
      Let $t = \sigma(b_{i_1}, b_{i_2}, \ldots, b_{i_k})$
      if $H[t] = \bar{0}$ **then**
        for $1 \leq j \leq d$ **do**
          $\sigma^j_{i_1,i_2,\ldots,i_k} \leftarrow 0$
      else
        Let $b_i \in B, \alpha \in \mathbb{R}$ be s.t. $t_{i_1,i_2,\ldots,i_k} \ltimes^{\alpha}_{H} b_i$
        for $1 \leq j \leq d$ **do**
          if $j = i$ **then**
            $\sigma^j_{i_1,i_2,\ldots,i_k} \leftarrow \alpha$
          else
            $\sigma^j_{i_1,i_2,\ldots,i_k} \leftarrow 0$
    Let μ_σ be the $d \times d^k$ matrix obtained from the respective coefficients.
for $1 \leq j \leq d$ **do**
  $\lambda[j] \leftarrow H(\diamond, b_j)$
return $\mathcal{A} = (\Sigma, \mathbb{R}, d, \mu, \lambda)$

6 Demonstration

As a demonstration, we apply our algorithm to the learning of gene cluster grammars — which is an important problem in functional genomics. A *gene cluster* is a group of genes that are co-locally conserved, not necessarily in the same order, across many genomes (Winter et al. 2016). The gene grammar corresponding to a given gene cluster describes its hierarchical inner structure and the relations between instances of the cluster succinctly; assists in predicting the functional association between the genes in the cluster; provides insights into the evolutionary history of the cluster; aids in filtering meaningful from apparently meaningless clusters; and provides a natural and meaningful way of visualizing complex clusters.

PQ trees have been advocated as a representation for gene-grammars (Booth and Lueker 1976; Bergeron, Gingras, and Chauve 2008). A PQ-tree represents the possible permutations of a given sequence, and can be constructed in polynomial-time (Landau, Parida, and Weimann 2005). A PQ-tree is a rooted tree with three types of nodes: *P-nodes*, *Q-nodes* and leaves. In the gene grammar inferred by a given PQ-tree, the children of a P-node can appear in any order, while the children of a Q-node must appear in either left-to-right or right-to-left order.

However, the PQ tree model suffers from limited specificity, which often does not scale up to encompass gene clusters that exhibit some rare-occurring permutations. It also does not model tandem gene-duplications, which are a common event in the evolution of gene-clusters. We exemplify how our algorithm can learn a grammar that addresses both of these problems. Using the more general model of context-free grammar, we can model evolutionary events that PQ-trees cannot, such as tandem gene-duplications. While the probabilities in our PCFGs grant our approach the capability to model rare-occurring permutations (and weighing them as such), thus creating a specificity which PQ-trees lack.

$S \rightarrow$ R N7 [0.456] | R N3 [0.185] | C N1 [0.137] | N7 R [0.078] | R N6 [0.053] | N1 C [0.034] | N5 R [0.028] | N6 R [0.013] | N3 R [0.012] | R N5 [0.004]
$N1 \rightarrow$ R N2 [0.640] | N2 R [0.160] | R N4 [0.160] | N4 R [0.040]
$N2 \rightarrow$ A B [1.000] $N3 \rightarrow$ C N2 [1.000] $N4 \rightarrow$ B A [1.000]
$N5 \rightarrow$ N4 C [1.000] $N6 \rightarrow$ C N4 [0.882] | N8 A [0.095] | A N8 [0.024]
$N7 \rightarrow$ N2 C [1.000] $N8 \rightarrow$ B C [0.800] | C B [0.200]
$R \rightarrow$ AcrR [1.000] $A \rightarrow$ AcrA [1.000] $B \rightarrow$ AcrB [1.000]
$C \rightarrow$ TolC [1.000]

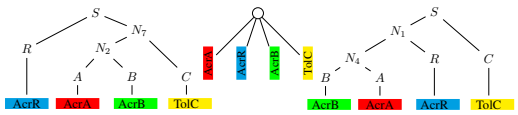

Figure 4: The PCFG learned from the MDR dataset. Also shown are the most probable tree according to the grammar (left) with $p = 0.456$, a non-probable tree (right) with $p = 0.001$, and the PQ-tree (middle) showing a complete lack of specificity.

In the extended version of this paper, we give two examples of gene-cluster grammars. The first is a PCFG describing a gene cluster corresponding to a multi-drug efflux pump (MDR). MDR's are used by some bacteria as a mechanism for antibiotic resistance, and hence are the focus of research aimed towards the development of new therapeutic strategies. In this example, we exemplify learning of a gene-cluster grammar which models distinctly ordered merge events of sub-clusters in the evolution of this pump. The resulting learned gene-cluster grammar is illustrated in Fig. 4. See the extended version for a biological interpretation of the learned grammar, associating the highly probable rules with possible evolutionary events that explain them.

Note that, in contrast to the highly specific PCFG learned by our algorithm (Fig. 4, top), the PQ-tree constructed for this gene cluster (Fig. 4, middle) places all leaves under a single P-node, resulting in a complete loss of specificity regarding conserved gene orders and hierarchical swaps — this PQ tree will accept all permutations of the four genes, without providing information about which permutations are more probable, as our learned PCFG does.

In a second example, we exemplify learning of a gene-cluster grammar which models tandem duplication events. The yielded grammar demonstrates learning of an infinite language, with exponentially decaying probabilities.

