Journal of Machine Learning Research 11 (2010) 3053-3096 Submitted 5/10; Revised 9/10; Published 11/10

Inducing Tree-Substitution Grammars

Trevor Cohn TCOHN@DCS.SHEF.AC.UK
Department of Computer Science
University of Sheffield
Sheffield S1 4DP, UK

Phil Blunsom PBLUNSOM@COMLAB.OX.AC.UK
Computing Laboratory
University of Oxford
Oxford OX1 3QD, UK

Sharon Goldwater SGWATER@INF.ED.AC.UK
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, UK

Editor: Dorota Glowacka

Abstract

Inducing a grammar from text has proven to be a notoriously challenging learning task despite decades of research. The primary reason for its difficulty is that in order to induce plausible grammars, the underlying model must be capable of representing the intricacies of language while also ensuring that it can be readily learned from data. The majority of existing work on grammar induction has favoured model simplicity (and thus learnability) over representational capacity by using context free grammars and first order dependency grammars, which are not sufficiently expressive to model many common linguistic constructions. We propose a novel compromise by inferring a probabilistic *tree substitution grammar*, a formalism which allows for arbitrarily large tree fragments and thereby better represent complex linguistic structures. To limit the model's complexity we employ a Bayesian non-parametric prior which biases the model towards a sparse grammar with shallow productions. We demonstrate the model's efficacy on supervised phrase-structure parsing, where we induce a latent segmentation of the training treebank, and on unsupervised dependency grammar induction. In both cases the model uncovers interesting latent linguistic structures while producing competitive results.

Keywords: grammar induction, tree substitution grammar, Bayesian non-parametrics, Pitman-Yor process, Chinese restaurant process

1. Introduction

Inducing a grammar from a corpus of strings is one of the central challenges of computational linguistics, as well as one of its most difficult. Statistical approaches circumvent the theoretical problems with learnability that arise with non-statistical grammar learning (Gold, 1967), and performance has improved considerably since the early statistical work of Merialdo (1994) and Carroll and Charniak (1992), but the problem remains largely unsolved. Perhaps due to the difficulty of this unsupervised grammar induction problem, a more recent

line of work has focused on a somewhat easier problem, where the input consists of a treebank corpus, usually in phrase-structure format, and the task is to induce a grammar from the treebank that yields better parsing performance than the basic maximum-likelihood probabilistic context free grammar (PCFG). Examples of work on this kind of grammar induction, which we will refer to as *grammar refinement* because the learned grammars can be viewed as refinements of the treebank PCFG, include the symbol-splitting approach of Petrov and Klein (2007) and the tree-substitution grammars of Data-Oriented Parsing (Bod et al., 2003; Bod, 2003). Although the grammars induced by these systems are latent, the resulting parsers are supervised in the sense that the input to the learning system consists of strings and parses, and the goal is to learn how to parse new strings. Consequently, these systems do not remove the necessity of hand-annotating a large corpus, but they can potentially reduce the amount of engineering effort required to develop a successful statistical parser for a new language or domain, by obviating the need for complex hand-engineering of features, independence assumptions, and backoff schemes.

The desire for automatic grammar refinement methods highlights one possible reason why unsupervised grammar induction is so difficult. Simple models of syntactic structure such as hidden Markov models (HMMs) or PCFGs make strong independence assumptions that fail to capture the true complexity of language, so these models tend to learn something other than the desired structure when used in an unsupervised way. On the other hand, more complex models with, for example, head-lexicalized rules have too many free parameters to be learned successfully from unannotated data without the use of sophisticated backoff schemes. Thus, finding the right balance between learnability and complexity is critical to developing a successful model of grammar induction. We posit that this balance can be achieved by using a rich grammatical formalism coupled with a nonparametric Bayesian prior to limit the model's complexity. In this way the model can learn sufficiently complex structure to model the data, but will only do so when there is enough evidence to support such complexity; otherwise it will generalise to simpler structures. The model can therefore learn plausible structures from either small or large training samples.

We present here a model for automatically learning a *Probabilistic Tree Substitution Grammar* (PTSG) from either a treebank or strings. A PTSG is an extension to the PCFG in which nonterminals can rewrite as entire tree fragments (*elementary trees*), not just immediate children (Joshi, 2003; Bod et al., 2003). These large fragments can be used to encode non-local context, such as argument frames, gender agreement and idioms. A fundamental problem with PTSGs is that they are difficult to estimate, even in the supervised (grammar refinement) scenario where treebanked data is available. This is because treebanks are typically not annotated with their TSG derivations—how to decompose a tree into elementary tree fragments—instead the derivation needs to be inferred.

Probably the best-known previous work on inducing PTSGs is within the framework of Data-Oriented Parsing (DOP; Bod et al., 2003), which, like our model, has been applied in both supervised and unsupervised settings (Bod, 2003; Prescher et al., 2004; Zollmann and Sima'an, 2005; Zuidema, 2007; Bod, 2006).[1] DOP seeks to use as TSG productions all subtrees of the training corpus, an approach which makes parameter estimation difficult and led to serious problems with early estimation methods (Johnson, 2002), namely inconsistency

1. Tree adjoining grammar induction (Chiang and Bikel, 2002; Xia, 2002) tackles a similar learning problem in the supervised case.

for DOP1 (Bod, 2003) and overfitting of the maximum likelihood estimate (Prescher et al., 2004). More recent work on DOP estimation has tackled these problems, drawing from estimation theory to solve the consistency problem (Prescher et al., 2004; Zollmann and Sima'an, 2005), or using a grammar brevity heuristic to avoid the degeneracy of the MLE (Zuidema, 2007). Our work differs from DOP in that we use an explicit generative model of TSG and a Bayesian prior for regularisation. The prior is nonparametric, which allows the model to learn a grammar of the appropriate complexity for the size of the training data. A further difference is that instead of seeking to use all subtrees from the training data in the induced TSG, our prior explicitly biases against such behaviour, such that the model learns a relatively compact grammar.[2] A final minor difference is that, because our model is generative, it assigns non-zero probability to all possible subtrees, even those that were not observed in the training data. In practice, unobserved subtrees will have very small probabilities.

We apply our model to the two grammar induction problems discussed above:

- **Inducing a TSG from a treebank.** This regime is analogous to the case of supervised DOP, where we induce a PTSG from a corpus of parsed sentences, and use this PTSG to parse new sentences. We present results using two different inference methods, training on either a subset of WSJ or on the full treebank. We report performance of 84.7% when training on the full treebank, far better than the 64.2% for a PCFG parser. These gains in accuracy are obtained with a grammar that is somewhat larger than the PCFG grammar, but still much smaller than the DOP all-subtrees grammar.

- **Inducing a TSG from strings.** As in other recent unsupervised parsing work, we adopt a dependency grammar (Mel′čuk, 1988) framework for the unsupervised regime. We use the split-head construction (Eisner, 2000; Johnson, 2007) to map between dependency and phrase-structure grammars, and apply our model to strings of POS tags. We report performance of 65.9% on the standard $\text{WSJ}_{\leq 10}$ data set, which is statistically tied with the best reported result on the task and considerably better than the EM baseline which obtains 46.1%. When evaluated on test data with no restriction on sentence length—a more realistic setting—our approach significantly improves the state-of-the-art.

Our work displays some similarities to previous work on both the grammar refinement and unsupervised grammar induction problems, but also differs in a number of ways. Aside from DOP, which we have already discussed, most approaches to grammar refinement can be viewed as symbol-splitting. That is, they allow each nonterminal to be split into a number of subcategories. The most notable examples of the symbol-splitting approach include Petrov et al. (2006), who use a likelihood-based splitting and merging algorithm, and Liang et al. (2007) and Finkel et al. (2007), who develop nonparametric Bayesian models. In theory, any PTSG can be recast as a PCFG with a sufficiently large number of subcategories (one for each unique subtree), so the grammar space of our model is a subspace of the

2. The prior favours compact grammars by assigning the majority of probability mass to few productions, and very little (but non-zero) mass to other productions. In practice we use Markov Chain Monte Carlo sampling for inference which results in sparse counts with structural zeros, thus permitting efficient representation.

symbol-splitting grammars. However, the number of nonterminals required to recreate our PTSG grammars in a PCFG would be exorbitant. Consequently, our model should be better able to learn specific lexical patterns, such as full noun phrases and verbs with their subcategorisation frames, while theirs are better suited to learning subcategories with larger membership, such as the days of the week or count versus mass nouns. The approaches are largely orthogonal, and therefore we expect that a PTSG with nonterminal refinement could capture both types of concept in a single model, thereby improving performance over either approach alone.

For the unsupervised grammar induction problem we adopt the Dependency Model with Valency (DMV; Klein and Manning, 2004) framework that is currently dominant for grammar induction (Cohen et al., 2009; Cohen and Smith, 2009; Headden III et al., 2009; Cohen et al., 2010; Spitkovsky et al., 2010). The first grammar induction models to surpass a trivial baseline concentrated on the task of inducing unlabelled bracketings for strings and were evaluated against treebank bracketing gold standard (Clark, 2001; Klein and Manning, 2002). Subsequently the DMV model has proved more attractive to researchers, partly because it defined a well founded generative stochastic grammar, and partly due to the popularity of dependency trees in many natural language processing (NLP) tasks. Recent work on improving the original DMV model has focused on three avenues: smoothing the head-child distributions (Cohen et al., 2009; Cohen and Smith, 2009; Headden III et al., 2009), initialisation (Headden III et al., 2009; Spitkovsky et al., 2010), and extending the conditioning distributions (Headden III et al., 2009). Our work falls into the final category: by extending the DMV CFG model to a TSG we increase the conditioning context of head-child decisions within the model, allowing the grammar to directly represent groups of linked dependencies.

Adaptor Grammars (Johnson et al., 2007b) are another recent nonparametric Bayesian model for learning hierarchical structure from strings. They instantiate a more restricted class of tree-substitution grammar in which each subtree expands completely, with only terminal symbols as leaves. Since our model permits nonterminals as subtree leaves, it is more general than Adaptor Grammars. Adaptor Grammars have been applied successfully to induce labeled bracketings from strings in the domains of morphology and word segmentation (Johnson, 2008a,b; Johnson and Goldwater, 2009) and very recently for dependency grammar induction (Cohen et al., 2010). The latter work also introduced a variational inference algorithm for Adaptor Grammar inference; we use a sampling method here.

The most similar work to that presented here is our own previous work (Cohn et al., 2009; Cohn and Blunsom, 2010), in which we introduced a version of the model described here, along with two other papers that independently introduced similar models (Post and Gildea, 2009; O'Donnell et al., 2009). Cohn et al. (2009) and Post and Gildea (2009) both present models based on a Dirichlet process prior and provide results only for the problem of grammar refinement, whereas in this article we develop a newer version of our model using a Pitman-Yor process prior, and also show how it can be used for unsupervised learning. These extensions are also reported in our recent work on dependency grammar induction (Blunsom and Cohn, 2010), although in this paper we present a more thorough exposition of the model and experimental evaluation. O'Donnell et al. (2009) also use a Pitman-Yor process prior (although their model is slightly different from ours) and present unsupervised results, but their focus is on cognitive modeling rather than natural language processing,

so their results are mostly qualitative and include no evaluation of parsing performance on standard corpora.

To sum up, although previous work has included some aspects of what we present here, this article contains several novel contributions. Firstly we present a single generative model capable of both supervised and unsupervised learning, to induce tree substitution grammars from either trees or strings. We demonstrate that in both settings the model outperforms maximum likelihood baselines while also achieving results competitive with the best current systems. The second main contribution is to provide a thorough empirical evaluation in both settings, examining the effect of various conditions including data size, sampling method and parsing algorithm, and providing an analysis of the structures that were induced.

In the remainder of this article, we briefly review PTSGs in Section 2 before presenting our model, including versions for both constituency and dependency parsing, in Section 3. In Section 4 we introduce two different Markov Chain Monte Carlo (MCMC) methods for inference: a local Gibbs sampler and a blocked Metropolis-Hastings sampler. The local sampler is much simpler but is only applicable in the supervised setting, where the trees are observed, whereas the Metropolis-Hastings sampler can be used in both supervised and unsupervised settings and for parsing. We discuss how to use the trained model for parsing in Section 5, presenting three different parsing algorithms. Experimental results for supervised parsing are provided in Section 6, where we compare the different training and parsing methods. Unsupervised dependency grammar induction experiments are described in Section 7, and we conclude in Section 8.

2. Tree-substitution grammars

A *Tree Substitution Grammar*[3] (TSG) is a 4-tuple, $G = (T, N, S, R)$, where T is a set of *terminal symbols*, N is a set of *nonterminal symbols*, $S \in N$ is the distinguished *root nonterminal* and R is a set of productions (rules). The productions take the form of *elementary trees*—tree fragments[4] of height ≥ 1—where each internal node is labelled with a nonterminal and each leaf is labelled with either a terminal or a nonterminal. Nonterminal leaves are called *frontier nonterminals* and form the substitution (recursion) sites in the generative process of creating trees with the grammar. For example, in Figure 1b the S $\rightarrow$ NP (VP (V hates) NP) production rewrites the S nonterminal as the fragment (S NP (VP (V hates) NP)).[5] This production has the two NPs as its frontier nonterminals.

A *derivation* creates a tree by starting with the root symbol and rewriting (substituting) it with an elementary tree, then continuing to rewrite frontier nonterminals with elementary trees until there are no remaining frontier nonterminals. We can represent derivations as sequences of elementary trees $\mathbf{e}$, where each elementary tree is substituted for the left-most frontier nonterminal of the tree being generated. Unlike Context Free Grammars (CFGs) a

3. A TSG is a *Tree Adjoining Grammar* (TAG; Joshi, 2003) without the adjunction operator, which allows insertions at internal nodes in the tree. This operation allows TAGs to describe the set of mildly context sensitive languages. A TSG in contrast can only describe the set of context free languages.
4. Elementary trees of height 1 correspond to productions in a context free grammar.
5. We use bracketed notation to represent tree structures as linear strings. The parenthesis indicate the hierarchical structure, with the first argument denoting the node label and the following arguments denoting child trees. The nonterminals used in our examples denote nouns, verbs, etc., and their respective phrasal types, using a simplified version of the Penn treebank tag set (Marcus et al., 1993).

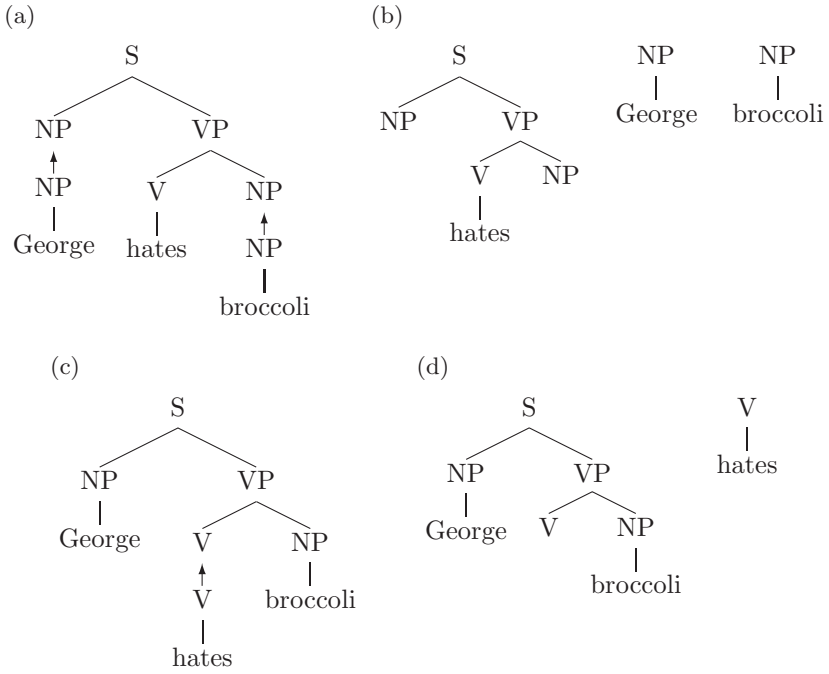


Figure 1: Example derivations for the same tree, where arrows indicate substitution sites. The left figures (a) and (c) show two different derivations and the right figures (b) and (d) show the elementary trees used in the respective derivation.

syntax tree may not uniquely specify the derivation, as illustrated in Figure 1 which shows two derivations using different elementary trees to produce the same tree.

A *Probabilistic Tree Substitution Grammar* (PTSG), like a PCFG, assigns a probability to each rule in the grammar, denoted $P(e|c)$ where the elementary tree e rewrites nonterminal c. The probability of a derivation $\mathbf{e}$ is the product of the probabilities of its component rules. Thus if we assume that each rewrite is conditionally independent of all others given its root nonterminal c (as in standard TSG models),[6] then we have

$$P(\mathbf{e}) = \prod_{c \to e \in \mathbf{e}} P(e|c)\,. \tag{1}$$

The probability of a tree, t, and string of words, $\mathbf{w}$, are given by

$$P(t) = \sum_{\mathbf{e}:\text{tree}(\mathbf{e})=t} P(\mathbf{e}) \quad \text{and}$$
$$P(\mathbf{w}) = \sum_{t:\text{yield}(t)=\mathbf{w}} P(t)\,,$$

respectively, where tree($\mathbf{e}$) returns the tree for the derivation $\mathbf{e}$ and yield(t) returns the string of terminal symbols at the leaves of t.

6. Note that this conditional independence does not hold for our model because (as we will see in Section 3) we integrate out the model parameters.

