JMLR: Workshop and Conference Proceedings 57:42–53, 2016

Testing Distributional Properties of Context-Free Grammars

Alexander Clark ALEXSCLARK@GMAIL.COM
Department of Philosophy
King's College London

Abstract

Recent algorithms for distributional learning of context-free grammars can learn all languages defined by grammars that have certain distributional properties: the finite kernel property (FKP) and the finite context property (FCP). In this paper we present some algorithms for approximately determining whether a given grammar has one of these properties. We then present the results of some experiments that indicate that with randomly generated context-free grammars in Chomsky normal form, which generate infinite languages and are derivationally sparse, nearly all grammars have the finite kernel property, whereas the finite context property is much less common.

1. Introduction

Recent algorithms for distributional learning of context-free grammars can be proven to be correct if the grammars have certain properties or define languages which has some properties: substitutability, or the *k-Finite Context Property* or the *k-Finite Kernel Property* (Clark and Eyraud, 2007; Clark and Yoshinaka, 2016; Leiß, 2014). We will refer to these as *distributional properties*: for each positive natural number k we have a class of languages that is efficiently learnable from positive data and membership queries, giving a proper hierarchy of classes of grammars and languages as discussed by Yoshinaka (2012), which increases with k. We discuss this theory in Section 3.

It can be shown for example that all regular languages satisfy the 1-FCP and the 1-FKP. However, it is known that some simple grammars do not have these properties and that some context-free languages cannot be described by any grammar that has the k-FCP/FKP for any value of k. A simple example is the language O_1^c, which we define as the language over the two letter alphabet $\{a, b\}$ which consists of any strings where the number of as does not equal the number of bs.

But it is hard to get a theoretical grip on the question of what sorts of CFGs will have these properties. Informally, we want to know whether it is the case that nearly all grammars have the FCP/FKP and only a few carefully constructed counterexamples don't, or whether nearly all large random CFGs do not have these properties. Moreover we want to know what the smallest values of k are such that the grammars have the k-FCP/FKP. Are they typically small, like 1 or 2, or can they be very large? Moreover what are the relative merits of the primal and dual approaches? Is it the case that there are substantial differences in the applicability of these two techniques?

Additionally, when we move to empirical experimentation with learning algorithms, we want to know whether the conditions of the algorithms are likely to be met. Without this

knowledge we will not be able to interpret the empirical success or failure of the techniques. So getting some sort of idea of how common these properties are in the class of CFGs seems an important precursor to learning experiments.

These sorts of questions are intrinsically imprecise. We can get approximate answers through some computational simulations. Accordingly in this paper we use an empirical strategy: we look at random CFGs and test to see whether they have the relevant properties. There is no single natural distribution over CFGs: accordingly all of our conclusions must be taken relative to particular methods of generating CFGs which we discuss in Section 4; these can be thought of as prior distributions, though we are not using Bayesian methods. We want nontrivial grammars, and so we need to tune the hyperparameters of the priors in order to obtain grammars that define suitable languages: neither finite nor co-finite (Section 5).

Once we have generated a random CFG, we can then approximately test whether it has the relevant learnability properties. We describe some algorithms for the special case when $k = 1$ in Section 6, algorithms for the primal case in Section 7 and for the dual case in Section 8. The code to reproduce the experiments is publicly available at `https://github.com/alexc17/testcfg.git`. All experiments here can be completed in less than 24 hours computation on an AWS EC2 instance of type `c4.4xlarge`.

2. Formal Preliminaries

We assume a finite alphabet Σ; we write Σ^* for the set of all strings of Σ, and λ for the empty string, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. A language L is just a subset of Σ^*. Concatenation of languages L, M is written $L \cdot M$.

A context is an ordered pair of strings written $l\Box r$, where $l, r \in \Sigma^*$, and $\Box$ is a distinguished symbol not in Σ. Given a context $l\Box r$ and a string w we define the operation $\odot$ as $l\Box r \odot w = lwr$. We lift this to sets of strings and/or contexts in the obvious way.

Given a language L and a set of strings W we define an associated set of contexts $W^{\triangleright} = \{l\Box r \mid l\Box r \odot W \subseteq L\}$. Given a set of contexts Z, we define an associated set of strings: $Z^{\triangleleft} = \{w \mid Z \odot w \subseteq L\}$. $W^{\triangleright\triangleleft}$, a set of strings, is called the closure of W. Note that for any set of strings W, $W \subseteq W^{\triangleright\triangleleft} = (W^{\triangleright\triangleleft})^{\triangleright\triangleleft}$ and so $(\cdot)^{\triangleright\triangleleft}$ is a closure operator on sets of strings. Crucially if X, Y, Z are arbitrary sets of strings such that $X \supseteq Y \cdot Z$ then $X^{\triangleright\triangleleft} \supseteq Y^{\triangleright\triangleleft} \cdot Z^{\triangleright\triangleleft}$.

