NAIST-IS-DD1261202

Doctoral Dissertation

Statistical Induction of Tree-Generating Grammars for Natural Language Parsing

Hiroyuki Shindo

September 18, 2013

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Hiroyuki Shindo

Thesis Committee:

Professor Yuji Matsumoto	(Supervisor)
Professor Satoshi Nakamura	(Co-supervisor)
Associate Professor Masashi Shimbo	(Co-supervisor)
Assistant Professor Kevin Duh	(Co-supervisor)

Statistical Induction of Tree-Generating Grammars for Natural Language Parsing*

Hiroyuki Shindo

Abstract

Syntactic analysis is a fundamental problem in natural language processing (NLP). Human languages are often ambiguous, however, statistical approaches assign a probability for each grammar rule, and thus they are capable of finding the most probable analysis in a sound manner. Statistical parsers require grammar rules and those probabilities for analyzing sentences, which are inferred from a collection of human-annotated syntax trees. Therefore, it is central for high-quality parsers that the modeling syntax trees with probabilistic grammars and learning the grammatical model from a collection of syntax trees.

A simple way of modeling syntax trees is to break down the tree into arbitrary size of smaller tree fragments and assign a probability to each fragment. This model is called Tree Substitution Grammar (TSG) and it has been successfully applied to many NLP tasks recently. However, the parsing performance based on the conventional TSG model is substantially less accurate than that of highly-optimized statistical parsers.

This thesis argues the major problems of TSG model, i.e., optional-obligatory distinction and context-free assumption, and challenges to overcome the problems by modeling richer and more powerful tree-generating grammars. We also propose efficient learning algorithms of our probabilistic grammars and show the proposed methods successfully achieve the state-of-the-art performance.

Concretely, we first present a tree insertion operator for TSG model to distinguish optional tree fragments from obligatory ones. We show the tree insertion operator is helpful not only for optional-obligatory distinction but reducing the

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1261202, September 18, 2013.

number of grammar rules and improving the parsing accuracy of the conventional TSG model.

Second, we propose a symbol-refined tree substitution grammar, which is an extension of the conventional TSG model where each symbol of tree fragments can be automatically subcategorized to address the problem of the strong context-free assumption of a TSG. Efficient learning algorithms of our grammar based on Markov Chain Monte Carlo methods are also presented. Our best model significantly outperforms the conventional TSG model and achieves state-of-the-art result in a standard parsing task.

Finally, we focus on the inference problem of probabilistic grammars and propose an efficient learning algorithm called pseudo blocked subtree sample. The proposed method improves the search efficiency by updating multiple variables at a time with low computational cost. Experiments show our proposed method improves the search efficiency of statistical grammar induction regardless of the amount of data.

Keywords:

natural language processing, parsing, tree substitution grammars, Bayesian non-parametrics, Pitman-Yor process

Acknowledgments

主指導教官の松本裕治教授には，終始暖かくご指導していただき，研究内容に関する様々な助言をいただきました．深く感謝致します．

中村哲教授には，お忙しい中審査委員をお引き受けいただき，公聴会などで研究に関する助言をいただきました．有難うございました．

新保仁准教授，Kevin Duh 助教には，お忙しい中審査委員をお引き受けいただき，研究に関する様々な助言をいただいたきました．また，研究室では日頃から声をかけていただき，非常に有意義な学生生活となりました．有難うございました．

その他，松本研究室の方々や秘書の北川祐子さんには，研究面や生活面で助けていただきました．有難うございました．

また，国立情報学研究所の宮尾祐介准教授，統計数理研究所の持橋大地准教授には，共同研究という形で研究に関する様々な助言をいただきました．深く感謝致します．今後ともよろしくお願いいたします．

早稲田大学の井上真郷教授には，私の学部・修士課程において研究指導をしていただきました．有難うございました．

また，学位取得の機会を与えていただいた NTT コミュニケーション科学基礎研究所の前田英作所長，上田修功前所長，山田武士部長，永田昌明グループリーダに深く感謝致します．

NTT コミュニケーション科学基礎研究所の藤野昭典さんには，入社当時から長い間研究指導をしていただき，論文の書き方や研究者としての心構えなど，様々なことを教えていただきました．有難うございました．

また，名前を挙げればきりがありませんが，日頃から研究面や生活面で様々なアドバイスをくださる塚田元さん，平尾努さん，鈴木潤さん，須藤克仁さん，西野正彬さん，安田宣仁さん，吉田康久さん，林克彦さん，西川仁さんに感謝致します．

最後に，これまで私を支えてくれた両親と姉妹に感謝致します．

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Syntactic analysis is a fundamental problem in natural language processing (NLP). The resulting analysis can be used in various ways to achieve high-quality NLP applications such as machine translation, automatic summarization and information extraction. For example, consider the word ordering of English-Japanese translation as follows:

English: I have a dream.
Japanese: I a dream have.