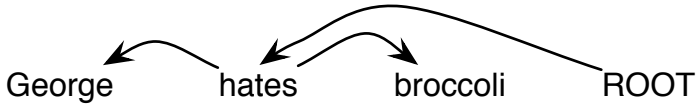


Figure 2: An unlabelled dependency analysis for the example sentence *George hates broccoli*. The artificial Root node denotes the head of the sentence.

Estimating a PTSG requires learning the sufficient statistics for $P(e|c)$ in (1) based on a training sample. Estimation has been done in previous work in a variety of ways, for example using heuristic frequency counts (Bod, 1993), a maximum likelihood estimate (Bod, 2000) and heldout estimation (Prescher et al., 2004). Parsing involves finding the most probable tree for a given string, that is, $\arg\max_t P(t|\mathbf{w})$. This is typically simplified to finding the most probable derivation, which can be done efficiently using the CYK algorithm. A number of improved algorithms for parsing have been reported, most notably a Monte-Carlo approach for finding the maximum probability tree (Bod, 1995) and a technique for maximising labelled recall using inside-outside inference in a PCFG reduction grammar (Goodman, 1998).

2.1 Dependency Grammars

Due to the wide availability of annotated treebanks, phrase structure grammars have become a popular formalism for building supervised parsers, and we will follow this tradition by using phrase structure trees from the Wall Street Journal corpus (Marcus et al., 1993) as the basis for our supervised grammar induction experiments (grammar refinement). However, the choice of formalism for unsupervised induction is a more nuanced one. The induction of phrase-structure grammars is notoriously difficult, since these grammars contain two kinds of ambiguity: the constituent structure and the constituent labels. In particular, constituent labels are highly ambiguous: firstly we don't know *a priori* how many there are, and secondly labels that appear high in a tree (e.g., an S category for a clause) rely on the correct inference of all the latent labels above and below them. Much of the recent work on unsupervised grammar induction has therefore taken a different approach, focusing on inducing dependency grammars (Mel′čuk, 1988). In applying our model to unsupervised grammar induction we follow this trend by inducing a dependency grammar. Dependency grammars represent the structure of language through directed links between words, which relate words (heads) with their syntactic dependents (arguments). An example dependency tree is shown in Figure 2, where directed arcs denote each word's arguments (e.g., hates has two arguments, 'George' and 'broccoli'). Dependency grammars are less ambiguous than phrase-structure grammars since the set of possible constituent labels (heads) is directly observed from the words in the sentence, leaving only the induction challenge of determining the tree structure. Most dependency grammar formalisms also include labels on the links between words, denoting, for example, subject, object, adjunct etc. In this work we focus on inducing unlabelled directed dependency links and assume that these links form a projective tree (there are no crossing links, which correspond to discontinuous constituents). We leave the problem of inducing labeled dependency grammars to further work.

Although we will be inducing dependency parses in our unsupervised experiments, we define our model in the following section using the formalism of a phrase-structure grammar. As detailed in Section 7, the model can be used for dependency grammar induction by using a specially designed phrase-structure grammar to represent dependency links.

3. Model

In defining our model, we focus on the unsupervised case, where we are given a corpus of text strings $\mathbf{w}$ and wish to learn a tree-substitution grammar G that we can use to infer the parses for our strings and to parse new data. (We will handle the supervised scenario, where we are given observed trees $\mathbf{t}$, in Section 4; we treat it as a special case of the unsupervised model using additional constraints during inference.) Rather than inferring a grammar directly, we go through an intermediate step of inferring a distribution over the derivations used to produce $\mathbf{w}$, that is, a distribution over sequences of elementary trees $\mathbf{e}$ that compose to form $\mathbf{w}$ as their yield. We will then essentially read the grammar off the elementary trees, as described in Section 5. Our problem therefore becomes one of identifying the posterior distribution of $\mathbf{e}$ given $\mathbf{w}$, which we can do using Bayes' Rule,

$$P(\mathbf{e}|\mathbf{w}) \propto P(\mathbf{w}|\mathbf{e})P(\mathbf{e})\,.$$

Note that any sequence of elementary trees uniquely specifies a corresponding sequence of words: those words that can be read off the leaves of the elementary trees in sequence. Therefore, given a sequence of elementary trees $\mathbf{e}$, $P(\mathbf{w}|\mathbf{e})$ either equals 1 (if $\mathbf{w}$ is consistent with $\mathbf{e}$) or 0 (otherwise). Thus, in our model, all the work is done by the prior distribution over elementary trees,

$$P(\mathbf{e}|\mathbf{w}) \propto P(\mathbf{e})\,\delta(w(\mathbf{e}), \mathbf{w})\,,$$

where δ is the Kronecker delta and $w(\mathbf{e}) = \text{yield}(\text{tree}(\mathbf{e}))$ returns the string yield of the tree defined by the derivation $\mathbf{e}$.

Because we have no way to know ahead of time how many elementary trees might be needed to account for the data, we use a nonparametric Bayesian prior, specifically the Pitman-Yor process (PYP) (Pitman, 1995; Pitman and Yor, 1997; Ishwaran and James, 2003), which is a generalization of the more widely known Dirichlet process (Ferguson, 1973). Drawing a sample from a PYP (or DP) yields a probability distribution G with countably infinite support. The PYP has three parameters: a *discount parameter* a, a *strength parameter* b, and a *base distribution* P_{E}. Informally, the base distribution determines which items will be in the support of G (here, we will define P_{E} as a distribution over elementary trees, so that G is also a distribution over elementary trees), and the discount and strength parameters a and b determine the shape of G. The discount parameter a ranges from 0 to 1; when $a = 0$, the PYP reduces to a Dirichlet process, in which case the strength parameter b is known as the *concentration parameter* and is usually denoted with α. We discuss the roles of a and b further below.

Assuming an appropriate definition for P_{E} (we give a formal definition below), we can use the PYP to define a distribution over sequences of elementary trees $\mathbf{e} = e_1 \ldots e_n$ as

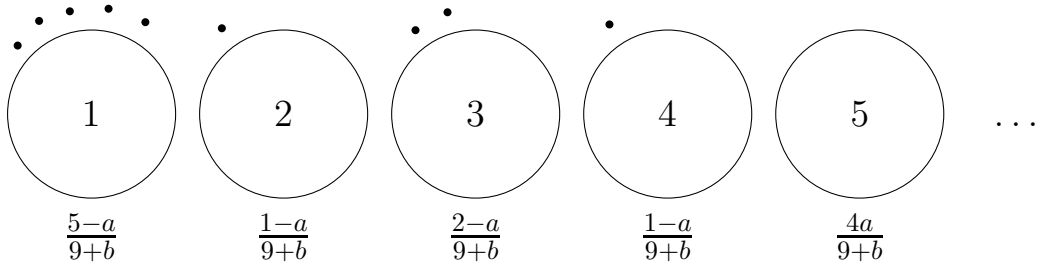


Figure 3: An example of the Pitman-Yor Chinese restaurant process with $\mathbf{z}_{-10} = (1, 2, 1, 1, 3, 1, 1, 4, 3)$. Black dots indicate the number of customers sitting at each table, and the value listed below table k is $P(z_{10} = k|\mathbf{z}_{-10})$.

follows:

$$\begin{aligned} G|a, b, P_{\text{E}} &\sim \text{PYP}(a, b, P_{\text{E}}) \\ e_i|G &\sim G\,. \end{aligned} \tag{2}$$

In this formulation, G is an infinite distribution over elementary trees drawn from the PYP prior, and the e_i are drawn *iid* from G. However, since it is impossible to explicitly represent an infinite distribution, we integrate over possible values of G, which induces dependencies between the e_i. Perhaps the easiest way to understand the resulting distribution over $\mathbf{e}$ is through a variant of the Chinese restaurant process (CRP; Aldous, 1985; Pitman, 1995) that is often used to explain the Dirichlet process. Imagine a restaurant with an infinite number of tables, each with an infinite number of seats. Customers enter the restaurant one at a time and seat themselves at a table. If z_i is the index of the table chosen by the ith customer, then the Pitman-Yor Chinese Restaurant Process (PYCRP) defines the distribution

$$P(z_i = k|\mathbf{z}_{-i}) = \begin{cases} \frac{n_k^- - a}{i-1+b} & 1 \leq k \leq K^- \\ \frac{K^- a + b}{i-1+b} & k = K^- + 1 \end{cases},$$

where $\mathbf{z}_{-i}$ is the seating arrangement of the $i-1$ previous customers, n_k^- is the number of customers in $\mathbf{z}_{-i}$ who are seated at table k, $K^- = K(\mathbf{z}_{-i})$ is the total number of tables in $\mathbf{z}_{-i}$, and $z_1 = 1$ by definition. Figure 3 illustrates. When $a = 0$, this process reduces to the standard Chinese restaurant process. Like the CRP, the PYCRP is exchangeable and produces a power-law distribution on the number of customers at each table (Pitman, 2006). The hyperparameters a and b together control the manner of the clustering, although the difference between the two is rather subtle. A high value of b will bias towards more clusters irrespective of their individual sizes, only accounting for their aggregate size. In contrast a large $a \to 1$ will bias the clustering towards smaller individual clusters.

The PYCRP produces a sequence of integers $\mathbf{z}$ whose joint probability is

$$\begin{aligned} P(\mathbf{z}) &= \prod_{i=1}^{n} P(z_i|\mathbf{z}_{1\ldots i-1}) \\ &= 1 \cdot \prod_{i=2}^{n} P(z_i|\mathbf{z}_{1\ldots i-1}) \\ &= \left(\prod_{j=1}^{n-1} \frac{1}{j+b}\right)\left(\prod_{k=1}^{K(\mathbf{z})-1}(ka+b)\right)\left(\prod_{k=1}^{K(\mathbf{z})}\prod_{j=1}^{n_k^- -1}(j-a)\right) \\ &= \frac{\Gamma(1+b)}{\Gamma(n+b)}\left(\prod_{k=1}^{K-1}(ka+b)\right)\left(\prod_{k=1}^{K}\frac{\Gamma(n_k^- -a)}{\Gamma(1-a)}\right), \end{aligned} \tag{3}$$

where K is the total number of tables in $\mathbf{z}$ and Γ is the gamma function. In order to produce a sequence of elementary trees $\mathbf{e}$ we need to introduce a second step in the process. We can do so by imagining that each table in the restaurant is labelled with an elementary tree, with $\boldsymbol{\ell}(\mathbf{z}) = \ell_1 \ldots \ell_K$ being the trees labelling each table. Whenever a customer sits at a previously unoccupied table, a label is chosen for that table according to the base distribution P_{E}, and this label is used by all following customers at that table, as illustrated in Figure 4. We define e_i to be ℓ_{z_i}, the label of the table chosen by the ith customer. This yields the following conditional distribution on e_i:

$$\begin{aligned} P(e_i = e|\mathbf{z}_{-i}, \boldsymbol{\ell}(\mathbf{z}_{-i})) &= \sum_{k=1}^{K(\mathbf{z}_{-i})} \delta(\ell_k, e)\frac{n_k^{(\mathbf{z}_{-i})} - a}{i-1+b} + \frac{K(\mathbf{z}_{-i})a+b}{i-1+b}P_{\mathrm{E}}(e) \\ &= \frac{n_e^- - K_e(\mathbf{z}_{-i})a + (K(\mathbf{z}_{-i})a+b)P_{\mathrm{E}}(e)}{i-1+b}, \end{aligned} \tag{4}$$

where K_e^- is the number of tables labelled with e in $\mathbf{z}_{-i}$, and δ is the Kronecker delta. The probability of an entire sequence of elementary trees is

$$P(\mathbf{e}) = \sum_{\mathbf{z},\boldsymbol{\ell}} P(\mathbf{e}, \mathbf{z}, \boldsymbol{\ell}),$$

where $P(\mathbf{e}, \mathbf{z}, \boldsymbol{\ell}) = 0$ except when $\ell_{z_i} = e_i$ for all i, in which case

$$\begin{aligned} P(\mathbf{e}, \mathbf{z}, \boldsymbol{\ell}) &= P(\mathbf{z}, \boldsymbol{\ell}) = P(\mathbf{z})P(\boldsymbol{\ell}|\mathbf{z}) \\ &= \frac{\Gamma(1+b)}{\Gamma(n+b)}\left(\prod_{k=1}^{K-1}(ka+b)\right)\left(\prod_{k=1}^{K}\frac{\Gamma(n_k^- -a)}{\Gamma(1-a)}P_{\mathrm{E}}(\ell_k)\right), \end{aligned}$$

where K is the total number of tables in $\mathbf{z}$.

Equation 4 shows that, like other PYP and DP models, this model can be viewed as a *cache model*, where e_i can be generated in one of two ways: by drawing from the base distribution or by drawing from a cache of previously generated elementary trees, where the probability of any particular elementary tree is proportional to the discounted frequency of

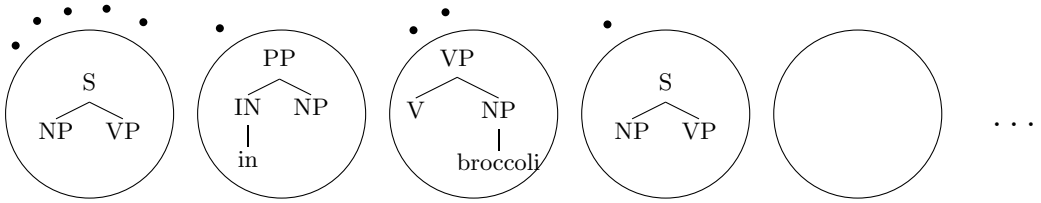


Figure 4: The Pitman-Yor process, illustrated as a labelled Chinese restaurant process. In this example, $\mathbf{z}_{-10} = (1, 2, 1, 1, 3, 1, 1, 4, 3)$ and each table k is labelled with an elementary tree ℓ_k. Black dots indicate the number of occurrences of each tree in $\mathbf{e} = (\ell_1, \ell_2, \ell_1, \ell_1, \ell_3, \ell_1, \ell_1, \ell_4, \ell_3)$. In this illustration, which corresponds to the model given in (2), a single Pitman-Yor process is used to generate all elementary trees, so the trees do not necessarily fit together properly. Our complete model, defined in (5), would have a separate Pitman-Yor restaurant for each root category.

that tree. This view makes it clear that the model embodies a "rich-get-richer" dynamic in which a few elementary trees will occur with high probability, but many will occur only once or twice, as is typical of natural language.

In the model just defined, a single PYP generates all of the elementary trees in $\mathbf{e}$. Notice, however, that these elementary trees might not tile together properly to create full syntax trees. For example, in Figure 4, $e_1 =$ (S NP VP) and $e_2 =$ (PP (IN in) NP), where the first substitution site in e_1 is an NP, but the root of e_2 is a PP, so e_2 cannot be used to expand e_1. To solve this problem, we modify the model so that there is a separate PYP for each non-terminal category c, with a base distribution conditioned on c. The distribution over elementary trees with root category c is defined as

$$\begin{aligned} G_c | a_c, b_c, P_{\text{E}} &\sim \text{PYP}(a_c, b_c, P_{\text{E}}(\cdot|c)) \\ e | c, G_c &\sim G_c \,, \end{aligned} \tag{5}$$

where $P_{\text{E}}(\cdot|c)$ is a distribution over the infinite space of elementary trees rooted with c, and a_c and b_c are the PYP hyper-parameters for non-terminal c. We elect not to tie together the values of these hyper-parameters as these control the tendency to infer larger or smaller sets of elementary trees from the observed data; we expect the distribution over productions to differ substantially between non-terminals. To generate $\mathbf{e}$, we now draw e_1 from G_S, giving us an elementary tree with frontier nodes $c_1 \dots c_m$. We then draw $e_2 \dots e_m$ in turn from $G_{c_1} \dots G_{c_m}$. We continue in this fashion until a full tree is generated, at which point we can start again with a draw from G_S.

Integrating over G_c, we obtain the following distribution over e_i, now conditioned on its root category as well as the previously generated table labels and assignments:

$$P(e_i = e | c, \mathbf{z}_{-i}, \boldsymbol{\ell}(\mathbf{z}_{-i})) = \frac{n_e^- - K_e^- a_c + (K_c^- a_c + b_c) P_{\text{E}}(e|c)}{n_c^- + b_c} \,, \tag{6}$$

where $K_c^- = \sum_{e:\text{root}(e)=c} K_e^-$ is the total number of tables for nonterminal c, n_e^- is the number of times e has been used to rewrite c and $n_c^- = \sum_{e:\text{root}(e)=c} n_e^-$ is the total count of

rules rewriting c. As before, the $^-$ superscript denotes that the counts are calculated over the previous elementary trees, $\mathbf{e}_{-i}$, and their seating arrangements, $\mathbf{z}_{-i}$.

Finally, we turn to the definition of the base distribution over elementary trees, P_{E}. Recall that in an elementary tree, each internal node is labelled with a non-terminal category symbol and each frontier (leaf) node is labelled with either a non-terminal or a terminal symbol. Given a probabilistic context-free grammar R, we assume that elementary trees are generated (conditioned on the root non-terminal c) using the following generative process. First, choose a PCFG production $c \to \alpha$ for expanding c according to the distribution given by R. Next, for each non-terminal in α decide whether to stop expanding (creating a non-terminal frontier node, also known as a substitution site) or to continue expanding. If the choice is to continue expanding, a new PCFG production is chosen to expand the child, and the process continues recursively. The generative process completes when the frontier is composed entirely of substitution sites and terminal symbols.