2.1. Context-free grammars

A context-free grammar (CFG) is a tuple $\langle \Sigma, V, S, P \rangle$ where Σ is a finite set of terminal symbols, V is a finite set of nonterminal symbols, $S \in V$ is a start symbol, P is a set of productions of the form $A \to \alpha$ where $A \in V$ and $\alpha \in (\Sigma \cup V)^*$. We will write nonterminals using uppercase letters at the start of the alphabet, $A, B, C, \ldots$ and terminals using lower case letters, $a, b, c, \ldots$.

A CFG is in Chomsky Normal Form (CNF)[1]if every production is of the form $A \to BC$ or $A \to a$, where $A, B, C \in V$ and $a \in \Sigma$. We write the derivation operation using $\overset{*}{\Rightarrow}$. The

1. Strictly speaking CNF requires some additional conditions.

language generated by a nonterminal is $\mathcal{L}(G, A) = \{w \in \Sigma^* \mid A \stackrel{*}{\Rightarrow} w\}$, and the language defined by the grammar is $\mathcal{L}(G) = \mathcal{L}(G, S)$.

We define $\mathcal{C}(G, A) = \{l \Box r \mid S \stackrel{*}{\Rightarrow}_G lAr\}$. Note that $\mathcal{C}(G, A) \odot \mathcal{L}(G, A) \subseteq \mathcal{L}(G)$: this is the context-free property of CFGs.

A language $L \subseteq \Sigma^*$ is a context-free language if there is a CFG G such that $L = \mathcal{L}(G)$.

We say that a grammar is *trim* iff for every nonterminal A there are strings $l, u, r \in \Sigma^*$ such that $A \stackrel{*}{\Rightarrow} u$ and $S \stackrel{*}{\Rightarrow} lAr$.

A grammar has a unary cycle if there are distinct nonterminals such that $A \stackrel{*}{\Rightarrow} B$. A lexical rule is one $A \to \alpha$, where $\alpha \in \Sigma^+$.

2.2. Uniform sampling

Sampling from the language generated by a CFG is important, in several places; we also need to be able to sample from $\mathcal{L}(G, A)$ and from $\mathcal{C}(G, A)$ for a nonterminal A. Since we are not using probabilistic techniques we need an alternative model. We use what we call uniform sampling, which is a straighforward application of the techniques developed in Hickey and Cohen (1983); Flajolet et al. (1994) for unambiguous CFGs.

We assume here that each string has only finitely many distinct parses. This requires excluding grammars which allow unary cycles:derivations of the form $A \stackrel{+}{\Rightarrow} A$. Grammars that contain such derivations will in general have an infinite degree of ambiguity.

We use a dynamic programming algorithm to compute the number of distinct derivations from each nonterminal that give rise to strings of a given length n. We do this for all lengths up to some maximal length: typically 50. Given some desired length, we can then sample uniformly from all derivations that give rise to strings of that length. Note that this is not quite the same as sampling from all strings of a given length, since the probability of a string is weighted by its degree of ambiguity. The values of these derivation counts are also useful for computing the density of the grammar as we shall see below. We use the same method to sample from the contexts of a nonterminal.

2.3. Intersection with a finite automaton.

We also rely heavily on a method of intersecting a CFG with a non-deterministic finite automaton (NFA). Here we use a variant of the standard Bar-Hillel construction. This construction involves creating for every nonterminal A in the original grammar and every pair of states q, q' in the automaton a new nonterminal $A_{q,q'}$ which generates the strings generated by A and which are accepted by a path from q to q'.

We can then combine this method with the uniform sampling method: in particular we can sample from all strings that contain a given substring w, by intersecting the CFG with an NFA that defines the language $\Sigma^* w \Sigma^*$, and then using uniform sampling to generate strings from the intersected grammar.

The naive alternative is to sample from the original CFG and discard all strings that do not contain w. This is of course inefficient, and may be impractical if w is long. Similarly we can sample from all strings that have the context $l \Box r$, by using an NFA that represents $l\Sigma^* r$.

3. Distributional learning

Distributional learning techniques rely on modeling the relation between the derivation contexts and yields of a grammar. Languages will be learnable if there is a grammar where the nonterminals of the grammar can be *characterised* by a small set of strings or contexts.

First of all, when we think of a set of strings associated with a nonterminal, there are three distinct sets of strings. We can see that

$$\mathcal{C}(G,A)^{\triangleleft} \supseteq \mathcal{L}(G,A)^{\triangleright\triangleleft} \supseteq \mathcal{L}(G,A) \tag{1}$$

Informally the first is the set of all strings that can occur in the contexts of a nonterminal. The second is the closure of the set of strings generated by the nonterminal, and the last is the familiar set of strings generated by the nonterminal.