English follows S-V-O word order, that is, the subject comes first, the verb second, and the object third. On the other hand, Japanese follows S-O-V word order. Thus, when translating from one language to another, syntactic information such as subject, verb, and object is required for correct word reordering of the target language.

Each has his own grammar of natural language in his head, however, there is a widespread consensus in the field of linguistics that the syntactic information can be encoded by tree-structured forms such as phrase structure trees and dependency structure trees. Many human-annotated resources of syntax trees, such as Penn Treebank [30], have been developed so far, and a number of studies have tackled to explore the principle that predicts grammatical correctness of a sequence of words from those resources.

Over the past two decades, the mainstream research on natural language parsing has focused primarily on *statistical approaches* with the remarkable progress of

machine learning techniques and the large-scale development of natural language corpora. Statistical approaches can automatically learn grammar rules from examples with little human supervision, which has been proven to provide broader coverage and less maintenance effort than manual rule-based approaches. Besides, it provides a principled framework for *structural disambiguation*. For example, comsider the following sentence:

I saw the woman with a telescope.

This sentence has at least two syntactic analyses: one where the prepositional phrase: "with a telescope" modifies "the woman", or the other where it modifies "saw". Such ambiguity is a major concern in natural language parsing. Statistical approach assigns each interpretation a probability, thereby we can judge which one is the most probable.

Essentially, statistical parsing involves the following three steps.

1. Modeling:
   How to model a collection of syntax trees? In other words, how to formalize the probability distribution over syntax trees: $P\,(\text{trees}\,|\Theta\,)$?

2. Learning:
   How to learn parameters Θ of the model from a collection of syntax trees?

3. Decoding:
   How to parse an input sentence given the learned model?

Figure 1.1 shows an illustration of modeling, learning and decoding of statistical parsing.

Modeling syntax trees is formalized as a *probabilistic grammar*. A probabilistic grammar consists of a set of structural rules, i.e. tree fragments, that governs the composition of sentence, clauses, phrases and words. Each rule is assigned with a probability. Given a probabilistic grammar and a collection of syntax trees, the learning process finds the optimal parameters that fit the training data based on some criteria such as maximum-likelihood estimation. For decoding, statistical parser searches over a space of all candidate syntactic analyses according to the grammar rules. It then computes each candidate's probability and determines the most probable parse tree.

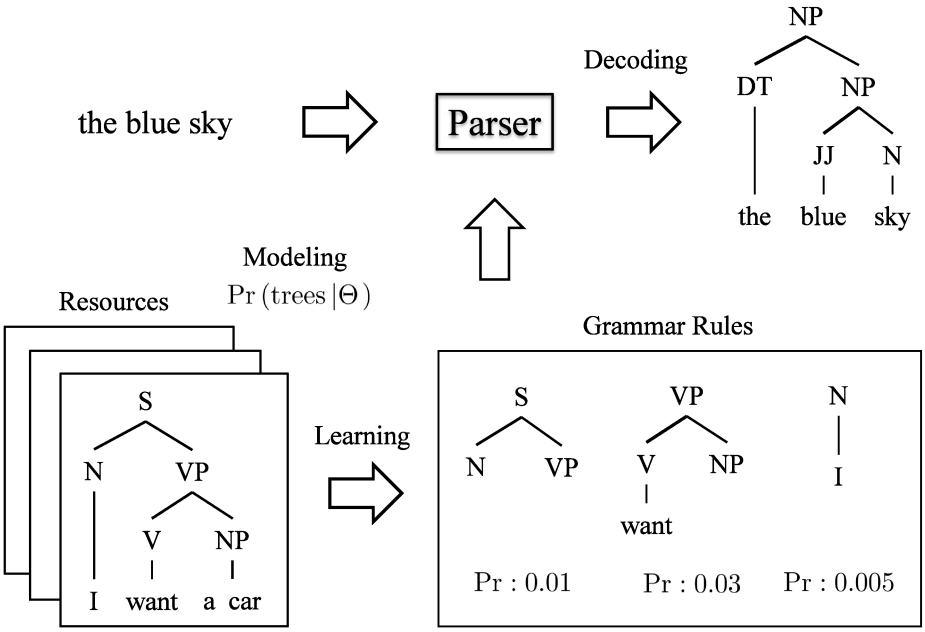


Figure 1.1: Illustration of modeling, learning and decoding of statistical parsing.