Assuming a fixed distribution P_{C} over the rules in R, this generative process leads to the following distribution over elementary trees:

$$P_{\mathrm{E}}(e|c) = \prod_{i \in I(e)} (1 - s_{c_i}) \prod_{f \in F(e)} s_{c_f} \prod_{c' \to \alpha \in e} P_{\mathrm{C}}(\alpha|c') \ , \tag{7}$$

where $I(e)$ are the set of internal nodes in e excluding the root, $F(e)$ are the set of frontier non-terminal nodes, c_i is the non-terminal symbol for node i and s_c is the probability of stopping expanding a node labelled c. We treat s_c as a parameter which is estimated during training, as described in Section 4.3. In the supervised case it is reasonable to assume that P_{C} is fixed; we simply use the maximum-likelihood PCFG distribution estimated from the training corpus (i.e., $P_{\mathrm{C}}(\alpha|c')$ is simply the relative frequency of the rule $c' \to \alpha$). In the unsupervised case, we will infer P_{C}; this requires extending the model to assume that P_{C} is itself drawn from a PYP prior with a uniform base distribution. We describe this extension below, along with its associated changes to equation 14.

The net effect of our base distribution is to bias the model towards simple rules with a small number of internal nodes. The geometric increase in cost associated with the stopping decisions discourages the model from using larger rules; for these rules to be included they must occur very frequently in the corpus. Similarly, rules which use high-probability (frequent) CFG productions are favoured. It is unclear if these biases are ideal: we anticipate that other, more sophisticated distributions would improve the model's performance.

In the unsupervised setting we no longer have a training set of annotated trees and therefore do not have a PCFG readily available to use as the base distribution in Equation 7. For this reason we extend the previous model to a two level hierarchy of PYPs. As before, the topmost level is defined over the elementary tree fragments (G_c) with the base distribution (P_{E}) assigning probability to the infinite space of possible fragments. The model differs from the supervised one by defining P_C in (7) using a PYP prior over CFG rules. Accordingly the model can now infer a two level hierarchy consisting of a PCFG embedded within a TSG, compared to the supervised parsing model which only learnt the TSG level with a

fixed PCFG. Formally, each CFG production is drawn from[7]

$$\begin{aligned} H_c|\, a'_c, b'_c &\sim \text{PYP}(a'_c, b'_c, \text{Uniform}(\cdot|c)) \\ \alpha|\, c, H_c &\sim H_c\,, \end{aligned} \tag{8}$$

where a'_c and b'_c are the PYP hyper-parameters and $\text{Uniform}(\cdot|c)$ is a uniform distribution over the space of rewrites for non-terminal c.[8] As before, we integrate out the model parameters, H_c. Consequently draws from P_{C} are no longer *iid* but instead are tied in the prior, and the probability of the sequence of component CFG productions $\{c' \to \alpha \in e\}$ now follows a Pitman-Yor Chinese Restaurant Process.

The CFG level and TSG level PYCRPs are connected as follows: every time an elementary tree is assigned to a new table in the TSG level, each of its component CFG rules are drawn from the CFG level prior. Note that, just as elementary trees are divided into separate restaurants at the TSG level based on their root categories, CFG rules are divided into separate restaurants at the CFG level based on their left-hand sides. Formally, the probability of r_j, the j^{th} CFG rule in the sequence, is given by

$$P_{\text{C}}(r_j = r|\, c_j = c, \mathbf{z}'_{-j}, \boldsymbol{\ell}'_{-j}) = \frac{n'^{-}_r - K'^{-}_r a'_c + (K'^{-}_c a'_c + b'_c)\,\frac{1}{|R_c|}}{K'^{-}_c + b'_c}\,, \tag{9}$$

where c_j is the left-hand side of r_j; $\mathbf{z}'_{-j}$ and $\boldsymbol{\ell}'_{-j}$ are the table assignments and table labels in the CFG-level restaurants (we use prime symbols to indicate variables pertaining to the CFG level); n'^{-}_r is the number of times rule r is used in any table label in a TSG restaurant (equivalently, the number of customers at tables labelled r in the CFG restaurants); K'^{-}_r and $K'^{-}_c = \sum_{r:\text{root}(r)=c} K'^{-}_r$ are the CFG-level table counts for r and all rules rewriting c, respectively; and R_c is the set of CFG productions which can rewrite c. This formulation reflects that we now have multiple tied restaurants, and each time an elementary tree opens a new table in a top-level restaurant all its rules are considered to have entered their own respective P_{C} restaurants (according to their root c). Accordingly the CFG-level customer count, n'^{-}_r, is the number of occurrences of r in the elementary trees that label the tables in the TSG restaurants (excluding r_j). Thus, in the unsupervised case, the product of rule probabilities (the final factor) in Equation (7) is computed by multiplying together the conditional probability of each rule (9) given the previous ones.

4. Training

We present two alternative algorithms for training our model, both based on Markov chain Monte Carlo techniques, which produce samples from the posterior distribution of the model by iteratively resampling the values of the hidden variables (tree nodes). The first algorithm is a *local* sampler, which operates by making a local update to a single tree node in each sampling step. The second algorithm is a *blocked* sampler, which makes much larger moves

7. As we are using a finite base distribution over CFG productions, we could use a Dirichlet instead of the PYP presented in (8). However we elect to use a PYP because it is more general, having additional expressive power from its discounting behaviour.
8. In our experiments on unsupervised dependency parsing the space of rewrites varied depending on c, and can be as large as the set of part-of-speech tags. See Section 7 for details.

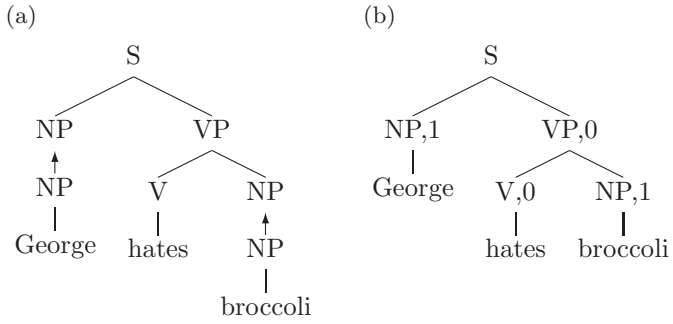


Figure 5: Gibbs sampler state (b) corresponding to the example derivation (a) (reproduced from Figure 1a). Each node is labelled with its substitution variable.

by sampling analyses for full sentences, which should improve the mixing over the local sampler. Importantly the blocked sampler is more general, being directly applicable to both supervised and unsupervised settings (and for parsing test sentences, which is equivalent to an unsupervised setting) while the local sampler is only applicable for supervised learning, where the trees are observed. We now present the two sampling methods in further detail.

4.1 Local Sampler

The *local* sampler is designed specifically for the supervised scenario, and samples a TSG derivation for each tree by sampling local updates at each tree node. It uses Gibbs sampling (Geman and Geman, 1984), where random variables are repeatedly sampled conditioned on the current values of all other random variables in the model. The actual algorithm is analogous to the Gibbs sampler used for inference in the Bayesian model of word segmentation presented by Goldwater et al. (2006); indeed, the problem of inferring the derivations $\mathbf{e}$ from $\mathbf{t}$ can be viewed as a segmentation problem, where each full tree must be segmented into one or more elementary trees. To formulate the local sampler, we associate a binary variable $x_d \in \{0, 1\}$ with each non-root internal node, d, of each tree in the training set, indicating whether that node is a substitution point ($x_d = 1$) or not ($x_d = 0$). Each substitution point forms the root of some elementary tree, as well as a frontier nonterminal of an ancestor node's elementary tree. Conversely, each non-substitution point forms an internal node inside an elementary tree. Collectively the training trees and substitution variables specify the sequence of elementary trees $\mathbf{e}$ that is the current state of the sampler. Figure 5 shows an example tree with its substitution variables and its corresponding TSG derivation.

Our Gibbs sampler works by sampling the value of the x_d variables, one at a time, in random order. If d is the node associated with x_d, the substitution variable under consideration, then the two possible values of x_d define two options for $\mathbf{e}$: one in which d is internal to some elementary tree e_M, and one in which d is the substitution site connecting two smaller trees, e_A and e_B. In the example in Figure 5, when sampling the VP node, e_M = (S NP (VP (V hates) NP)), e_A = (S NP VP), and e_B = (VP (V hates) NP). To sample a value for x_d, we compute the probabilities of e_M and (e_A, e_B), conditioned on $\mathbf{e}^-$: all other elementary trees in the training set that share at most a root or frontier nonterminal with e_M, e_A, or e_B. These probabilities are easy to compute because the PYP

is *exchangeable*, meaning that the probability of a set of outcomes does not depend on their ordering. Therefore we can treat the elementary trees under consideration as the last ones to be sampled, and apply Equation (6). We then sample one of the two outcomes (merging or splitting) according to the relative probabilities of these two events. More specifically, the probabilities of the two outcomes, conditioned on the current analyses of the remainder of the corpus, are

$$P(e_M|c_M) = \frac{n^-_{e_M} - K^-_{e_M} a_{c_M} + (K^-_{c_M} a_{c_M} + b_{c_M}) P_{\mathrm{E}}(e_M|c_M)}{n^-_{c_M} + b_{c_M}} \quad \text{and}$$

$$\begin{aligned} P(e_A, e_B|c_A, c_B) &= \sum_{z_{e_A}} P(e_A, z_{e_A}|c_A) P(e_B|e_A, z_{e_A}, c_A, c_B) \\ &= \frac{n^-_{e_A} - K^-_{e_A} a_{c_A}}{n^-_{c_A} + b_{c_A}} \times \frac{n^-_{e_B} + \delta_e - K^-_{e_B} a_{c_B} + (K^-_{c_B} a_{c_B} + b_{c_B}) P_{\mathrm{E}}(e_B|c_B)}{n^-_{c_B} + \delta_c + b_{c_B}} \\ &\quad + \frac{(K^-_{c_A} a_{c_A} + b_{c_A}) P_{\mathrm{E}}(e_A|c_A)}{n^-_{c_A} + b_{c_A}} \\ &\quad \times \frac{n^-_{e_B} + \delta_e - (K^-_{e_B} + \delta_e) a_{c_B} + \left((K^-_{c_B} + \delta_c) a_{c_B} + b_{c_B}\right) P_{\mathrm{E}}(e_B|c_B)}{n^-_{c_B} + \delta_c + b_{c_B}}, \end{aligned} \quad (10)$$

where c_y is the root label of e_y, the counts n^- and K^- are derived from $\mathbf{z}_{-M}$ and $\boldsymbol{\ell}(\mathbf{z}_{-M})$ (this dependency is omitted from the conditioning context for brevity), $\delta_e = \delta(e_A, e_B)$ is the Kronecker delta function which has value one when e_A and e_B are identical and zero otherwise, and similarly for $\delta_c = \delta(c_A, c_B)$ which compares their root nonterminals c_A and c_B. The δ terms reflect the changes to n^- that would occur after observing e_A, which forms part of the conditioning context for e_B. The two additive terms in (10) correspond to different values of z_{e_A}, the seating assignment for e_A. Specifically, the first term accounts for the case where e_A is assigned to an existing table, $z_{e_A} < K^-_{e_A}$, and the second term accounts for the case where e_A is seated at a new table, $z_{e_A} = K^-_{e_A}$. The seating affects the conditional probability of e_B by potentially increasing the number of tables $K^-_{e_A}$ or $K^-_{c_A}$ (relevant only when $e_A = e_B$ or $c_A = c_B$).

4.2 Blocked Sampler

The local sampler has the benefit of being extremely simple, however it may suffer from slow convergence (poor mixing) due to its very local updates. That is, it can get stuck because many locally improbable decisions are required to escape from a locally optimal solution. Moreover it is only applicable to the supervised setting: it cannot be used for unsupervised grammar induction or for parsing test strings. For these reasons we develop the *blocked* sampler, which updates blocks of variables at once, where each block consists of all the the nodes associated with a single sentence. This sampler can make larger moves than the local sampler and is more flexible, in that it can perform inference with both string (unsupervised) or tree (supervised) input.[9]

9. A recently-proposed alternative approach is to perform *type-level* updates, which samples updates to many similar tree fragments at once (Liang et al., 2010). This was shown to converge faster than the local Gibbs sampler.

The blocked sampler updates the analysis for each sentence given the analyses for all other sentences in the training set. We base our approach on the algorithm developed by Johnson et al. (2007a) for sampling parse trees using a finite Bayesian PCFG model with Dirichlet priors over the multinomial rule probabilities. As in our model, they integrate out the parameters (in their case, the PCFG rule probabilities), leading to a similar caching effect due to interdependences between the latent variables (PCFG rules in the parse). Thus, standard dynamic programming algorithms cannot be used to sample directly from the desired posterior, $p(t|\mathbf{w}, \mathbf{t}^-)$, that is, the distribution of parse trees for the current sentence given the words in the corpus and the trees for all other sentences. To solve this problem, Johnson et al. (2007a) developed a Metropolis-Hastings (MH) sampler. The MH algorithm is an MCMC technique which allows for samples to be drawn from a probability distribution, $\pi(s)$, by first drawing samples from a *proposal distribution*, $Q(s'|s)$, and then correcting these to the true distribution using an acceptance/rejection step. Given a state s, we sample a next state $s' \sim Q(\cdot|s)$ from the proposal distribution; this new state is accepted with probability

$$A(s, s') = \min\left\{\frac{\pi(s')Q(s|s')}{\pi(s)Q(s'|s)}, 1\right\}$$

and is rejected otherwise, in which case s is retained as the current state. The Markov chain defined by this process is guaranteed to converge on the desired distribution, $\pi(s)$. Critically, the MH algorithm enables sampling from distributions from which we cannot sample directly, and moreover, we need not know the normalisation constant for $\pi(\cdot)$, since it cancels in $A(s, s')$.

In Johnson et al.'s (2007a) algorithm for sampling from a Bayesian PCFG, the proposal distribution is simply $Q(t'|t) = P(t'|\theta^{MAP})$, the posterior distribution over trees given fixed parameters θ^{MAP}, where θ^{MAP} is the MAP estimate based on the conditioning data, $\mathbf{t}^-$. Note that the proposal distribution is a close fit to the true posterior, differing only in that under the MAP the production probabilities in a derivation are *iid*, while for the true model the probabilities are tied by the prior (giving rise to the caching effect). The benefit of using the MAP is that its independences mean that inference can be solved using dynamic programming, namely the inside algorithm (Lari and Young, 1990). Given the inside chart, which stores the aggregate probability of all subtrees for each word span and rooted with a given nonterminal label, samples can be drawn using a simple top-down recursion (Johnson et al., 2007a).

Our model is similar to Johnson et al.'s, as we also use a Bayesian prior in a model of grammar induction and consequently face similar problems with directly sampling due to the caching effects of the prior. For this reason, we use the MH algorithm in a similar manner, except in our case we draw samples of derivations of elementary trees and their seating assignments, $p(\mathbf{z}_i, \boldsymbol{\ell}_i|\mathbf{w}, \mathbf{z}_{-i}, \boldsymbol{\ell}_{-i})$, and use a MAP estimate over $(\mathbf{z}_{-i}, \boldsymbol{\ell}_{i-})$ as our proposal distribution.[10] However, we have an added complication: the MAP cannot be estimated directly. This is a consequence of the base distribution having infinite support, which means the MAP has an infinite rule set. For finite TSG models, such as those used in DOP, constructing a CFG equivalent grammar is straightforward (if unwieldy). This can

10. Calculating the proposal and acceptance probabilities requires sampling not just the elementary trees, but also their table assignments (for both levels in the hierarchical model). We elected to simplify the implementation by separately sampling the elementary trees and their table assignments.

be done by creating a rule for each elementary tree which rewrites its root nonterminal as its frontier. For example under this technique S → NP (VP (V hates) NP would be mapped to S → NP hates NP.[11] However, since our model has infinite support over productions, it cannot be mapped in the same way. For example, if our base distribution licences the CFG production NP → NP PP then our TSG grammar will contain the infinite set of elementary trees NP → NP PP, NP → (NP NP PP) PP, NP → (NP (NP NP PP) PP) PP, ..., each with decreasing but non-zero probability. These would all need to be mapped to CFG rules in order to perform inference under the grammar, which is clearly impossible.

Thankfully it is possible to transform the infinite MAP-TSG into a finite CFG, using a method inspired by Goodman (2003), who developed a grammar transform for efficient parsing with an all-subtrees DOP grammar. In the transformed grammar inside inference is tractable, allowing us to draw proposal samples efficiently and thus construct a Metropolis-Hastings sampler. The resultant grammar allows for efficient inference, both in unsupervised and supervised training and in parsing (see Section 5).