Example 1 *Consider the grammar $S \to a_1Sb, S \to a_2Sb, S \to \lambda$. We add the productions $S \to Ab, A \to a_1$. Consider the nonterminal A. Here we have all three sets being different: $\mathcal{C}(G,A)^{\triangleleft} = \{\{a_1,a_2\}^{n+1}b^n \mid n \geq 0\}$, $\mathcal{L}(G,A)^{\triangleright\triangleleft} = \{a_1,a_2\}$ and $\mathcal{L}(G,A) = \{a_1\}$.*

Definition 1 *Given a nonterminal A in a* CFG *G we say that*

- *a finite set of contexts Z characterises A if $\mathcal{L}(G,A)^{\triangleright\triangleleft} = Z^{\triangleleft}$, and*
- *a finite set of strings W characterises A if $\mathcal{L}(G,A)^{\triangleright\triangleleft} = W^{\triangleright\triangleleft}$.*

Grammars have the distributional properties if every nonterminal in the grammar is characterised by a small set of strings or contexts.

Definition 2 *A* CFG *G has the k-*FCP *if for every nonterminal A, there is a finite set of contexts Z_A, $|Z_A| \leq k$, which characterises A.*

Definition 3 *A* CFG *G has the k-*FKP *if for every nonterminal A, there is a finite set of strings W_A, $|W_A| \leq k$, which characterises A.*

There are algorithms that can learn all CFGs with the k-FCP or the k-FKP efficiently using membership queries. We will not describe the details of these algorithms here. Informally these are grammars where the nonterminals are *obvious* and clearly visible in the data. The algorithm can select k-tuples of observed strings or contexts, and construct a grammar using these as nonterminals, while using membership queries to eliminate incorrect rules.

For every regular language there is a context-free grammar with the 1-FKP that generates it, and likewise a context-free grammar with the 1-FCP (Yoshinaka, 2012).

Example 2 *Consider the language $L = \{a^nb^n \mid n > 0\} \cup \{a^nb^{2n} \mid n > 0\}$, defined by the grammar with $V = \{S,O,T\}$ and $P = \{S \to O, S \to T, O \to ab, O \to aOb, T \to abb, T \to aTbb\}$. We can see that this grammar has the 2-*FKP *but not the 1-*FKP*, and the 2-*FCP *but not the 1-*FCP*.*

Starting with the primal approach we can see that neither ab, nor any other single yield of O will characterise O, since $ab^{\triangleright} = \{a^n\Box b^n \mid n \geq 0\} \cup \{a^n\Box b^{2n+1} \mid n \geq 0\}$. However $\{ab, aabb\}^{\triangleright} = \mathcal{C}(G,O)$ and similarly $\{abb, aabbbb\}^{\triangleright} = \mathcal{C}(G,T)$. S can be characterised by $\{ab, abb\}$.

Dually $\{a\Box b\} \in \mathcal{C}(G,O)$ but $\{a\Box b\}^{\triangleleft} = \{a^nb^n \mid n \geq 0\} \cup \{a^nb^{2n+1} \mid n \geq 0\}$. However, $\{\Box, a\Box b\}^{\triangleleft} = \mathcal{L}(G,O)$. Similarly, $\{\Box, a\Box bb\}^{\triangleleft} = \mathcal{L}(G,T)$, and S is of course characterised by the single context $\Box$.

3.1. Computational Testing

We will present the precise testing algorithms later on, but they are built on various primitive operations that we discuss here. Computationally if we have a string w, we can easily see whether $w \in \mathcal{L}(G, A)$ by using a modified parser for G. Similarly, we can directly test whether a context $l\Box r$ is in $\mathcal{C}(G, A)$ using a parser. We can approximately test whether $w \in \mathcal{C}(G, A)^{\triangleleft}$ by sampling contexts from $\mathcal{C}(G, A)$ and seeing if w can occur in them using a parser. If it occurs in all of the contexts in a large sample, we can be reasonably confident that $\mathcal{C}(G, A) \odot \{w\} \subseteq L$, i.e. that $w \in \mathcal{C}(G, A)^{\triangleleft}$. Checking whether a string is in the closure of $\mathcal{L}(G, A)$ is more involved and requires two sampling steps and it seems that it cannot be carried out accurately.

In the case of testing whether a grammar has the k-FCP/FKP, there are several steps. First we want to test for every nonterminal whether that nonterminal has a small characterising set of strings or contexts. In order to do that, for each nonterminal we need to be able to search for some suitable set of strings or contexts; and then given such a set we need to test whether that set does in fact characterise the nonterminal in some appropriate sense.