Modeling and learning are two sides of the same coin. Simple grammars that have limited number of rules and parameters are easy to handle, and the learning process is efficient since the search space over parameters is small. However, it is not much helpful for structural disambiguation due to the simplified independence assumption. On the other hand, complex but powerful grammars are able to capture rich linguistic features such as long-distance dependencies, however, the learning process becomes more difficult due to the larger search space. Therefore, if we manage to reduce the search space in a sound manner, we might have a more accurate parser with expressive grammars.

1.2 Contributions

This thesis aims to construct an accurate parser based on the statistical modeling of rich, powerful, and linguistically-motivated tree-generating grammars. We focus primarily on methods for modeling and learning of probabilistic grammars since the parsing performance is heavily dominated by them.

A simple way of modeling syntax trees is to break down the tree into smaller tree

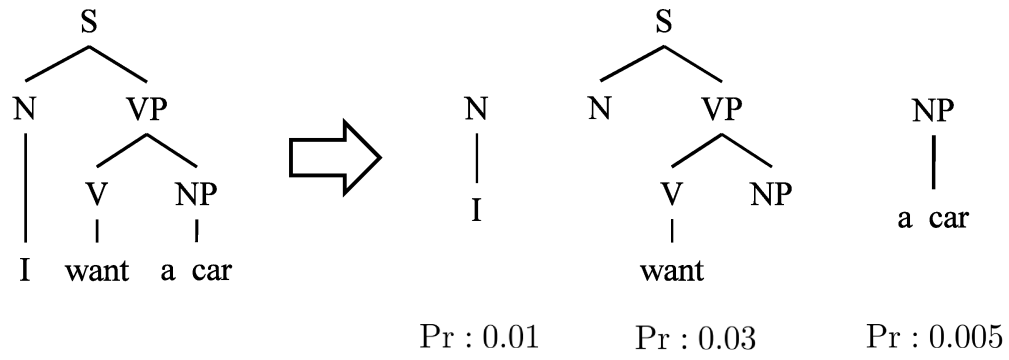


Figure 1.2: Example of breaking down syntax tree into grammar rules.

fragments and assign a probability to each fragment, as shown in Figure 1.2. The overall probability is simply defined as the product of each fragment's probability. Hopefully, these tree fragments capture linguistic features such as predicate-argument structure. For example, the tree fragment anchored by the verb "want" in Figure 1.2 represents the subject-verb-object pattern which takes "N" (noun) on the left-hand side as a subject and "NP" (noun phrase) on the right-hand side as an object.

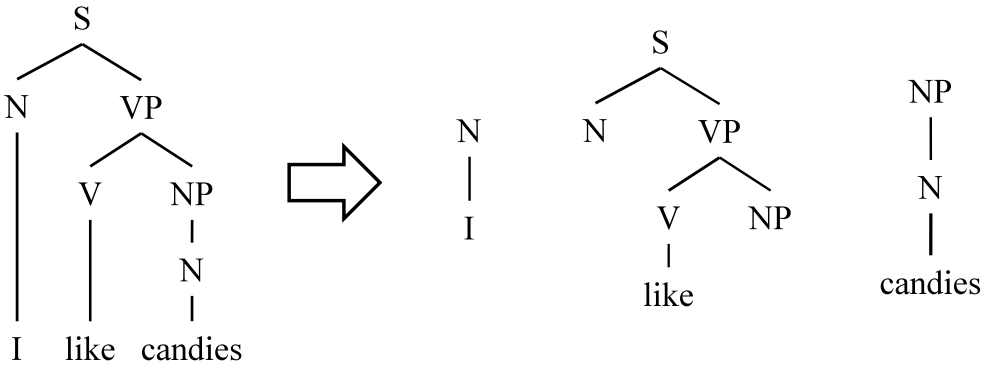
This model is called Tree Substitution Grammar (TSG), which is a generalization of the well-known Context-Free Grammar (CFG). Recently, statistical induction of probabilistic TSG from a collection of syntax trees has been an active area, and it has been applied to many NLP tasks such as word segmentation, parsing, and language modeling [23, 11]. However, it has been reported that the parsing performance based on TSG is substantially less accurate than that of highly-optimized statistical parsers [11, 38, 10].

In this thesis, we argue the problems of TSG model for syntactic parsing and propose more expressive grammars to overcome the problems. Furthermore, we show the proposed methods improve the parsing accuracy and achieve state-of-the-art performance.