We represent the MAP using the grammar transformation in Table 1, which separates the count and base distribution terms in Equation 6 into two separate CFGs, denoted A and B. We reproduce Equation 6 below along with its decomposition:

$$\begin{aligned} P(e_i = e | c, \mathbf{z}_{-i}, \boldsymbol{\ell}(\mathbf{z}_{-i})) &= \frac{n_e^- - K_e^- a_c + (K_c^- a_c + b_c) P_{\mathrm{E}}(e|c)}{n_c^- + b_c} \\ &= \underbrace{\frac{n_e^- - K_e^- a_c}{n_c^- + b_c}}_{\text{count}} + \underbrace{\frac{K_c^- a_c + b_c}{n_c^- + b_c} P_{\mathrm{E}}(e|c)}_{\text{base}} . \end{aligned} \tag{11}$$

Grammar A has productions for every elementary tree e with $n_e^- \geq 1$, which are assigned as their probability the count term in Equation 11.[12] The function sig(e) returns a string signature for elementary trees, for which we use a form of bracketed notation. To signify the difference between these nonterminal symbols and trees, we use curly braces and hyphens in place of round parentheses and spaces, respectively, for example, the elementary tree (S NP (VP (V hates) NP)) is denoted by the nonterminal symbol {S-NP-{VP-{V-hates}-NP}}. Grammar B has productions for every CFG production licensed under P_{E}; its productions are denoted using primed (') nonterminals. The rule $c \to c'$ bridges from A to B, weighted by the base term in Equation 11 excluding the $P_{\mathrm{E}}(e|c)$ factor. The contribution of the base distribution is computed recursively via child productions. The remaining rules in grammar B correspond to every CFG production in the underlying PCFG base distribution, coupled with the binary decision of whether or not nonterminal children should be substitution sites (frontier nonterminals). This choice affects the rule probability by including an s or $1 - s$ factor, and child substitution sites also function as a bridge back from grammar B to A. There are often two equivalent paths to reach the same chart cell

11. Alternatively, interspersing a special nonterminal, for example, S → {S-NP-{VP-{V-hates}-NP} → NP hates NP, encodes the full structure of the elementary tree, thereby allowing the mapping to be reversed. We use a similar technique to encode non-zero count rules in our grammar transformation, described below.

12. The transform assumes inside inference, where alternate analyses for the same span of words with the same nonterminal are summed together. In Viterbi inference the summation is replaced by maximisation, and therefore we need different expansion probabilities. This requires changing the weight for $c \to \mathrm{sig}(e)$ to $P(e_i = e | c, \mathbf{z}_{-i}, \boldsymbol{\ell}(\mathbf{z}_{-i}))$ in Table 1.

Grammar A	For every ET, e, rewriting c with non-zero count:	
	$c \rightarrow \text{sig}(e)$	$\frac{n_e^- - K_e^- a_c}{n_c^- + b_c}$
	For every internal node e_i in e with children $e_{i,1}, \ldots, e_{i,n}$	
	$\text{sig}(e_i) \rightarrow \text{sig}(e_{i,1}) \ldots \text{sig}(e_{i,n})$	1
A $\rightarrow$ B	For every nonterminal, c:	
	$c \rightarrow c'$	$\frac{K_c^- a_c + b_c}{n_c^- + b_c}$
Grammar B	For every pre-terminal CFG production, $c \rightarrow t$:	
	$c' \rightarrow t$	$P_{\text{C}}(c \rightarrow t)$
	For every unary CFG production, $c \rightarrow a$:	
	$c' \rightarrow a$	$P_{\text{C}}(c \rightarrow a) s_a$
	$c' \rightarrow a'$	$P_{\text{C}}(c \rightarrow a)(1 - s_a)$
	For every binary CFG production, $c \rightarrow ab$:	
	$c' \rightarrow ab$	$P_{\text{C}}(c \rightarrow ab) s_a s_b$
	$c' \rightarrow ab'$	$P_{\text{C}}(c \rightarrow ab) s_a (1 - s_b)$
	$c' \rightarrow a'b$	$P_{\text{C}}(c \rightarrow ab)(1 - s_a) s_b$
	$c' \rightarrow a'b'$	$P_{\text{C}}(c \rightarrow ab)(1 - s_a)(1 - s_b)$

Table 1: Grammar transformation rules to map an infinite MAP TSG into an equivalent CFG, separated into three groups for grammar A (top), the bridge between A $\rightarrow$ B (middle) and grammar B (bottom). Production probabilities are shown to the right of each rule. The $\text{sig}(e)$ function creates a unique string signature for an ET e (where the signature of a frontier node is itself) and s_c is the probability of c being a substitution variable, thus stopping the P_{E} recursion.

using the same elementary tree—via grammar A using observed TSG productions and via grammar B using P_{E} backoff—which are summed to yield the desired net probability. The transform is illustrated in the example in Figures 6 and 7.

Using the transformed grammar we can represent the MAP grammar efficiently and draw samples of TSG derivations using the inside algorithm. In an unsupervised setting, that is, given a yield string as input, the grammar transform above can be used directly with the inside algorithm for PCFGs (followed by the reverse transform to map the sampled derivation into TSG elementary trees). This has an asymptotic time complexity cubic in the length of the input.

For supervised training the trees are observed and thus we must ensure that the TSG analysis matches the given tree structure. This necessitates constraining the inside algorithm to only consider spans that are present in the given tree and with the given nonterminal. Nonterminals are said to match their primed and signed counterparts, for example, VP$'$ and {VP-{V-hates}-NP} both match VP. A sample from the constrained inside chart will specify the substitution variables for each node in the tree: For each node d if it has a non-primed category in the sample then it is a substitution site, $x_d = 1$, otherwise it is an internal node, $x_d = 0$. For example, both trees in Figure 7 encode that both NP nodes

S → {NP-{VP-{V-hates}-NP}}	$\frac{n_e^- - K_e^- a_S}{n_S^- + b_S}$
{NP-{VP-{V-hates}-NP}} → NP {VP-{V-hates}-NP}	1
{VP-{V-hates}-NP} → {V-hates} NP	1
{V-hates} → hates	1
S → S'	$\frac{K_S^- a_S + b_S}{n_S^- + b_S}$
S' → NP VP'	$P_{\mathrm{C}}(\mathrm{S} \rightarrow \mathrm{NP\ VP}) s_{NP}(1 - s_{VP})$
VP' → V' NP	$P_{\mathrm{C}}(\mathrm{VP} \rightarrow \mathrm{V\ NP})(1 - s_V) s_{NP}$
V' → hates	$P_{\mathrm{C}}(\mathrm{V} \rightarrow \mathrm{hates})$

Figure 6: Example showing the transformed grammar rules for the single elementary tree $e =$(S NP (VP (V hates) NP)) and the scores for each rule. Only the rules which correspond to e and its substitution sites are displayed. Taking the product of the rule scores above the dashed line yields the *count* term in (11), and the product of the scores below the line yields the *base* term. When the two analyses are combined and their probabilities summed together, we get $P(e_i = e | c, \mathbf{z}_{-i}, \boldsymbol{\ell}(\mathbf{z}_{-i}))$.

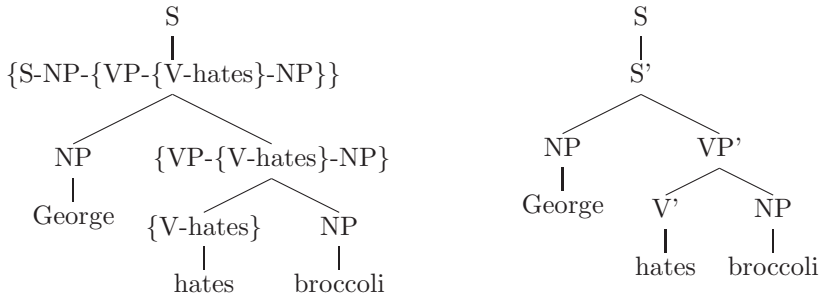


Figure 7: Example trees under the grammar transform, which both encode the same TSG derivation from Figure 1a. The left tree encodes that the S → NP (VP (V hates) NP elementary tree was drawn from the cache, while for the right tree this same elementary tree was drawn from the base distribution (the count and base terms in (11), respectively).

are substitution sites and that the VP and V nodes are not substitution sites (the same configuration as Figure 5).

The time complexity of the constrained inside algorithm is linear in the size of the tree and the length of the sentence. The local sampler also has the same time complexity, however it is not immediately clear which technique will be faster in practise. It is likely that the blocked sampler will have a slower runtime due to its more complicated implementation, particularly in transforming the grammar and inside inference. Although the two samplers have equivalent asymptotic complexity, the constant factors may differ greatly. In Section 6 we compare the two training methods empirically to determine which converges more quickly.

4.3 Sampling Hyperparameters

In the previous discussion we assumed that we are given the model hyperparameters, $(\mathbf{a}, \mathbf{b}, \mathbf{s})$. While it might be possible to specify their values manually or fit them using a development set, both approaches are made difficult by the high dimensional parameter space. Instead we treat the hyper-parameters as random variables in our model, by placing vague priors over them and infer their values during training. This is an elegant way to specify their values, although it does limit our ability to tune the model to optimise a performance metric on held-out data.

For the PYP discount parameters $\mathbf{a}$, we employ independent Beta priors, $a_c \sim \text{Beta}(1, 1)$. The prior is uniform, encoding that we have no strong prior knowledge of what the value of each a_c should be. The conditional probability of a_c given the current derivations $\mathbf{z}, \boldsymbol{\ell}$ is

$$P(a_c|\mathbf{z}, \boldsymbol{\ell}) \propto P(\mathbf{z}, \boldsymbol{\ell}|a_c) \times \text{Beta}(a_c|1, 1)\,.$$

We cannot calculate the normaliser for this probability, however $P(\mathbf{z}, \boldsymbol{\ell}|a_c)$ can be calculated using Equation 3 and thus $P(a_c|\mathbf{z}, \boldsymbol{\ell})$ can be calculated up to a constant. We use the range doubling slice sampling technique of Neal (2003) to draw a new sample of a_c' from its conditional distribution.[13]

We treat the concentration parameters, $\mathbf{b}$, as being generated by independent gamma priors, $b_c \sim \text{Gamma}(1, 1)$. We use the same slice-sampling method for a_c to sample from the conditional over b_c,

$$P(b_c|\mathbf{z}, \boldsymbol{\ell}) \propto P(\mathbf{z}, \boldsymbol{\ell}|b_c) \times \text{Gamma}(b_c|1, 1)\,.$$

This prior is not vague, that is, the probability density function decays exponentially for higher values of b_c, which serves to limits the influence of the P_{C} prior. In our experimentation we found that this bias had little effect on the generalisation accuracy of the supervised and unsupervised models, compared to a much vaguer Gamma prior with the same mean.

We use a vague Beta prior for the stopping probabilities in P_{E}, $s_c \sim \text{Beta}(1, 1)$. The Beta distribution is conjugate to the binomial, and therefore the posterior is also a Beta distribution from which we can sample directly,

$$s_c \sim \text{Beta}\left(1 + \sum_e K_e(\mathbf{z}) \sum_{n \in F(e)} \delta(n, c),\; 1 + \sum_e K_e(\mathbf{z}) \sum_{n \in I(e)} \delta(n, c)\right),$$

where e ranges over all non-zero count elementary trees, $F(e)$ are the nonterminal frontier nodes in e, $I(e)$ are the non-root internal nodes and the δ terms count the number of nodes in e with nonterminal c. In other words, the first Beta argument is the number of tables in which a node with nonterminal c is a stopping node in the P_{E} expansion and the second argument is the number of tables in which c has been expanded (a non-stopping node).

All the hyper-parameters are resampled after every full sampling iteration over the training trees, except in the experiments in Section 7 where they are sampled every 10th iteration.

13. We used the slice sampler included in Mark Johnson's Adaptor Grammar implementation, available at `http://web.science.mq.edu.au/~mjohnson/Software.htm`.

5. Parsing

We now turn to the problem of using the model to parse novel sentences. This requires finding the maximiser of

$$p(t|\omega, \mathbf{w}) = \int\int\int\int p(t|\omega, \mathbf{z}, \boldsymbol{\ell}, \mathbf{a}, \mathbf{b}, \mathbf{s})\, p(\mathbf{z}, \boldsymbol{\ell}, \mathbf{a}, \mathbf{b}, \mathbf{s}|\mathbf{w})\; d\mathbf{z}\; d\boldsymbol{\ell}\; d\mathbf{a}\; d\mathbf{b}\; d\mathbf{s}\,, \tag{12}$$

where ω is the sequence of words being parsed, t is the resulting tree, $\mathbf{w}$ are the training sentences, $\mathbf{z}$ and $\boldsymbol{\ell}$ represent their parses, elementary tree representation and table assignments and $(\mathbf{a}, \mathbf{b}, \mathbf{s})$ are the model's hyper-parameters. For the supervised case we use the training trees, $\mathbf{t}$, in place of $\mathbf{w}$ in Equation 12.

Unfortunately solving for the maximising parse tree in Equation 12 is intractable. However, it can be approximated using Monte Carlo techniques. Given a sample of $(\mathbf{z}, \boldsymbol{\ell}, \mathbf{a}, \mathbf{b}, \mathbf{s})$ we can reason over the space of possible TSG derivations, e, for sentence w using the same Metropolis-Hastings sampler presented in Section 4.2 for blocked inference in the unsupervised setting.[14] This gives us a set of samples from the posterior $p(\mathbf{e}|w, \mathbf{z}, \boldsymbol{\ell}, \mathbf{a}, \mathbf{b}, \mathbf{s})$. We then use a Monte Carlo integral to obtain a marginal distribution over trees (Bod, 2003),

$$\hat{p}^{MPT}(t) = \sum_{m=1}^{M} \delta(t, \text{tree}(\mathbf{e}_m))\,, \tag{13}$$

where $\{\mathbf{e}_m\}_{m=1}^{M}$ are our sample of derivations for w. It is then straightforward to find the best parse, $t^* = \arg\max \hat{p}(t)$, which is simply the most frequent tree in the sample.

In addition to solving from the maximum probability tree (MPT) using Equation 13, we also present results for a number of alternative objectives. To test whether the derivational ambiguity is important, we also compute the maximum probability derivation (MPD),

$$\hat{p}^{MPD}(\mathbf{e}) = \sum_{m=1}^{M} \delta(\mathbf{e}, \mathbf{e}_m)\,,$$

using a Monte-Carlo integral, from which we recover the tree, $t^* = \text{tree}(\arg\max_{\mathbf{e}} \hat{p}^{MPD}(\mathbf{e}))$. We also compare using the Viterbi algorithm directly with the MAP grammar, $t^* = \text{tree}(\arg\max_{\mathbf{e}} P_{MAP}(\mathbf{e}|u$ which constitutes an approximation to the true model in which we can search exactly. This contrasts with the MPD which performs approximate search under the true model. We compare the different methods empirically in Section 6.

The MPD and MPT parsing algorithms require the computation of Monte-Carlo integrals over the large space of possible derivations or trees. Consequently, unless the distribution is extremely peaked the chance of sampling many identical structures is small, vanishingly so for long sentences (the space of trees grows exponentially with the sentence length). In other words, the sampling variance can be high which could negatively affect parsing performance. For this reason we present an alternative parsing method which compiles more local statistics for which we can obtain reliable estimates. The technique is

14. Using many samples in a Monte Carlo integral is a straight-forward extension to our parsing algorithm. We did not observe a significant improvement in parsing accuracy when using a multiple samples compared to a single sample, and therefore just present results for a single sample. However, using multiple models has been shown to improve the performance of other parsing models (Petrov, 2010).

Partition	sections	sentences	tokens	types	types (unk)
training	2–21	33180	790237	40174	21387
development	22	1700	40117	6840	5473
testing	23	2416	56684	8421	6659
small training	2	1989	48134	8503	3728

Table 2: Corpus statistics for supervised parsing experiments using the Penn treebank, reporting for each partition its WSJ section/s, the number of sentences, word tokens and unique word types. The final column shows the number of word types after unknown word processing using the full training set, which replaces rare words with placeholder tokens. The number of types after preprocessing in the development and testing sets is roughly halved when using the the small training set.

based on Goodman's (2003) algorithm for maximising labelled recall in DOP parsing and subsequent work on parsing in state-splitting CFGs (Petrov and Klein, 2007). The first step is to acquire marginal distributions over the CFG productions within each sampled tree. Specifically, we collect counts for events of the form $(c \rightarrow \alpha, i, j, k)$, where $c \rightarrow \alpha$ is a CFG production spanning words $[i, j)$ and k is the split point between child constituents for binary productions, $i < k < j$ ($k = 0$ for unary productions). These counts are then marginalised by the number of trees sampled. Finally the Viterbi algorithm is used to find the tree with the maximum cumulative probability under these marginals, which we call the maximum expected rule (MER) parse. Note that this is a type of minimum Bayes risk decoding and was first presented in Petrov and Klein (2007) as the MAX-RULE-SUM method (using exact marginals, not Monte-Carlo estimates as is done here).

6. Supervised Parsing Experiments

In this section we present an empirical evaluation of the model on the task of supervised parsing. In this setting the model learns a segmentation of a training treebank, which defines a TSG. We present parsing results using the learned grammar, comparing the effects of the sampling strategy, initialisation conditions, parsing algorithm and the size of the training set. The unsupervised model is described in the following section, Section 7.

We trained the model on the WSJ part of Penn. treebank (Marcus et al., 1993) using the standard data splits, as shown in Table 2. As our model is parameter free (the $\mathbf{a}, \mathbf{b}, \mathbf{s}$ parameters are learnt in training), we do not use the development set for parameter tuning. We expect that fitting the hyperparameters to maximise performance on the development set would lead to a small increase in generalisation performance, but at a significant cost in runtime. We adopt Petrov et al.'s (2006) method for data preprocessing: right-binarizing the trees to limit the branching factor and replacing tokens with count ≤ 1 in the training sample with one of roughly 50 generic unknown word markers which convey the token's lexical features and position. The predicted trees are evaluated using EVALB[15] and we report the F1 score over labelled constituents and exact match accuracy over all sentences in the testing sets.