4. Methodology: generating CFGs

We want to generate random instances of CFGs, and test whether they have these distributional properties. There is no natural distribution over CFGs; any distribution we pick will be to a certain extent arbitrary and must reflect some particular modeling goals. In the current work we are primarily interested in natural language syntax. Natural languages have some particular properties: one of their defining properties is often thought to be that they are infinite (Hauser et al., 2002; Kornai, 2014) and they are also sparse in the sense that the proportion of grammatical strings decreases as the length increases. Generating arbitrary CFGs is rather complex as it requires many different parameters to control the distribution of lengths of the right hand sides of productions and the proportions of terminals and nonterminals. Accordingly we restrict ourselves in this paper to a restricted normal form.

4.1. Chomsky Normal Form

We use a variant of Chomsky normal form, where all the productions of the grammar are of the form $A \to a$, or $A \to BC$, where $A, B, C \in V$ and $a \in \Sigma$.

Given this normal form, Σ and V, there are clearly a finite number of possible CFGs: indeed ignoring permutations of V, there are $2^{|V|^3+|\Sigma||V|}$ such grammars. Sampling uniformly from this set will give, with probability very close to 1 for large V, a grammar where nearly all the strings are in the language: see Figure 1. We therefore need some more refined method of producing these grammars.

We will denote the set of lexical productions by P_L and the set of binary productions by P_B; clearly $P = P_L \cup P_B$ in this case. We define as parameters $|V|, |\Sigma|, |P_B|$ and $|P_L|$. Given these parameters we sample uniformly without replacement from the appropriate sets of productions. Note that if $|\Sigma| > |P_L|$, not every element of Σ will be used in the grammar,

so the effective alphabet size may be smaller. Similarly, the grammar may not be trim and thus the effective nonterminal size may also be smaller.

5. Preliminary experiments

We want to tune the (hyper)-parameters to get reasonable grammars. Consider a grammar with fixed V and Σ. If we start off with no productions, the grammar will generate no strings. As we increase the number of productions, then the language generated will increase, and eventually will generate Σ^+. Initially the grammar may only define a finite language; in the end it will define a cofinite one. Neither of these possibilities is interesting, we therefore want to carefully control the number of productions then: neither too many nor too few. Note that a grammar must have at least $|V|$ productions if it is trim. We can test efficiently whether a grammar generates an infinite language by first making it trim and then looking for cycles in the reachability graph using Tarjan's algorithm (Tarjan, 1972).

We want to define an appropriate notion of sparsity. We can compute for any length of string n, the number of derivations that will give rise to a string of length n; we write this as D_n. We can then compare this to the number of possible strings of length n, which is $|\Sigma|^n$. The ratio of these two quantities, $D_n/|\Sigma|^n$, is the derivational density of the grammar. If this density is less than 1, for some value of n, then we know that the grammar does not generate Σ^n. This density can be thought of as the mean degree of ambiguity, taken over all strings, not just those generated by the grammar. In order to test this separately from the degree of lexical ambiguity we use a trivial lexicon – we take $|\Sigma| = |V|$ and have exactly one unambiguous lexical production for each nonterminal.

We can use a more accurate but less efficient technique to estimate the actual proportion of strings in the grammar of length n; namely the string density which is $|\mathcal{L}(G) \cap \Sigma^n|/|\Sigma|^n$. Note that we can compute the number of derivations of a string w, which we denote $D(w)$, using an Earley parser which outputs a shared packed parse forest. In order to get a Monte Carlo estimate of the string density, we sample N derivations uniformly from derivations that give rise to strings of length n; for each derivation we extract the yield w_i, compute $D(w)$, and use the approximation:

$$\frac{|\mathcal{L}(G) \cap \Sigma^n|}{|\Sigma|^n} \approx \frac{D_n}{|\Sigma|^n} \cdot \frac{1}{N} \sum_{i \in \{1,\ldots,N\}} \frac{1}{D(w_i)}$$

We can see that this is an unbiased estimator. The naive method of sampling a string w uniformly from Σ^n and testing if $w \in \mathcal{L}(G)$ will be inefficient for cases where the density is exponentially small.

Looking at the graphs in Figure 1, we can see that they are broadly straight lines after some preliminary oscillations, as we would expect. So we can measure whether they are dense or not simply by picking a length $l = 20$, and testing whether $D_l < |\Sigma|^l$. Figure 2 shows the proportions of grammars that generate an infinite language and are also derivationally sparse in this sense, as the number of binary productions increases.

For efficiency purposes we don't want to use grammars which are too large so we restrict ourselves to $|V| = 10$, and take $|P_B| = 30$.