7 Discussion

We have presented an algorithm for learning structurally unambiguous PCFGs from a given black-box language model using structured membership and equivalence queries. To our knowledge this is the first algorithm provided for this question. A recent paper (Weiss, Goldberg, and Yahav 2019) advocates one can obtain an interpretable model of practically black-box models such as recurrent neural networks, using PDFA learning. The present work extends on this and offers obtaining intrepretable models also in cases where the studied object exhibits non-regular (yet context-free) behavior, as is the case, e.g. in Gene Cluster grammars.

References

Abe, N.; and Warmuth, M. K. 1992. On the Computational Complexity of Approximating Distributions by Probabilistic Automata. *Machine Learning* 9: 205–260.

Abney, S.; McAllester, D.; and Pereira, F. 1999. Relating probabilistic grammars and automata. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 542–549.

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75(2): 87–106.

Angluin, D. 1990. Negative Results for Equivalence Queries. *Machine Learning* 5: 121–150.

Angluin, D.; and Kharitonov, M. 1995. When Won't Membership Queries Help? *J. Comput. Syst. Sci.* 50(2): 336–355.

Baker, J. K. 1979. Trainable grammars for speech recognition. In Klatt, D. H.; and Wolf, J. J., eds., *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, 547–550.

Beimel, A.; Bergadano, F.; Bshouty, N. H.; Kushilevitz, E.; and Varricchio, S. 2000. Learning functions represented as multiplicity automata. *J. ACM* 47(3): 506–530. doi:10.1145/337244.337257. URL https://doi.org/10.1145/337244.337257.

Bergeron, A.; Gingras, Y.; and Chauve, C. 2008. Formal models of gene clusters. *Bioinformatics algorithms: techniques and applications* 8: 177–202.

Booth, K. S.; and Lueker, G. S. 1976. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences* 13(3): 335–379.

Chomsky, N. 1956. Three models for the description of language. *IRE Trans. Inf. Theory* 2(3): 113–124. doi:10.1109/TIT.1956.1056813. URL https://doi.org/10.1109/TIT.1956.1056813.

Church, K. W. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Second Conference on Applied Natural Language Processing*, 136–143. Austin, Texas, USA: Association for Computational Linguistics. doi:10.3115/974235.974260. URL https://www.aclweb.org/anthology/A88-1019.

Cohen, J. E.; and Rothblum, U. G. 1993. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra and its Applications* 190: 149–168.

de la Higuera, C. 2010. *Grammatical Inference: Learning Automata and Grammars*. USA: Cambridge University Press. ISBN 0521763169.

Drewes, F.; and Högberg, J. 2007. Query learning of regular tree languages: How to avoid dead states. *Theory of Computing Systems* 40(2): 163–185.

Gold, E. M. 1978. Complexity of Automaton Identification from Given Data. *Information and Control* 37(3): 302–320.

Grate, L. 1995. Automatic RNA Secondary Structure Determination with Stochastic Context-Free Grammars. In Rawlings, C. J.; Clark, D. A.; Altman, R. B.; Hunter, L.; Lengauer, T.; and Wodak, S. J., eds., *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, Cambridge, United Kingdom, July 16-19, 1995*, 136–144. AAAI. URL http://www.aaai.org/Library/ISMB/1995/ismb95-017.php.

Habrard, A.; and Oncina, J. 2006. Learning multiplicity tree automata. In *International Colloquium on Grammatical Inference*, 268–280. Springer.

Hopcroft, J. E.; and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company.

Landau, G. M.; Parida, L.; and Weimann, O. 2005. Gene proximity analysis across whole genomes via pq trees1. *Journal of Computational Biology* 12(10): 1289–1306.

Lari, K.; and Young, S. J. 1990. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language* 4: 35–56.

Levy, L. S.; and Joshi, A. K. 1978. Skeletal structural descriptions. *Information and Control* 39(2): 192 – 211. ISSN 0019-9958. doi:https://doi.org/10.1016/S0019-9958(78)90849-5. URL http://www.sciencedirect.com/science/article/pii/S0019995878908495.

Nitay, D.; Fisman, D.; and Ziv-Ukelson, M. 2021. Learning of Structurally Unambiguous Probabilistic Grammars (full version). ArXiv, 2011.07472.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE*, 257–286.

Regis, R. G. 2016. On the properties of positive spanning sets and positive bases. *Optimization and Engineering* 17(1): 229–262. ISSN 1573-2924. doi:10.1007/s11081-015-9286-x. URL https://doi.org/10.1007/s11081-015-9286-x.

Sakakibara, Y. 1988. Learning Context-Free Grammars from Structural Data in Polynomial Time. In *Proceedings of the First Annual Workshop on Computational Learning Theory, COLT '88, Cambridge, MA, USA, August 3-5, 1988*, 330–344. URL http://dl.acm.org/citation.cfm?id=93109.

Sakakibara, Y. 1992. Efficient learning of context-free grammars from positive structural examples. *Inform. Comput.* 97: 23–60.

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

Weiss, G.; Goldberg, Y.; and Yahav, E. 2019. Learning Deterministic Weighted Automata with Queries and Counterexamples. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 8558–8569. URL http://papers.nips.cc/paper/9062-learning-deterministic-weighted-automata-with-queries-and-counterexamples.

Winter, S.; Jahn, K.; Wehner, S.; Kuchenbecker, L.; Marz, M.; Stoye, J.; and Böcker, S. 2016. Finding approximate gene clusters with Gecko 3. *Nucleic Acids Research* 44(20): 9600–9610.