15. See `http://nlp.cs.nyu.edu/evalb/`.

In our experiments, we initialised the sampler by setting all substitution variables to 0, thus treating every full tree in the training set as an elementary tree. Unless otherwise specified, the blocked sampler was used for training. We later evaluated the effect of different starting conditions on the quality of the configurations found by the sampler and on parsing accuracy. The sampler was trained for 5000 iterations and we use the final sample of $\mathbf{z}, \boldsymbol{\ell}, \mathbf{a}, \mathbf{b}, \mathbf{s}$ for parsing. We ran all four different parsing algorithms and compare their results on the testing sets. For the parsing methods that require a Monte Carlo integral (MPD, MPT and MER), we sampled 1000 derivations from the MAP approximation grammar which were then input to the Metropolis-Hastings acceptance step before compiling the relevant statistics. The Metropolis-Hastings acceptance rate was around 99% for both training and parsing. Each experiment was replicated five times and the results averaged.

6.1 Small Data Sample

For our first treebank experiments we train on a small data sample by using only section 2 of the treebank (see Table 2 for corpus statistics.) Bayesian methods tend to do well with small data samples, while for larger samples the benefits diminish relative to point estimates. For this reason we present a series of exploratory experiments on the small data set before moving to the full treebank.

In our experiments we aim to answer the following questions: Firstly, in terms of parsing accuracy, does the Bayesian TSG model outperform a PCFG baseline, and how does it compare to existing high-quality parsers? We will also measure the effect of the parsing algorithm: Viterbi, MPD, MPT and MER. Secondly, which of the local and blocked sampling techniques is more efficient at mixing, and which is faster per iteration? Finally, what kind of structures does the model learn and do they match our expectations? The hyper-parameter values are also of interest, particularly to evaluate whether the increased generality of the PYP is justified over the DP. Our initial experiments aim to answer these questions on the small data set, after which we take the best model and apply it to the full set.

Table 3 presents the prediction results on the development set. The baseline is a maximum likelihood PCFG. The TSG models significantly outperform the baseline. This confirms our hypothesis that CFGs are not sufficiently powerful to model syntax, and that the increased context afforded to the TSG can make a large difference. Surprisingly, the MPP technique is only slightly better than the MPD approach, suggesting that derivational ambiguity is not as much of a problem as previously thought (Bod, 2003). Also surprising is the fact that exact Viterbi parsing under the MAP approximation is much worse than the MPD method which uses an approximate search technique under the true model. The MER technique is a clear winner, however, with considerably better F1 scores than either MPD or MPP, with a margin of 1–2 points. This method is less affected by sampling variance than the other MC algorithms due to its use of smaller tree fragments (CFG productions at each span).

We also consider the difference between using a Dirichlet process prior (DP) and a Pitman-Yor process prior (PYP). This amounts to whether the $\mathbf{a}$ hyper-parameters are set to 0 (DP) or are allowed to take on non-zero values (PYP), in which case we sample their values as described in Section 4.3. There is a small but consistent gain of around 0.5 F1

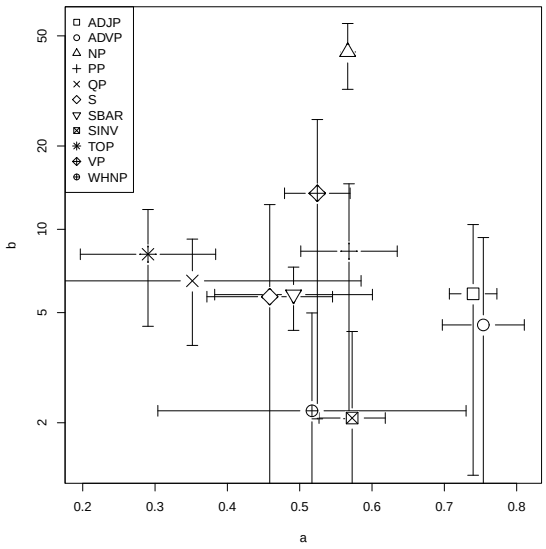
Model	Viterbi	MPD	MPP	MER	# rules
PCFG	60.20	60.20	60.20	-	3500
TSG PYP	74.90	76.68	77.17	78.59	25746
TSG DP	74.70	75.86	76.24	77.91	25339
Berkeley parser ($\tau = 2$)	71.93	71.93	-	74.32	16168
Berkeley parser ($\tau = 5$)	75.33	75.33	-	77.93	39758

Table 3: Development results for models trained on section 2 of the Penn treebank, showing labelled constituent F1 and the grammar size. For the TSG models the grammar size reported is the number of CFG productions in the transformed MAP PCFG approximation. Unknown word models are applied to words occurring less than two times (TSG models and Berkeley $\tau = 2$) or less than five times (Berkeley $\tau = 5$).

points across the different parsing methods from using the PYP, confirming our expectation that the increased flexibility of the PYP is useful for modelling linguistic data. Figure 8a shows the learned values of the PYP hyperparameters after training for each nonterminal category. It is interesting to see that the hyper-parameter values mostly separate the open-class categories, which denote constituents carrying semantic content, from the closed-class categories, which are largely structural. The open classes (noun-, verb-, adjective- and adverb-phrases: NP, VP, ADJP and ADVP, respectively) tend to have higher a and b values (towards the top right corner of the graph) and therefore can describe highly diverse sets of productions. In contrast, most of the closed classes (the root category, quantity phrases, wh-question noun phrases and sentential phrases: TOP, QP, WHNP and S, respectively) have low a and b (towards the bottom left corner of the graph), reflecting that encoding their largely formulaic rewrites does not necessitate diverse distributions.

The $\mathbf{s}$ hyper-parameter values are shown in Figure 8b, and are mostly in the mid-range (0.3–0.7). Prepositions (IN), adverbs (RB), determiners (DT) and some tenses of verbs (VBD and VBP) have very low s values, and therefore tend to be lexicalized into elementary trees. This is expected behaviour, as these categories select strongly for the words they modify and some (DT, verbs) must agree with their arguments in number and tense. Conversely particles (RP), modal verbs (MD) and possessive particles (PRP$) have high s values, and are therefore rarely lexicalized. This is reasonable for MD and PRP$, which tend to be exchangeable for one another without rendering the sentence ungrammatical (e.g., 'his' can be substituted for 'their' and 'should' for 'can'). However, particles are highly selective for the verbs they modify, and therefore should be lexicalized by the model (e.g., for 'tied in', we cannot substitute 'out' for 'in'). We postulate that the model does not learn to lexicalise verb-particle constructions because they are relatively uncommon, they often occur with different tenses of verb and often the particle is not adjacent to the verb, therefore requiring large elementary trees to cover both words. The phrasal types all have similar s values except for VP, which is much more likely to be lexicalized. This allows for elementary trees combining a verb with its subject noun-phrase, which is typically not part of the VP, but instead a descendent of its parent S node. Finally, the s values for the binarized nodes (denoted with the -BAR suffix) on the far right of Figure 8b are all quite low, encoding a preference for the model to reconstitute binarized productions into their

(a) PYP hyper-parameters, $\mathbf{a}, \mathbf{b}$



(b) Substitution hyper-parameters, $\mathbf{s}$

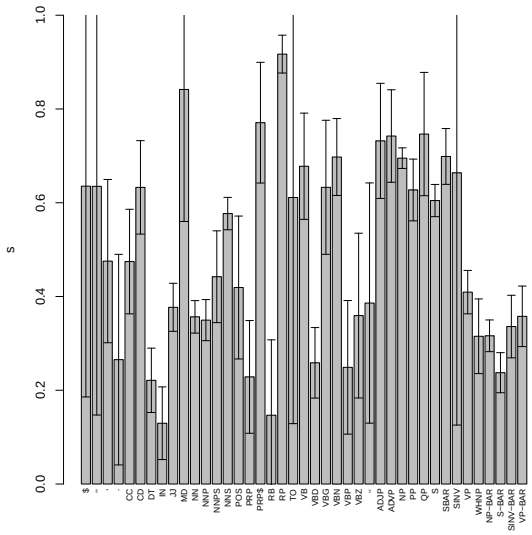


Figure 8: Inferred hyper-parameter values. Points (left) or bars (right) show the mean value with error bars indicating one standard deviation, computed over the final samples of five sampling runs. For legibility, a) has been limited to common phrasal nonterminals, while b) also shows preterminals and binarized nonterminals (those with the -BAR suffix). Note that in a) b is plotted on a log-scale.

original form. Some of the values have very high variance, for example, \$, which is due to their rewriting as a single string with probability 1 under P_C (or a small set of strings and a low entropy distribution), thus making the hyper-parameter value immaterial.

For comparison, we trained the Berkeley split-merge parser (Petrov et al., 2006) on the same data and decoded using the Viterbi algorithm (MPD) and expected rule count (MER, also known as MAX-RULE-SUM). We ran two iterations of split-merge training, after which the development F1 dropped substantially (in contrast, our model is not fit to the development data). The result (denoted $\tau = 2$) is an accuracy slightly below that of our model. To be fairer to their model, we adjusted the unknown word threshold to their default setting, that is, to apply to word types occurring fewer than five times (denoted $\tau = 5$).[16] Note that our model's performance degraded using the higher threshold, which impedes the model's ability to learn highly lexicalized fragments. The grammar sizes are not strictly comparable, because we are comparing different types of grammar. For our TSG models we report the number of CFG productions in the transformed MAP PCFG, in which non-zero count TSG rules typically rewrite as many CFG rules[17] and CFG rules from the base distribution are replicated up to four times. Nevertheless the trend is clear: our model produces similar results to a state-of-the-art parser, and does so using a similar sized grammar. With additional rounds of split-merge training the Berkeley grammar grows exponentially larger (200K rules after six iterations). Our TSG grammar is also far smaller than the full DOP grammar induced from this data set, which extracts every possible TSG rule from the training set with no size limit, and has approximately 700K rules.

6.2 Full Treebank

We now train the model on the full training partition of the Penn treebank, using sections 2–21 (see Table 2 for corpus statistics). We initialise the sampler using a converged model from the end of a sampling run on the small data set and run the blocked Metropolis Hastings sampler for 20,000 iterations. The MAP PCFG approximation had 156k productions and training took 1.2 million seconds in total or 61 seconds per iteration.[18] We repeated this three times and present the averaged results in Table 4.

The TSG models far surpass the MAP PCFG baseline, while the relative orderings of the different parsing algorithms corroborate our earlier evaluation on the small training set. The model outperforms the most comparable DOP result, although the numbers are not strictly comparable as Zuidema (2007) used an enriched nonterminal set for testing. However, our results are still well below state-of-the art parsers, and even underperform the Berkeley parser when it is restricted to the same preprocessing steps for rare tokens and binarization as we used (results labelled *restricted* in Table 4). But we must bear in mind that these parsers have benefited from years of tuning to the Penn-treebank, where our model is much

16. The Berkeley parser has a number of further enhancements that we elected not to use, most notably, a more sophisticated means of handling unknown words. These enhancements produce further improvements in parse accuracy, but could also be implemented in our model to similar effect.
17. The encoding of TSG rules could be made more compact by skipping the internal rewrite steps, instead directly rewriting the transformed root node as the rule's frontier. This would mean that each input TSG rule would produce only two rules in the transformed CFG. It would also affect the choice of parsing algorithm because the transformed grammar would no longer be binary.
18. Measured using a single core of an AMD Opteron 2.6GHz machine.

	≤ 40		all	
Parser	**F1**	**EX**	**F1**	**EX**
MLE PCFG	64.2	7.2	63.1	6.7
TSG PYP Viterbi	83.6	24.6	82.7	22.9
TSG PYP MPD	84.2	27.2	83.3	25.4
TSG PYP MPT	84.7	28.0	83.8	26.2
TSG PYP MER	85.4	27.2	84.7	25.8
DOP (Zuidema, 2007)			83.8	26.9
Berkeley parser (Petrov and Klein, 2007)	90.6		90.0	
Berkeley parser (restricted)	87.3	31.0	86.6	29.0
Reranking parser (Charniak and Johnson, 2005)	92.0		91.4	

Table 4: Full treebank testing results showing labelled F1 and exact match accuracy for sentences of up to 40 words, and for all sentences. The results of several treebank parsers are also shown (as reported in the literature, hence the missing values), representing a baseline (PCFG), systems similar to our own (DOP, Berkeley) and state-of-the-art (Berkeley, Reranking parser). Berkeley (restricted) uses simplified data preprocessing as compared to Berkeley; the simplified preprocessing is the same as used in our system, so provides a more fair comparison.

simpler and is largely untuned. We anticipate that careful data preparation, model tuning and improved inference algorithms would greatly improve our model's performance, bringing it closer to the state-of-the-art.

6.3 Sampling Strategy

Next we investigate which of the two sampling methods is more effective for training the model. Recall that in Section 4 we described a blocked sampler and a local sampler; these samplers differ in the number of variables that they update in every sampling step. We return to the small training sample for these experiments. Figure 9 shows how the log posterior over the training set varies with the sampling iteration and with time for the two different sampling algorithms. It is clear that the blocked MH sampler exhibits faster convergence in general than the local Gibbs sampler, despite being somewhat slower per iteration. The speed difference is fairly minor, amounting to roughly a 50% increase in time over the local sampler,[19] although on the full data set the time differential reduces to 19%. This difference is largely due to the cost of performing the grammar transformation, which could potentially be further optimised to reduce the gap.

Figure 9 shows the results for a number of different initialisations, using *minimal* elementary trees where every node is a substitution point (the CFG analysis), initialising the substitution variables uniformly at random (*even*), and using *maximal* elementary trees where no nodes are substitution points. The blocked sampler is more robust to starting

19. To account for the speed differential, the local samplers were run for 15k iterations and the blocked samplers for 10k iterations to produce Figure 9 and Table 5.

Sampling method	Initialisation	F1	σ_{F1}	Training time (s)
Local	minimal	78.2	0.50	44674
Local	even	78.3	0.31	43543
Local	maximal	78.5	0.51	44453
Block	minimal	78.5	0.18	46483
Block	even	78.6	0.35	46535
Block	maximal	78.6	0.38	39789

Table 5: Parsing performance and training time for the local versus blocked samplers with different initialisations. Results are shown on the development set using the MER parsing, reporting the mean F1 and standard deviation (σ_{F1}) from five independent runs. The blocked samplers were run for 10k iterations and the local sampler for 15k iterations iterations in order to allow all methods approximately the same running time.

point than the local sampler and converges faster in terms of iterations and total time in general. Interestingly, the blocked sampler converges faster with the maximal initialisation, which is due to the initialisation condition resulting in much smaller initial table counts, and therefore it is quite easy for the model to move away from that solution. However, with the minimal initialisation the counts begin with very high values, and therefore deviating from the initial solution will be much less likely. In contrast, the local sampler behaves in the opposite way with respect to initialisation, such that with the minimal initialisation it performs at or above the level of the blocked sampler. This is a surprising finding which contradicts our intuition about the mixing properties of the sampler and warrants further research.

In the above we have been comparing the log training posterior as a measure of the quality of the sampler, however it is not a given that a probable model is one with good parsing accuracy. Figure 10 shows that the posterior is highly correlated with generalisation F1, with a Pearson's correlation efficient of 0.95 on this data, and therefore improving the sampler convergence will have immediate positive effects on performance. This is corroborated in Table 5, which shows the F1 scores using the final sample for each initialisation condition. The blocked sampler out-performs the local sampler for all initialisation conditions and has lower lower variance. Moreover, the blocked sampler is less dependent on its initialisation, performing well independent of initialisation. In contrast, the local sampler performs well only with the maximal initialisation, with which it roughly equals the blocked sampler in terms of F1 and log posterior (see Figure 9).

6.4 Discussion

The previous sections show that our model performs far better than a standard PCFG trained on the same corpus; it is natural to ask what kinds of rules it is learning that allow it to do so well. Figure 11 shows the grammar statistics for a TSG model trained on the full treebank training set. This model has a total of 72955 rules with an aggregate count of 733755. Of these, only 46% are CFG rules. The TSG rules vary in height from one to nineteen with the majority between one and four. Most rules combine a small amount of

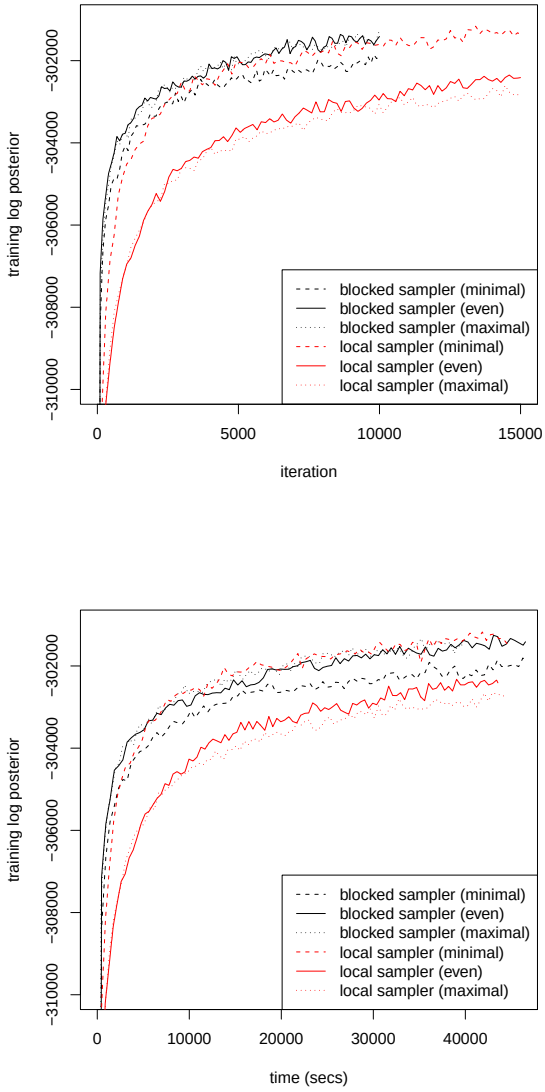


Figure 9: Training likelihood vs. iteration (left) or elapsed time (right). Both sampling methods were initialised in three different ways, *minimal* (all substitution variables set to 1, $\mathbf{x} = 1$), *even* ($x_d \sim \text{Uniform}(0, 1)$) and *maximal* ($\mathbf{x} = 0$).