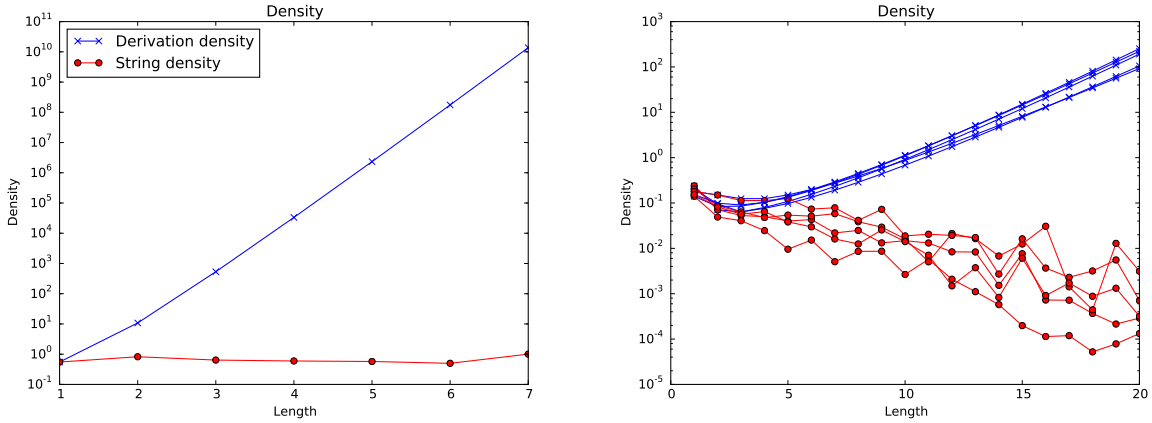

Figure 1: In both diagrams we have $|V| = 10$, $|\Sigma| = 100$, and we plot the derivational density and string density. On the left we have a single grammar with $|P_B| = 500, |P_L| = 500$, which approximates one sampled uniformly from all CFGs. On the right, we have 5 grammars sampled with $|P_B| = 30, |P_L| = 100$. Note that on the left the string density is constant and close to 1, and on the right it decreases exponentially.

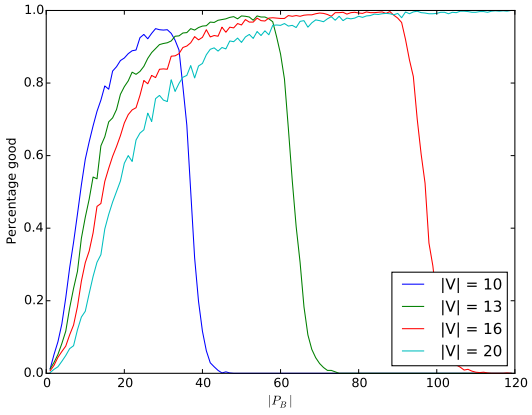

Figure 2: Graph showing which grammars define infinite languages and derivationally sparse (derivational density at length 10 is less than 1); in this experiment we use a trivial lexicon with one unambiguous lexical production per nonterminal.

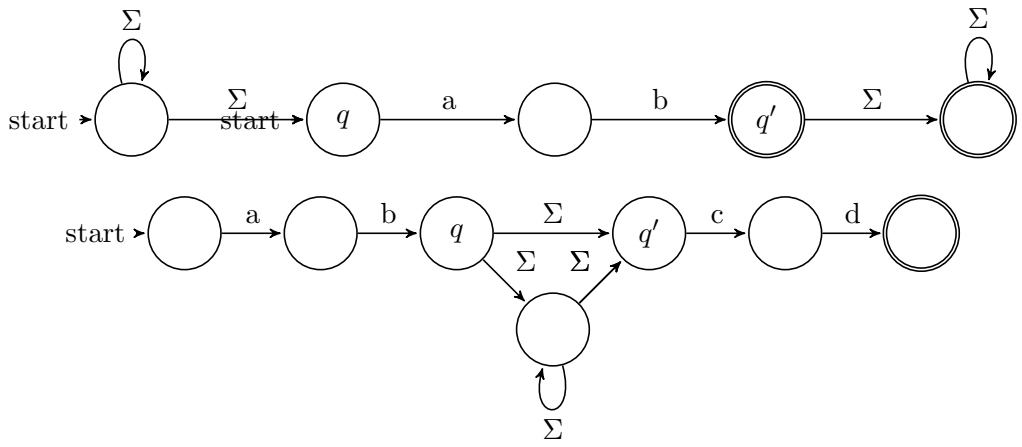

Figure 3: On the top an automaton for the string ab, with two initial states and two accepting states. Transitions labeled Σ accept all elements of the alphabet. A path from q to q' generates exactly ab. On the bottom an automaton for the context (ab, cd), with only one initial state and accepting state. A path from q to q' only occurs in the context (ab, cd). Note that this automaton does not accepted $abcd$.

6. Exact algorithms for 1-FKP and 1-FCP

We will start by considering the special case when $k = 1$, which admit algorithms which will not make false positives; we will call these algorithms *exact* algorithms though they do involve some approximations.

In the case of the 1-FKP, we are looking for a string which characterises the yields of the nonterminal. So given a nonterminal A and a string w we want to test whether $\mathcal{L}(G, A)^{\triangleright} = \{w\}^{\triangleright}$. This is hard to do. But we can easily test a stronger condition namely whether every occurrence of w is a yield of A. More precisely whether every derivation of a string lwr is a derivation of the form $S \stackrel{*}{\Rightarrow} lAr \stackrel{*}{\Rightarrow} w$. We can test this exactly, by constructing a NFA, that accepts $\Sigma^* w \Sigma^*$ and contains a pair of states q, q' such that the only string accepted on the path from q to q' is the string w. See Figure 3 for an example. We then intersect this with the CFG, and remove the nonterminal $A_{q,q'}$ from the grammar, and test whether the resulting grammar is empty. If the grammar is empty (i.e. fails to generate any strings), then we know that the grammar has the 1-FKP for A. If it is non-empty, this may be because w does not characterise A, or it might be that it does, but that all of the other contexts in which w occurs are also contexts of A. This is possible,[2] but seems not to occur in practice.

We can use an exactly similar technique for the dual approach, where we want to test whether a nonterminal is characterised by the context $l \Box r$ by intersecting the CFG with a NFA that recognises the language $l\Sigma^+ r$ as shown in the second part of Figure 3.

2. An example: $S \to aSb, S \to ab, S \to aabb$. Here ab does characterise S, but ab also occurs as a substring of $S \to aabb$ and thus it will fail this test.

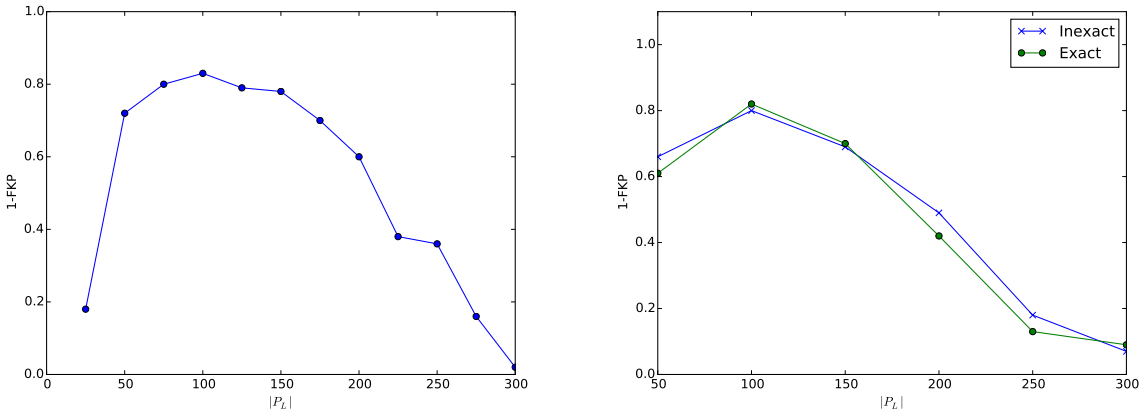

Figure 4: Graph showing how the 1-FKP depends on the number of lexical rules: $|V| = 10, |P_B| = 30, |\Sigma| = 100$, 100 grammars sampled for each data point. On the right we have a comparison of the exact and inexact methods for the 1-FKP.

In both cases we need to look for appropriate yields and contexts. We do this by fixing some number n and sampling n distinct strings or contexts from either $\mathcal{L}(G, A)$ or $\mathcal{C}(G, A)$ considering first the shortest strings or contexts. In preliminary experiments we did not find much sensitivity to the parameter n between the values of 10 to 1000. Moreover, in the context of the question we are investigating, if a nonterminal has only very long characterising strings or contexts, then it may be impractical for a learning algorithm to find it. Thus the restriction to using a small number of samples seems reasonable here.

6.1. Results

For the 1-FCP, essentially no grammars have this property. Each individual context defines far too large a set of strings. This is partly to do with the distribution over grammars that we have. Suppose we have a production $S \to AB$, and $A \to a$. Then one might think that the context $a\Box$ would define B. But if we have a reasonably complex grammar, then it is likely that we may have other derivations, for example $A \stackrel{*}{\Rightarrow} A\alpha$ for some $\alpha \in V^+$. This means that $S \stackrel{*}{\Rightarrow} a\alpha B$ and thus the context $a\Box$ will not be characteristic of B. Alternatively there may be some other nonterminal C with a productions $S \to AC$. Moreover, in many cases a nonterminal A may not appear on the right hand side of a production with S on the left hand side, and thus the possible contexts may all be quite long. We won't present results for the exact 1-FCP.