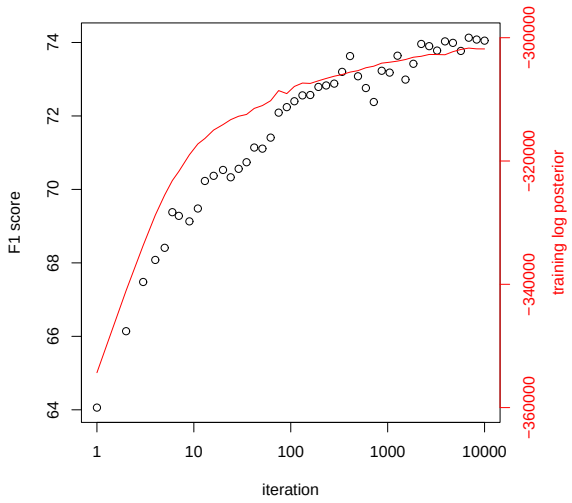


Figure 10: Likelihood and generalisation F1 are highly correlated. The black circles show the development F1 score (left axis) and the red line shows the training log-likelihood (right axis) during a Gibbs sampling run. The parsing results were obtained using Viterbi parsing with the MAP approximation grammar.

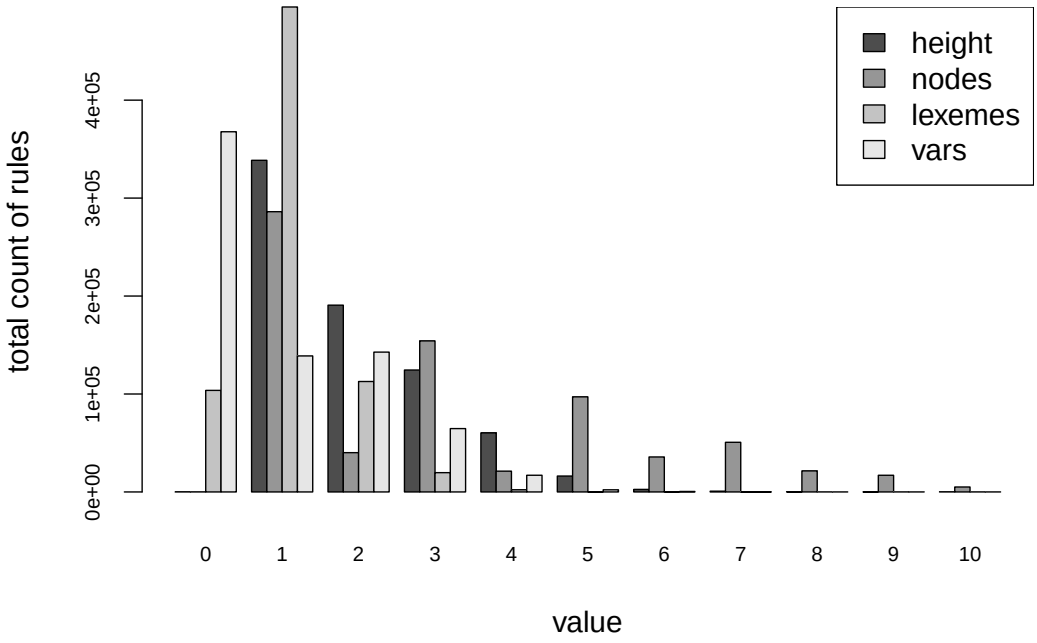


Figure 11: Grammar statistics for a TSG model trained on the full Penn treebank training set, showing the aggregate count for elementary trees of given height, number of nodes, terminals (lexemes) and frontier nonterminals (vars). An insignificant fraction of the rules had a height or number of nodes > 10; these have been truncated for display purposes.

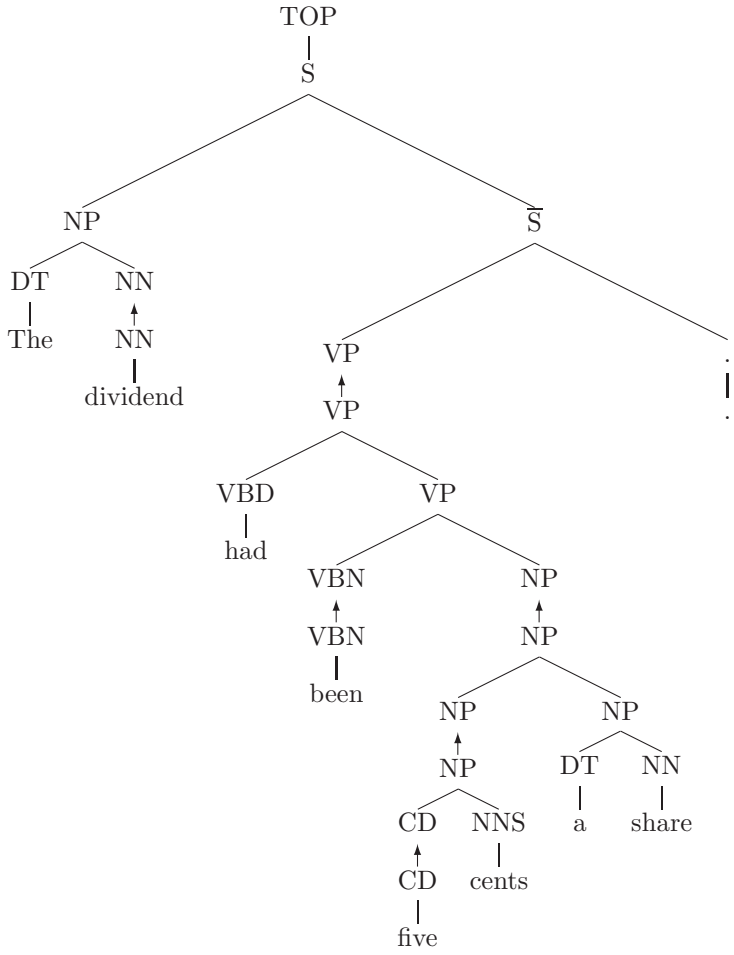


Figure 12: Inferred TSG structure for one of the training trees. Nonterminals shown with an over-bar (e.g., $\overline{\text{S}}$) denote a binarized sub-span of the given phrase type.

lexicalisation and a variable or two. This confirms that the model is learning local structures to encode, for example, multi-word units, subcategorisation frames and lexical agreement. The few very large rules specify full parses for sentences which were repeated in the training corpus. These complete trees are also evident in the long tail of node counts (up to 30; not shown in the figure) and counts for highly lexicalized rules (up to 11).

It is also instructive to inspect the inferred segmentations for individual trees in the training set. Figure 12 shows an example taken from the training set showing the learned derivation. Notice that the model has learned to use a large rule from the TOP node to capture the typical NP VP . sentence structure while lexicalizing the initial determiner, which is necessary to properly describe its capitalisation. It has also learnt a subcategorisation frame for `had` which specifies a VBN argument and an NP, and learnt the `<num> cents` and `NP a share` fragments, which are both common in the training corpus (Wall Street Journal articles).

To provide a better picture of the types of rules being learnt, Table 6 shows the top fifteen rules for three phrasal categories for the model trained on the full Penn treebank. We can see that many of these rules are larger than CFG rules, confirming that the CFG rules alone are inadequate to model the treebank. Two of the NP rules encode the prevalence of prepositional phrases headed by 'of' within a noun phrase, as opposed to other prepositions. Highly specific tokens are also incorporated into lexicalized rules.

The model also learns to use large rules to describe the majority of root node expansions (we add a distinguished TOP node to all trees). These rules mostly describe cases when the S category is used for a full sentence and includes punctuation such as the full stop and quotation marks. In contrast, the majority of expansions for the S category do not include any punctuation. The model has learnt to distinguish between the two different classes of S—full sentence versus internal clause—due to their different expansions.

Many of the verb phrase expansions have been lexicalized, encoding verb subcategorisation, as shown in Table 7. Notice that each verb here accepts only one or a small set of argument frames, indicating that by lexicalizing the verb in the VP expansion the model can find a less ambiguous and more parsimonious grammar. The lexicalized noun phrase expansions in Table 7 also show some interesting patterns, such as reconstituting binarized productions and lexicalizing prepositional phrase arguments. Of particular interest are the rules NP $\rightarrow$ CD (NN %) and NP $\rightarrow$ (NNP Mr.) NNP, in which the lexicalized item has a common tag but a very specific function. These rules are considerably more expressive than their CFG counterparts. Overall these results demonstrate that the model uses deep elementary trees to describe regular patterns in the data, thereby out-performing a simple PCFG. Despite having an infinite space of grammar productions, the induced grammars are compact, with the majority of probability mass on a small number of individually small productions.

7. Unsupervised Dependency Induction Experiments

Having considered the application of our non-parametric TSG model to supervised parsing, in this section we present an empirical evaluation for our model on the task of unsupervised parsing. Our approach to the unsupervised task differs from the supervised one in several ways. Two differences result from the fact that parse trees are not observed during training: first, we cannot use the local Gibbs sampler presented above, leaving the blocked sampler as our only option. Second, we cannot directly estimate P_{C} from the training data, so we must extend the model with an extra level of hierarchy (as described in Section 3) in order to induce P_{C}. A final difference in our unsupervised experiments is that we now focus on dependency grammar induction rather than phrase-structure induction; as outlined in Section 2.1, dependency grammars are a more feasible formalism for unsupervised grammar learning. Before presenting our experimental results, we first describe how to represent dependency grammars using our Bayesian TSG model.

To date, the most successful framework for unsupervised dependency induction is the Dependency Model with Valence (DMV, Klein and Manning, 2004). This approach induces a dependency grammar using a simple generative model in which the strings are said to be generated by a top-down process which recursively generates child dependents from each parent head word. This model has been adapted and extended by a number of authors (e.g.,

NP →
(DT the) $\overline{\text{NP}}$
NNS
NP ($\overline{\text{NP}}$ (CC and) NP)
JJ NNS
NP (PP (IN of) NP)
NP PP
(DT the) NN
DT ($\overline{\text{NP}}$ JJ NN)
NN NNS
(DT the) NNS
JJ $\overline{\text{NP}}$
(NP DT NN) (PP (IN of) NP)

VP →
VBD $\overline{\text{VP}}$
VBZ NP
VBD NP
VBZ $\overline{\text{VP}}$
VBP NP
VBP $\overline{\text{VP}}$
MD (VP VB NP)
(VBD said) (SBAR (S NP VP))
MD (VP VB $\overline{\text{VP}}$)
VP ($\overline{\text{VP}}$ (CC and) VP)
(VBZ is) ADJP
(VBD said) (SBAR (S (NP (PRP it)) VP))

PP →
(IN in) NP
(IN for) NP
(IN on) NP
(IN with) NP
TO NP
(IN at) NP
(IN from) NP
(IN by) NP
(TO to) NP
(IN of) NP
IN (S (VP VBG NP))
IN NP

ADJP →
JJ
(JJ able) S
RB JJ
RB $\overline{\text{ADJP}}$
(RB very) JJ
JJ ($\overline{\text{ADJP}}$ CC JJ)
ADJP ($\overline{\text{ADJP}}$ CC ADJP)
(RBR more) JJ
JJ PP
(NP CD (NNS years)) (JJ old)
(JJ UNK-LC)
(RB too) JJ

ADVP →
(RB also)
(RB now)
RB
(RB still)
(NP (DT a) (NN year)) (RBR earlier)
(RB already)
(RB UNK-LC-ly)
(RB again)
(RB just)
(RB then)
(RB never)
(RB currently)

TOP →	
(S NP ($\overline{\text{S}}$ VP (. .)))	
SINV	
(S (NP (DT The) $\overline{\text{NP}}$) ($\overline{\text{S}}$ VP (. .)))	
(S S ($\overline{\text{S}}$, $\overline{\text{S}}$))	
(S (CC But) $\overline{\text{S}}$)	
(S (PP (IN In) NP) ($\overline{\text{S}}$ (, ,) $\overline{\text{S}}$))	
(S (NP (DT The) NN) ($\overline{\text{S}}$ VP (. .)))	
(S (“ “) ($\overline{\text{S}}$ S ($\overline{\text{S}}$ (, ,) $\overline{\text{S}}$)))	
(S (NP NP ($\overline{\text{NP}}$, ($\overline{\text{NP}}$ NP ,))) ($\overline{\text{S}}$ VP (. .)))	
(S PP ($\overline{\text{S}}$ (, ,) ($\overline{\text{S}}$ NP ($\overline{\text{S}}$ VP (. .)))))	
(S (NP (PRP He)) ($\overline{\text{S}}$ VP .))	3085
(S (CC And) $\overline{\text{S}}$)	

S →
NP VP
(VP TO (VP VB NP))
(NP PRP) VP
(VP TO (VP VB $\overline{\text{VP}}$))
(VP TO (VP VB ($\overline{\text{VP}}$ NP PP)))
(VP TO (VP VB))
(VP TO (VP VB PP))
(VP VBG NP)
(VP VBG $\overline{\text{VP}}$)
(NP (PRP We)) VP
(NP (PRP it)) VP
(NP (PRP I)) VP

Table 6: Most frequent expansions for a selection of nonterminals. Counts were taken from the final sample of a model trained on the full Penn treebank.

NP →
(DT the) $\overline{\text{NP}}$
NP ($\overline{\text{NP}}$ (CC and) NP)
NP (PP (IN of) NP)
(DT the) NN
(DT the) NNS
(NP DT NN) (PP (IN of) NP)
(DT a) $\overline{\text{NP}}$
(PRP$ its) $\overline{\text{NP}}$
NP ($\overline{\text{NP}}$ (, ,) (SBAR WHNP (S VP)))
NP ($\overline{\text{NP}}$, ($\overline{\text{NP}}$ (SBAR WHNP (S VP)) (, ,)))
CD (NN %)
NP ($\overline{\text{NP}}$ (, ,) NP)
(NP NNP (POS 's)) $\overline{\text{NP}}$
(NNP Mr.) NNP
(PRP it)
(NP DT ($\overline{\text{NP}}$ JJ NN)) (PP (IN of) NP)
(DT a) ($\overline{\text{NP}}$ JJ NN)
NP (SBAR (WHNP (WDT that)) (S VP))
NP ($\overline{\text{NP}}$ (, ,) ($\overline{\text{NP}}$ NP (, ,)))
(NP (DT the) NN) (PP (IN of) NP)

VP →
(VBD said) (SBAR (S NP VP))
VP ($\overline{\text{VP}}$ (CC and) VP)
(VBD said) (SBAR (S (NP (PRP it)) VP))
VBD ($\overline{\text{VP}}$ (NP CD (NN %)) $\overline{\text{VP}}$)
VP ($\overline{\text{VP}}$, ($\overline{\text{VP}}$ (CC and) VP))
VP ($\overline{\text{VP}}$ (, ,) ($\overline{\text{VP}}$ (CC but) VP))
(VBD said) (SBAR (S (NP (PRP he)) VP))
(MD will) (VP VB $\overline{\text{VP}}$)
(VBD said) (SBAR (S (NP (DT the) NN) VP))
(VBD agreed) S
(VBZ is) (VP (VBN expected) S)
(VBP say) (SBAR (S NP VP))
MD ($\overline{\text{VP}}$ (RB n't) (VP VB $\overline{\text{VP}}$))
(VBZ says) (SBAR (S NP VP))
(VBP do) ($\overline{\text{VP}}$ (RB n't) (VP VB NP))
(MD will) (VP VB NP)
(VBZ plans) S
(VBD was) (VP VBN (PP (IN by) NP))
(VBD did) ($\overline{\text{VP}}$ (RB n't) (VP VB NP))
VP ($\overline{\text{VP}}$ (CC but) VP)

Table 7: Most frequent lexicalized expansions for noun and verb phrases. Forms of `to be` and `to have` dominate the VP expansions and consequently have been excluded.

Cohen and Smith, 2009; Headden III et al., 2009); these approaches currently represent the state-of-the-art for dependency induction. In this section, we present a method for dependency grammar induction that incorporates the basic intuitions of the DMV, but is also capable of modelling larger units than just single parent-child dependency relations. We approach the problem by representing dependency structures using the formalism of a CFG, and then applying our model (Section 3) to learn a TSG based on that CFG.