The 1-FKP crucially depends on the degree of lexical ambiguity of the grammar. Figure 4 shows a graph plotting how the number of lexical productions affects the proportion of the grammars that have the 1-FKP. This effect is almost entirely due to strings of length 1: that is to say, when we have a production $A \to a$, and the letter a occurs only in this production, then the string a is a 1-FKP for A.

7. Primal testing

We now consider testing the k-FKP for $k > 1$. In this case, the approximation algorithms can make both false positive and false negative errors – we will call these *inexact* algorithms.

Suppose we have some finite set of strings $K = \{w_1, \ldots, w_k\}$ and we want to check whether they characterise a nonterminal A. We assume that $K \subseteq \mathcal{L}(G, A)$, which means that $K^{\triangleright} \supseteq \mathcal{L}(G, A)^{\triangleright} \supseteq \mathcal{C}(G, A)$. The set K can fail to characterise A if it defines too small a set of strings; in other words if $K^{\triangleright} \supset \mathcal{L}(G, A)^{\triangleright}$. This is hard to test directly so we will test instead whether $K^{\triangleright} = \mathcal{C}(G, A)$, which is a stronger condition than needed. We start by intersecting the grammar with an NFA which generates $\Sigma^* w_1 \Sigma^*$; this gives us a grammar which generates strings that contain w_1. This allows us to sample contexts from $\{w_1\}^{\triangleright}$, by sampling strings from the grammar and extracting the context or contexts of w_1. For each context $l\Box r$ we see if it is a context of $\{w_2, \ldots, w_k\}$, by testing whether $lw_i r \in L$ with a parser. If it is, then we know that $l\Box r \in K^{\triangleright}$ and then we test to see if it is in $\mathcal{C}(G, A)$. If we find some context $l\Box r$ which is in $K^{\triangleright}$ but is not in $\mathcal{C}(G, A)$, then we know that $K^{\triangleright} \neq \mathcal{C}(G, A)$, and therefore that K does not characterise A. If on the other hand we do not find such a context, we might be wrong.

In order to check whether the grammar has the k-FKP, for each nonterminal we select 10 short strings from $\mathcal{L}(G, A)$ and then select 50 k-tuples from this set. We compare the inexact method for the 1-FKP with the exact method presented in Section 6 as a reality check; see the right hand side of Figure 4. We can see an excellent match between these two methods.

8. Dual testing

The key part when testing the FCP is to decide whether a nonterminal A is characterised by a finite set of contexts $F = \{l_1\Box r_1, \ldots, l_k\Box r_k\}$. We assume that $F \subseteq \mathcal{C}(G, A)$, in other words that the contexts we use are in fact contexts of the desired nonterminal. We will test, approximately, whether $F^{\triangleleft} = \mathcal{L}(G, A)$.

We will do this by sampling from $F^{\triangleleft}$, and testing whether every string is in $\mathcal{L}(G, A)$. We already know that $F^{\triangleleft} \supseteq \mathcal{L}(G, A)$ since $F \subseteq \mathcal{C}(G, A)$ implies that $F^{\triangleleft} \supseteq \mathcal{C}(G, A)^{\triangleleft} \supseteq \mathcal{L}(G, A)$. So we pick one context from F, say $l_1\Box r_1$ and intersect G with a NFA that generates $l_1\Sigma^* r_1$, to get a grammar $G_{l_1\Box r_1}$. We then sample from $G_{l_1\Box r_1}$ to get strings which will be of the form $l_1 w r_1$. We remove the context, and then test whether $l_i w r_i \in L$ using a parser, for each $i \in \{2, \ldots, k\}$. If it passes these tests, then $w \in F^{\triangleleft}$.

We then test whether $w \in \mathcal{L}(G, A)$; if we find that this is not the case then we know that it is not the case that $F^{\triangleleft} \subseteq \mathcal{L}(G, A)$, and so therefore $F^{\triangleleft} \supset \mathcal{L}(G, A)$. Of course, since we are only sampling a small finite number of strings, 100 in this case, we may get a false positive result, if the relevant counterexamples are rare.

In order to find suitable sets of contexts, we use a naive heuristic: we simply sample short contexts from $\mathcal{C}(G, A)$ using uniform sampling and then sample k-tuples from this set with replacement. We sampled 25 contexts and then tried 100 k-tuples.[3] These are slightly higher values than we used for the primal methods, as the initial performance was poor. See the right hand side of Figure 5. We can see that the results are generally poor

3. When $k = 1$ this is redundant.

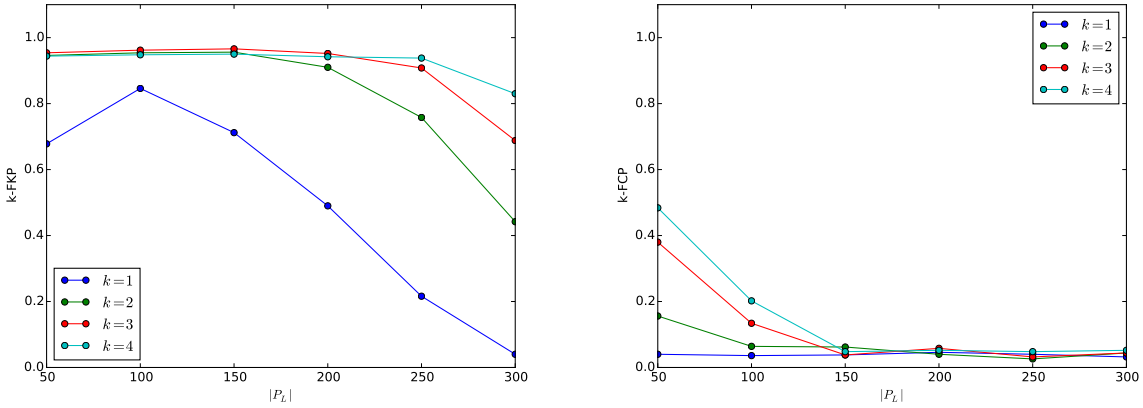

Figure 5: On the left we have the k-FKP and on the right the k-FCP for $k \in \{1, 2, 3, 4\}$ showing dependency of lexical ambiguity and value of k. This is with 500 grammars per data point, $|\Sigma| = 100, |V| = 10, |P_B| = 30$.

even for values of $k = 4$. Moreover, the performance drops sharply as the lexical ambiguity increases.

9. Conclusion

We have presented some algorithms for approximately deciding whether grammars have certain distributional properties that allow them to be learned. This allows us to study large numbers of grammars to see whether they are learnable. The learnability analysis however still relies on some implausible assumptions, and so the next step is to see what additional constraints we need on probabilistic CFGs to see whether they are learnable under a more realistic setting: namely from stochastic positive examples (Clark, 2006; Shibata and Yoshinaka, 2013).

The experiments here are clearly not definitive, but we can draw some preliminary conclusions. Nearly all CFGs have the k-FKP for small values of k, it seems. The value of k depends crucially on the degree of lexical ambiguity. The comparatively poor results for the dual methods are perhaps an artifact of the use of CNF. The existence of productions like $S \to aB$ for example could give a distinguishing context for B; indeed in preliminary experiments with general CFGs we do observe better results for the k-FCP. Similarly the positive results for the primal methods are due to the fact that in the CNF model, it is highly likely that we will have for every nonterminal a length one string derived from that nonterminal; this is not in general the case, particularly for CFGs that are the result of binarising a grammar which contains productions with more than two nonterminals on the right hand side.

References

Alexander Clark. PAC-learning unambiguous NTS languages. In Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita, editors, *Grammatical Inference: Algorithms and Applications*, pages 59–71. Springer Berlin Heidelberg, 2006.

Alexander Clark and Rémi Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745, August 2007.

Alexander Clark and Ryo Yoshinaka. Distributional learning of context-free and multiple context-free grammars. In Jeffrey Heinz and M. José Sempere, editors, *Topics in Grammatical Inference*, pages 143–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

Philippe Flajolet, Paul Zimmermann, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1): 1–35, 1994.

Marc D. Hauser, Noam Chomsky, and W. Tecumseh Fitch. The faculty of language: What is it, who has it, and how did it evolve? *Science*, 298:1569–1579, 11 2002.

T. Hickey and J. Cohen. Uniform random generation of strings in a context-free language. *SIAM J. COMPUT.*, 12(4):645–655, 1983.

András Kornai. Resolving the infinitude controversy. *Journal of Logic, Language and Information*, 23(4):481–492, 2014.

Hans Leiß. Learning context free grammars with the finite context property: A correction of A.Clark's algorithm. In Glyn Morrill, Reinhard Muskens, Rainer Osswald, and Frank Richter, editors, *Formal Grammar*, pages 121–137. Springer Berlin Heidelberg, 2014.

Chihiro Shibata and Ryo Yoshinaka. PAC learning of some subclasses of context-free grammars with basic distributional properties from positive data. In Sanjay Jain, Rmi Munos, Frank Stephan, and Thomas Zeugmann, editors, *Algorithmic Learning Theory*, pages 143–157. Springer Berlin Heidelberg, 2013.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

Ryo Yoshinaka. Integration of the dual approaches in the distributional learning of context-free grammars. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 538–550. Springer Berlin Heidelberg, 2012.