A dependency grammar can be represented in a CFG by labelling constituents with their head word, and encoding each dependency edge between a head and a child word in the grammar productions. This grammar has productions of the form $S \rightarrow H$ (word H heads the sentence), $H \rightarrow C\ H$ (C is a left child of H) and $H \rightarrow H\ C$ (C is a right child of H), where S is the start nonterminal and the H and C denote head and child nonterminals, respectively, which are both drawn from the same alphabet as the terminals (in our case part-of-speech tags). Unfortunately parsing with this representation is inefficient, having an asymptotic time complexity of $\mathcal{O}(|\mathbf{w}|^5)$.[20] The complexity can be improved to $\mathcal{O}(|\mathbf{w}|^3)$ by replicating (splitting) each terminal and processing all left and right dependents of each head word separately (Eisner, 2000). This is illustrated in Figure 13 where the leaves are all replicated with the l and r subscripts, while the spans defined by the tree structure denote the left and right dependents of each head word. Here we employ the *fold-unfold* representation (Johnson, 2007) that generalises Eisner's (2000) split-head parsing algorithm by defining an equivalent CFG under which standard inference methods can be used. Table 8 shows the CFG grammar for the DMV model (CFG-DMV), while Figure 13 shows the derivation in this grammar for the example sentence in Figure 2. The key insight to understanding the nonterminals in this grammar is that the subscripts encode the terminals at the boundaries of the span dominated by that nonterminal. For example the nonterminal L_H encodes that the right most terminal spanned by this constituent is H (and the reverse for ${}_H\mathrm{R}$), while ${}_A\mathrm{M}_B$ encodes that A and B are the left and right terminals of the span. The superscripts * and ^{1} denote the valency of the head: both indicate that the head has at least one attached dependent in the specified direction, with ^{1} indicating that the head will continue to attach more children. The time complexity of inference in this grammar is only $\mathcal{O}(|\mathbf{w}|^3)$ because each span in the parse chart bounded by terminals A and B can only contain nonterminal labels from the set $\{\mathrm{L}_B,\ \mathrm{L}^*_B,\ \mathrm{L}^1_B,\ {}_A\mathrm{R},\ {}_A\mathrm{R}^*,\ {}_A\mathrm{R}^1,\ {}_A\mathrm{M}_{B^*},\ {}_{A^*}\mathrm{M}_B,\ \mathrm{S}\}$. Consequently the grammar constant is fixed rather than quadratic in the sentence length.

We apply our TSG model to unsupervised dependency grammar induction using the CFG-DMV as the underlying grammar representation. Our model is capable of learning tree fragments which combine multiple adjacent CFG productions, affording considerable additional modelling power above that of a PCFG. Thereby the model can learn to condition dependency links on the valence. For example by combining $\mathrm{L}_{NN} \rightarrow \mathrm{L}^1_{NN}$ and $\mathrm{L}^1_{NN} \rightarrow \mathrm{L}_{DT\ DT}\mathrm{M}_{NN^*}$ rules into an elementary tree the model can describe that the left-most child of a noun (NN) is a determiner (DT). Moreover, the model can learn groups of dependencies that occur together by conjoining multiple L^1_H or ${}_H\mathrm{R}^1$ nonterminals. This can represent, for example, the complete preferred argument frame of a verb.

20. This is a result of the usual CYK complexity of $\mathcal{O}(G^2|\mathbf{w}|^3)$, and the grammar constant G being equal to the number of terminals $|\mathbf{w}|$ in the sentence.

CFG Rule	**DMV Distribution**	**Description**
$\mathrm{S} \rightarrow \mathrm{L}_{H}\ {}_{H}\mathrm{R}$	$p(root = H)$	The head of the sentence is H.
$\mathrm{L}_{H} \rightarrow \mathrm{H}_{l}$	$p(STOP \vert dir = L, head = H, val = 0)$	H has no left children.
$\mathrm{L}_{H} \rightarrow \mathrm{L}^{1}_{H}$	$p(CONT \vert dir = L, head = H, val = 0)$	H has at least one left child.
$\mathrm{L}^{*}_{H} \rightarrow \mathrm{H}_{l}$	$p(STOP \vert dir = L, head = H, val = 1)$	H has no more left children.
$\mathrm{L}^{*}_{H} \rightarrow \mathrm{L}^{1}_{H}$	$p(CONT \vert dir = L, head = H, val = 1)$	H has another left child.
${}_{H}\mathrm{R} \rightarrow \mathrm{H}_{r}$	$p(STOP \vert dir = R, head = H, val = 0)$	H has no right children.
${}_{H}\mathrm{R} \rightarrow {}_{H}\mathrm{R}^{1}$	$p(CONT \vert dir = R, head = H, val = 0)$	H has at least one right child.
${}_{H}\mathrm{R}^{*} \rightarrow \mathrm{H}_{r}$	$p(STOP \vert dir = R, head = H, val = 1)$	H has no more right children.
${}_{H}\mathrm{R}^{*} \rightarrow {}_{H}\mathrm{R}^{1}$	$p(CONT \vert dir = R, head = H, val = 1)$	H has another right child.
$\mathrm{L}^{1}_{H} \rightarrow \mathrm{L}_{C}\ {}_{C}\mathrm{M}_{H^{*}}$	$p(C \vert dir = L, head = H)$	C is a left child of H.
${}_{H}\mathrm{R}^{1} \rightarrow {}_{H^{*}}\mathrm{M}_{C}\ {}_{C}\mathrm{R}$	$p(C \vert dir = R, head = H)$	C is a right child of H.
${}_{C}\mathrm{M}_{H^{*}} \rightarrow {}_{C}\mathrm{R}\ \mathrm{L}^{*}_{H}$	$p = 1$	Structural rule.
${}_{H^{*}}\mathrm{M}_{C} \rightarrow {}_{H}\mathrm{R}^{*}\ \mathrm{L}_{C}$	$p = 1$	Structural rule.

Table 8: The CFG-DMV grammar schema. Note that the actual CFG is created by instantiating these templates with part-of-speech tags observed in the data for the variables H and C. Valency (*val*) can take the value 0 (no attachment in direction *dir*) and 1 (one or more attachment). L and R indicates child dependents left or right of the parent; superscripts encode the stopping and valency distributions, X^{1} indicates that the head will continue to attach more children and X^{*} that it has already attached a child.

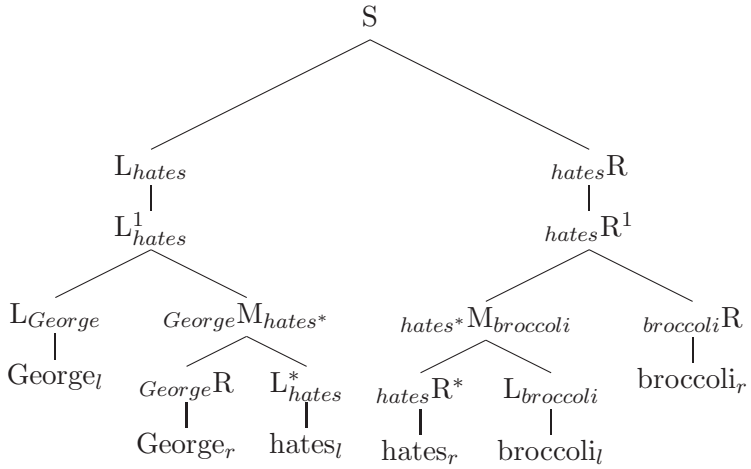


Figure 13: The CFG-DMV derivation for the example sentence *George hates broccoli*. The dependency parse for this sentence is given in Figure 2.

Partition	Sections	Words	Sentences
training$_{\leq 10}$	2–21	42505	6007
training$_{\leq 15}$	2–21	132599	12880
development$_{\leq 10}$	22	1805	258
test$_{\leq 10}$	23	2649	398
test$_{\leq 15}$	23	16591	1286
test$_{\leq \infty}$	23	49368	2416

Table 9: Corpus statistics for the training and testing data for the TSG-DMV model. All models are trained on the gold standard part-of-speech tags after removing punctuation.

7.1 Experiments

We perform inference for the TSG-DMV model by drawing 1000 samples using the blocked Metropolis-Hastings sampler described in Section 4.2 and evaluate the model using the final sample. Given that we do not observe parse trees in training, we cannot use the local Gibbs sampler as it only allows the sampling of the segmentation of a fixed tree, not the tree itself. In order to parse sentences in the test set we use the Viterbi algorithm to find the maximum probability parse under the MAP grammar (see Section 5). All hyperparameters, $\mathbf{a}, \mathbf{b}$ and $\mathbf{s}$, are sampled after every ten full samples over the training set.

A final and important consideration is the initialisation of the sampler. Klein and Manning (2004) emphasised the importance of the initialiser for achieving good performance with their model. We employ Klein and Manning's *harmonic* initialiser which defines a PCFG in which all words have the same valency distribution and probability of being the sentence head, while the probability of a head word attaching to a child word is inversely proportional to the average distance between these words in the training corpus. To obtain the initial derivations for the sampler we take the Viterbi derivations under this PCFG.

We follow the standard evaluation regime for DMV style models by performing experiments on the text of the WSJ section of the Penn. Treebank (Marcus et al., 1993). The corpus statistics are reported in Table 9. Like previous work we pre-process the training and test data to remove the words and punctuation, training our models on the gold-standard part-of-speech tags.

It is difficult for an unsupervised model to learn from long training sentences as their structure is highly ambiguous, and therefore the majority of DMV based approaches have been trained on sentences restricted in length to ≤ 10 tokens. This has the added benefit of decreasing the runtime for experiments. We present experiments with this training scenario, plus an additional experiment where we increase the length cutoff to ≤ 15. For the ≤ 15 experiment we start by sampling only sentences up to length 10, then gradually relax this length cutoff until we are sampling all sentences up to length 15 after 900 samples.[21] The training data is taken from sections 2-21, while section 23 is used for evaluation (see Table 9). An advantage of our sampling based approach over previous work is that we infer all the hyperparameters; consequently there is no need to tune on the development set (section 22).

21. This training method is similar in spirit to the Baby Steps algorithm (Spitkovsky et al., 2010).

Model	Initialiser	Directed Attachment Accuracy % on Section 23		
		$\|\mathbf{w}\| \leq 10$	$\|\mathbf{w}\| \leq 20$	$\|\mathbf{w}\| \leq \infty$
Attach-Right	-	38.4	33.4	31.7
EM (Klein and Manning, 2004)	Harmonic	46.1	39.9	35.9
Dirichlet (Cohen et al., 2009)	Harmonic	46.1	40.6	36.9
LN (Cohen et al., 2009)	Harmonic	59.4	45.9	40.5
SLN, TIE V&N (Cohen and Smith, 2009)	Harmonic	61.3	47.4	41.4
DMV (Headden III et al., 2009)	Random	$55.7_{\sigma=8.0}$	-	-
DMV smoothed (Headden III et al., 2009)	Random	$61.2_{\sigma=1.2}$	-	-
EVG smoothed (Headden III et al., 2009)	Random	$65.0_{\sigma=5.7}$	-	-
L-EVG smoothed (Headden III et al., 2009)	Random	$\mathbf{68.8}_{\sigma=4.5}$	-	-
Less is More WSJ15 (Spitkovsky et al., 2010)	Harmonic	56.2	48.2	44.1
Leap Frog WSJ45 (Spitkovsky et al., 2010)	Harmonic	57.1	48.7	45.0
Adaptor Grammar (Cohen et al., 2010)	Harmonic	50.2	-	-
TSG-DMV	Harmonic	$65.9_{\sigma=2.4}$	$58.3_{\sigma=2.3}$	$53.1_{\sigma=2.4}$
TSG-DMV WSJ15	Harmonic	$66.4_{\sigma=1.7}$	$\mathbf{58.5}_{\sigma=1.7}$	$\mathbf{53.4}_{\sigma=1.8}$
Supervised MLE (Cohen and Smith, 2009)	-	84.5	74.9	68.8

Table 10: Head attachment accuracy for our two TSG-DMV models (highlighted), and many other high performing models.

The models are evaluated in terms of head attachment accuracy (the percentage of correctly predicted head dependency links for each token in the test data), on three subsets of the testing data. Although unsupervised models are better learnt from a corpus of short rather than long sentences, they must still be able to parse long sentences. The most commonly employed test set mirrors the training data by only including sentences ≤ 10, however we also include results for sentences ≤ 20 and the whole test set with no length restriction. As we are using MCMC sampling the result of any single run is non-deterministic and will exhibit a degree of variance. Our reported results are the mean and standard deviation (σ) from 40 sampling runs.

7.2 Discussion

Table 10 shows the head attachment accuracy results for our TSG-DMV, along with those of several other competitive models. Our model performs very well in comparison to the others; in particular it achieves the highest reported accuracy on the full test set by a considerable margin. On the $|\mathbf{w}| \leq 10$ test set the TSG-DMV is second only to the L-EVG model of Headden III et al. (2009). The L-EVG model extends DMV by adding additional lexicalisation, valency conditioning, interpolated back-off smoothing and a random initialiser. In particular Headden III et al. show that the random initialiser is crucial for

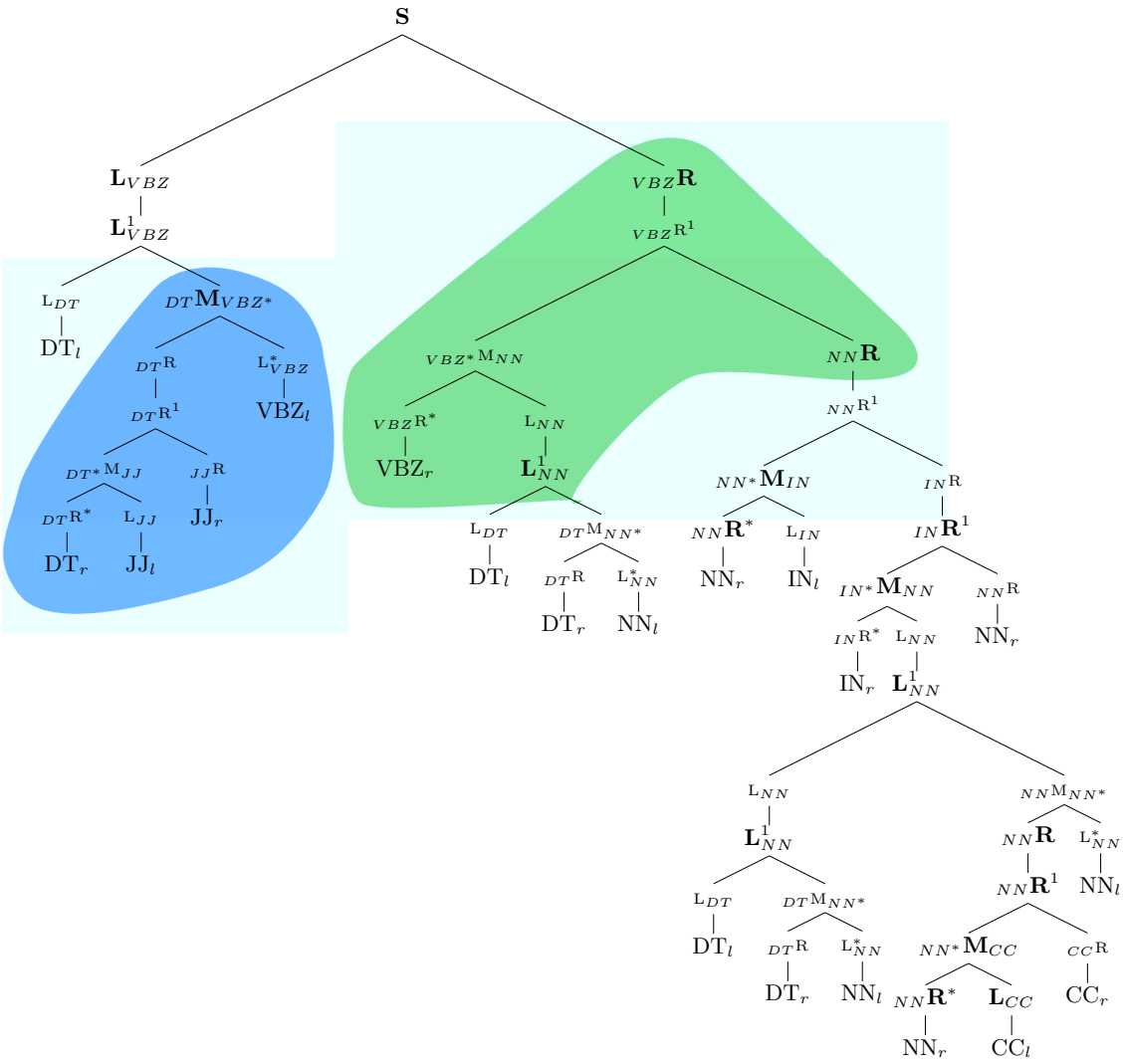
good performance, but their approach requires training 1000 models to select a single best model for evaluation and leads to considerable variance in test set performance. Our model exhibits considerably less variance than those induced using this random initialiser, suggesting that the combination of the harmonic initialiser and blocked MH sampling may be a more practical training regime. The recently-proposed Adaptor Grammar DMV model of Cohen et al. (2010) is similar in many ways to our TSG model, incorporating a Pitman Yor prior over units larger than CFG rules. As such it is surprising that our model performs significantly better than this model. We can identify a number of possible explanations for these results: the Adaptor Grammar model is trained using variational inference with the space of tree fragments truncated, while we employ a sampler which can nominally explore the full space of tree fragments; and the tree fragments in the Adaptor Grammar model must be complete subtrees (i.e., they don't contain variables), whereas our model can make use of arbitrary tree fragments. An interesting avenue for further research would be to extend the variational algorithm of Cohen et al. (2010) to our TSG model, possibly improving inference time while also allowing for the implementation to be more easily parallelised.

To illustrate the kinds of structures the model induces and the types of errors it makes, Figure 14 presents a representative example tree for a sentence from Section 22 of the WSJ. Though many of the elementary trees in this example resist an obvious linguistic explanation, on the right side of the derivation (highlighted in green) we see that the model has learnt to encode that the verb takes a single noun phrase as its object, while on the left (highlighted in blue) is a rule specifying the DT JJ subject dependent of the VBZ. This derivation is typical of the analyses produced by the model as it contains a number of dependency links which are inconsistent with the treebank. However we can see that the model has learnt to analyse the noun, verb and preposition phrases in a manner which is quite plausible, despite not matching the reference tree. In particular there would seem to be little obvious reason to prefer the treebank's analysis of the conjunction phrase ('either apathy or civility') over that produced by the unsupervised model. This highlights the difficulty in evaluating the output of unsupervised grammar induction algorithms against a treebank reference; in this instance it is clear that the analysis found by the model, despite its low accuracy, could be very useful for a downstream NLP application reliant on a dependency analysis.

For further analysis Tables 11 and 12 show the accuracy of the model at predicting the head for each tag type and the accuracy for dependencies spanning a given number of tokens. Clearly the model is far more accurate when predicting short dependencies, a result that is also reflected in the per-tag results. We also note that the model accurately predicts the head of the sentence 84% of the time, indicating an ability to capture the high level sentence structure. As mentioned above, conjunctions pose a particular difficulty with unsupervised models as the correct modelling of these remains a contentious linguistic issue. Nevertheless on conjunctions the model does achieve a reasonable level of agreement with the treebank.

Table 13 lists the most frequent TSG rules learnt by the model. The most frequent rule at the top of the table models noun phrases, encoding the fact that determiners have no children and attach as a left child to a phrase headed by a noun. It is interesting to see that our model has used a TSG rule to analyse noun phrases in a way consistent with the treebank, whereas the original DMV model preferred the opposite analysis of having DTs

(a) TSG-DMV representation. Large bold nodes indicate substitution points.



(b) Dependency tree representation. The red links below the sentence show where the treebank reference analysis differs from the predicted tree.

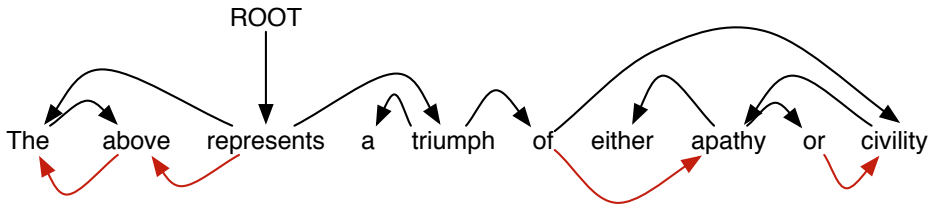


Figure 14: An example induced tree, shown as an elementary tree (a) and as a dependency tree (b). The sentence was taken from the development set: "$\texttt{The}_{\texttt{DT}}$ $\texttt{above}_{\texttt{JJ}}$ $\texttt{represents}_{\texttt{VBZ}}$ $\texttt{a}_{\texttt{DT}}$ $\texttt{triumph}_{\texttt{NN}}$ $\texttt{of}_{\texttt{IN}}$ $\texttt{either}_{\texttt{DT}}$ $\texttt{apathy}_{\texttt{NN}}$ $\texttt{or}_{\texttt{CC}}$ $\texttt{civility}_{\texttt{NN}}$".

Tag	Frequency	Accuracy	Tag	Frequency	Accuracy
NN	564	0.70	CC	62	0.77
NNP	366	0.67	VBG	48	0.71
NNS	302	0.74	POS	26	0.92
DT	292	0.81	MD	22	0.82
IN	292	0.59	JJR	20	0.60
JJ	266	0.67	PRP$	18	1.00
VBD	266	0.79	EX	12	1.00
CD	224	0.21	WP	12	0.17
RB	212	0.40	JJS	10	0.40
PRP	132	0.94	WDT	6	1.00
VBZ	118	0.88	RP	6	0.33
VBN	84	0.71	RBS	4	1.00
VBP	78	0.67	UH	4	0.50
TO	70	0.43			

Table 11: Head attachment accuracy stratified by child tag, as measured on the held-out development set (WSJ 22, $|\mathbf{w}| \leq 10$). The tags are sorted by frequency.

Link Distance	Precision	Recall	F1-Score
ROOT	0.84	0.84	0.84
1	0.68	0.72	0.70
2	0.61	0.53	0.57
3	0.56	0.46	0.51
4	0.47	0.52	0.49
5	0.27	0.35	0.30
6	0.44	0.57	0.50
7	0.33	0.38	0.35
8	0.25	0.12	0.17

Table 12: Performance on dependency links of varying distance. Precision, recall and f-score on the WSJ Section 22 ($|\mathbf{w}| \leq 10$) held-out set.

TSG-DMV Rules			Frequency
L_{NN}	$\rightarrow$	$(\mathrm{L}_{NN}\ (\mathrm{L}^1_{NN}\ \mathrm{L}_{DT}\ ({}_{DT}\mathrm{M}_{NN^*}\ {}_{DT}\mathrm{R}\ \mathrm{L}^*_{NN})))$	906
${}_{IN}\mathrm{R}$	$\rightarrow$	$({}_{IN}\mathrm{R}\ ({}_{IN}\mathrm{R}^1\ {}_{IN^*}\mathrm{M}_{NN}\ {}_{NN}\mathrm{R}))$	839
S	$\rightarrow$	$(\mathrm{S}\ (\mathrm{L}_{VBD}\ \mathrm{L}^1_{VBD)}\ {}_{VBD}\mathrm{R})$	707
${}_{JJ}\mathrm{M}_{NN^*}$	$\rightarrow$	$({}_{JJ}\mathrm{M}_{NN^*}\ {}_{JJ}\mathrm{R}\ (\mathrm{L}^*_{NN}\ \mathrm{NN}_l))$	600
${}_{NN^*}\mathrm{M}_{NN}$	$\rightarrow$	$({}_{NN^*}\mathrm{M}_{NN}\ {}_{NN}\mathrm{R}^*\ (\mathrm{L}_{NN}\ \mathrm{NN}_l))$	589
L^1_{NN}	$\rightarrow$	$(\mathrm{L}^1_{NN}\ \mathrm{L}_{DT}\ ({}_{DT}\mathrm{M}_{NN^*}\ {}_{DT}\mathrm{R}\ \mathrm{L}^*_{NN}))$	587
L_{NNP}	$\rightarrow$	$(\mathrm{L}_{NNP}\ (\mathrm{L}^1_{NNP}\ (\mathrm{L}_{NNP}\ \mathrm{NNP}_l)\ {}_{NNP}\mathrm{M}_{NNP^*}))$	540
L^*_{NN}	$\rightarrow$	$(\mathrm{L}^*_{NN}\ (\mathrm{L}^1_{NN}\ \mathrm{L}_{JJ}\ {}_{JJ}\mathrm{M}_{NN^*}))$	500
${}_{TO^*}\mathrm{M}_{VB}$	$\rightarrow$	$({}_{TO^*}\mathrm{M}_{VB}\ ({}_{TO}\mathrm{R}^*\ \mathrm{TO}_r)\ \mathrm{L}_{VB})$	437
${}_{NN}\mathrm{R}$	$\rightarrow$	$({}_{NN}\mathrm{R}\ ({}_{NN}\mathrm{R}^1\ {}_{NN^*}\mathrm{M}_{NN}\ ({}_{NN}\mathrm{R}\ \mathrm{NN}_r)))$	412
${}_{DT}\mathrm{M}_{NNS^*}$	$\rightarrow$	$({}_{DT}\mathrm{M}_{NNS^*}\ ({}_{DT}\mathrm{R}\ \mathrm{DT}_r)\ \mathrm{L}^*_{NNS})$	397
${}_{IN}\mathrm{R}$	$\rightarrow$	$({}_{IN}\mathrm{R}\ ({}_{IN}\mathrm{R}^1\ {}_{IN^*}\mathrm{M}_{NNS}\ ({}_{NNS}\mathrm{R}\ \mathrm{NNS}_r)))$	328
L_{NNS}	$\rightarrow$	$(\mathrm{L}_{NNS}\ (\mathrm{L}^1_{NNS}\ \mathrm{L}_{DT}\ {}_{DT}\mathrm{M}_{NNS^*}))$	326
${}_{IN}\mathrm{M}_{CD^*}$	$\rightarrow$	$({}_{IN}\mathrm{M}_{CD^*}\ ({}_{IN}\mathrm{R}\ \mathrm{IN}_r)\ (\mathrm{L}^*_{CD}\ \mathrm{CD}_l))$	302
${}_{NNS}\mathrm{M}_{VBD^*}$	$\rightarrow$	$({}_{NNS}\mathrm{M}_{VBD^*}\ ({}_{NNS}\mathrm{R}\ \mathrm{NNS}_r)\ \mathrm{L}^*_{VBD})$	296

Table 13: The fifteen most frequent TSG-DMV rules in the training data.

as the heads of noun phrases (Klein and Manning, 2004). Both results could be supported from a linguistic standpoint (Abney, 1987), but nevertheless it is a notable outcome that our more powerful model prefers to head noun phrases with nouns. Further down the table we see another interesting rule: ${}_{TO^*}\mathrm{M}_{VB} \rightarrow ({}_{TO^*}\mathrm{M}_{VB}\ ({}_{TO}\mathrm{R}^*\ \mathrm{TO}_r)\ \mathrm{L}_{VB})$. This rule specifies that a verb phrase headed by an infinitive attaches as the first child of the particle *to* on its left. Here the model has used the tree fragment to encode that the verb must be the first right child of the particle, an analysis both consistent with the treebank and expressing a bias against any form of split infinitive construction.

8. Conclusion

In this work we have presented a non-parametric Bayesian model for inducing tree substitution grammars in both supervised and unsupervised settings. By incorporating a structured prior over elementary rules our model is able to reason over the infinite space of all such rules, producing compact and simple grammars. In doing so, our model learns local structures for latent linguistic phenomena, such as verb subcategorisation and lexical agreement.

Our experimental results indicate that our model holds significant potential for a range of grammar induction tasks. In experiments using a treebank for training, we showed that the induced TSG grammars strongly out-perform standard PCFGs, and are comparable to a state-of-the-art parser on small data samples. While our results when training on the full treebank are well shy of the best available parsers, we have proposed a number of improvements to the model and the parsing algorithm that could lead to state-of-the-art performance in the future. Our second set of experiments removed the reliance on a treebank and showed that our TSG model achieves performance similar to the best recent models on sentences up to length 10, and outperforms all other models on longer sentences. This

result is particularly promising, since it demonstrates the possibility of successfully learning complex hierarchical models, beyond just CFGs, without supervision. We hope that our work will open the door to further research into inducing linguistically rich grammars, such as tree adjoining and categorial grammars, that have so far been considered too difficult to learn from raw strings.

References

Steven Paul Abney. *The English Noun Phrase in its Sentential Aspect*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1987.

David Aldous. Exchangeability and related topics. In *École d'été de probabilités de Saint-Flour, XIII—1983*, pages 1–198. Springer, Berlin, 1985.

Phil Blunsom and Trevor Cohn. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1204–1213, Boston, Massachusetts, October 2010.

Rens Bod. Using an annotated language corpus as a virtual stochastic grammar. In *11th National Conference on Artificial Intelligence*, pages 778–783, Washington D.C., USA, July 1993.

Rens Bod. The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars. In *Proceedings of the 7th conference on European chapter of the Association for Computational Linguistics*, pages 104–111, Dublin, Ireland, 1995.

Rens Bod. Combining semantic and syntactic structure for language modeling. In *Proceedings of the 6th International Conference on Spoken Language Processing*, pages 106–109, Beijing, China, 2000.

Rens Bod. An efficient implementation of a new DOP model. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, April 2003.

Rens Bod. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 865–872, Sydney, Australia, July 2006.

Rens Bod, Remko Scha, and Khalil Sima'an, editors. *Data-oriented parsing*. Center for the Study of Language and Information — Studies in Computational Linguistics. University of Chicago Press, 2003.

Glenn Carroll and Eugene Charniak. Two experiments on learning probabilistic dependency grammars from corpora. In *Proceedings of the AAAI Workshop on Statistically-Based Natural Language Processing Techniques*, San Jose, California, 1992.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180, Ann Arbor, Michigan, June 2005.

David Chiang and Daniel M. Bikel. Recovering latent information in treebanks. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 183–189, Taipei, Taiwan, 2002.

Alexander Clark. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the 2001 workshop on Computational Natural Language Learning*, pages 1–8, Toulouse, France, 2001.

Shay B. Cohen and Noah A. Smith. Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 74–82, 2009.

Shay B. Cohen, Kevin Gimpel, and Noah A. Smith. Logistic normal priors for unsupervised probabilistic grammar induction. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Lon Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 321–328. 2009.

Shay B. Cohen, David M. Blei, and Noah A. Smith. Variational inference for adaptor grammars. In *Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 564–572, 2010.

Trevor Cohn and Phil Blunsom. Blocked inference in Bayesian tree substitution grammars. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 225–230, Uppsala, Sweden, July 2010.

Trevor Cohn, Sharon Goldwater, and Phil Blunsom. Inducing compact but accurate tree-substitution grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 548–556, Boulder, Colorado, June 2009.

Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers, October 2000.

Thomas S. Ferguson. A bayesian analysis of some nonparametric problems. *Annals of Statistics*, 1(2):209–230, 1973.

Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. The infinite tree. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 272–279, Prague, Czech Republic, June 2007.

Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 673–680, Sydney, Australia, July 2006.

Joshua Goodman. *Parsing Inside-Out*. PhD thesis, Harvard University, 1998.

Joshua Goodman. Efficient parsing of DOP with PCFG-reductions. In Bod et al. (2003), chapter 8.

William P. Headden III, Mark Johnson, and David McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 101–109, Boulder, Colorado, June 2009.

Hemant Ishwaran and Lancelot F. James. Generalized weighted Chinese restaurant processes for species sampling mixture models. *Statistica Sinica*, 13:1211–1235, 2003.

Mark Johnson. The DOP estimation method is biased and inconsistent. *Computational Lingusitics*, 28(1):71–76, March 2002.

Mark Johnson. Transforming projective bilexical dependency grammars into efficiently-parsable cfgs with unfold-fold. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 168–175, Prague, Czech Republic, June 2007.

Mark Johnson. Using adaptor grammars to identify synergies in the unsupervised acquisition of linguistic structure. In *Proceedings of ACL-08: HLT*, pages 398–406, Columbus, Ohio, June 2008a.

Mark Johnson. Unsupervised word segmentation for Sesotho using adaptor grammars. In *Proceedings of the Tenth Meeting of ACL Special Interest Group on Computational Morphology and Phonology*, pages 20–27, Columbus, Ohio, June 2008b.

Mark Johnson and Sharon Goldwater. Improving nonparameteric bayesian inference: experiments on unsupervised word segmentation with adaptor grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 317–325, Boulder, Colorado, June 2009.

Mark Johnson, Thomas Griffiths, and Sharon Goldwater. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pages 139–146, Rochester, NY, April 2007a.

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 641–648. 2007b.

Aravind Joshi. Tree adjoining grammars. In Ruslan Mikkov, editor, *The Oxford Handbook of Computational Linguistics*, pages 483–501. Oxford University Press, Oxford, England, 2003.

Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Philadelphia, Pennsylvania, USA, July 2002.

Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 478–485, 2004.

Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

Percy Liang, Slav Petrov, Michael Jordan, and Dan Klein. The infinite PCFG using hierarchical Dirichlet processes. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 688–697, Prague, Czech Republic, June 2007.

Percy Liang, Michael I. Jordan, and Dan Klein. Type-based mcmc. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 573–581, Los Angeles, California, June 2010.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Igor′ A. Mel′čuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, 1988.

Bernard Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–172, 1994.

Radford Neal. Slice sampling. *Annals of Statistics*, 31:705–767, 2003.

Timothy J. O'Donnell, Noah D. Goodman, and Joshua B. Tenenbaum. Fragment grammar: Exploring reuse in hierarchical generative processes. Technical Report MIT-CSAIL-TR-2009-013, MIT, 2009.

Slav Petrov. Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Los Angeles, California, June 2010.

Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pages 404–411, Rochester, NY, April 2007.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006.

Jim Pitman. Exchangeable and partially exchangeable random partitions. *Probability Theory and Related Fields*, 102:145–158, 1995.

Jim Pitman. *Combinatorial Stochastic Processes*. Springer-Verlag, New York, 2006.

Jim Pitman and Marc Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *Annals of Probability*, 25:855–900, 1997.

Matt Post and Daniel Gildea. Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48, Suntec, Singapore, August 2009.

Detlef Prescher, Remko Scha, Khalil Sima'an, and Andreas Zollmann. On the statistical consistency of DOP estimators. In *Proceedings of the 14th Meeting of Computational Linguistics in the Netherlands*, Antwerp, Belgium, 2004.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. From Baby Steps to Leapfrog: How "Less is More" in unsupervised dependency parsing. In *Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759, 2010.

Fei Xia. *Automatic grammar generation from two different perspectives*. PhD thesis, University of Pennsylvania, 2002.

Andreas Zollmann and Khalil Sima'an. A consistent and efficient estimator for data-oriented parsing. *Journal of Automata, Languages and Combinatorics*, 10(2):367–388, 2005.

Willem Zuidema. Parsimonious data-oriented parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 551–560, Prague, Czech Republic, June 2007.