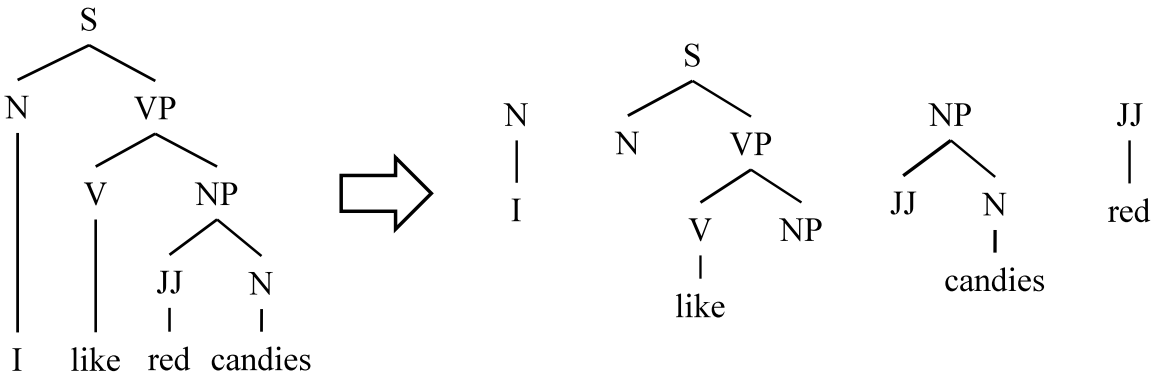
Here we bring up two major problems of the conventional TSG models.

1.2.1 Optional-Obligatory Distinction

The first problem is concerned with the distinction of optional and obligatory elements of syntax trees. As described before, TSG model breaks down the syntax



(a) "I like candies".



(b) "I like red candies".

Figure 1.3: Example syntax trees and grammar rules.

tree into smaller *continuous* tree fragments, thus it lacks the clear representation of optional or structurally dispensable elements (phrases, clauses, etc.).

Consider the following example:

I like candies.
I like red candies.

These two sentences are grammatically correct, thus the modifier "red" in the second sentence is optional. Figure 1.3 shows example syntax trees and TSG rules of the above two sentences. As shown in Figure 1.3, once the modifier is inserted to the syntax tree, the tree fragments anchored by "candies" differ from each other since the additional nonterminal symbols are required in each place at which an optional modification might occur. That is, TSG model must distinguish all of the optional patterns. Therefore, it is difficult to put a probability mass on the obligatory elements ("candies" in the example).

Essentially, the problem is that the TSG model assumes the grammar rules in the syntax tree must be continuous, which does not clearly model optional elements such as modifier and punctuation. We have deeper discussions about this problem in Chapter 3 and propose a probabilistic grammar that distinguishes optional and obligatory elements.

1.2.2 Context-Free Assumption

The second problem is concerned with the context-free assumption of TSG model. A probabilistic TSG model assumes the context-free assumption, that is, each tree fragment is generated conditioned only on the root nonterminal symbol. This assumption is poor and problematic when the annotated symbols are coarse in the syntax trees. For example, Penn Treebank data annotates all of noun phrase, such as "his approach" and "the markets", as the symbol "NP". Therefore, when the grammar rules are naively extracted from the resource, the subject and object noun phrases must share the same distribution, which is obviously helpless for structural disambiguation of subject phrase from object.

This problem has been pointed out for a long time in the previous work on statistical CFG parsing. Although learning a probabilistic CFG directly from a parse tree data via maximum likelihood estimation results in poor parsing performance, it has been shown that dividing coarse symbols into subcategories,

called *symbol refinement* approach, significantly improves parsing accuracy [22, 5, 12, 31, 35]. However, it is an open problem that how to introduce symbol refinement approach into the TSG model. There are at least two problems to be considered.

The first problem is concerned with modeling. Incorporating TSG model into symbol refinement approach makes the grammar rules complex and leads to aggravate data sparseness problem. Therefore, it becomes more difficult to estimate reliable probabilities of grammar rules from a limited amount of resources.

The second problem is concerned with learning. Symbol refinement approach makes the search space over parameters significantly larger. Therefore, efficient learning algorithms must be required for TSG model with symbol refinement.

We have deeper discussions about this problems in Chapter 4 and propose a method that handles data sparseness problem and efficient learning algorithms for TSG model with symbol refinement.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2: Preliminaries This chapter reviews the definition of conventional probabilistic grammars and the basics of nonparametric Bayesian models. Several learning algorithms based on Markov Chain Monte Carlo methods are also presented. Nonparametric Bayesian models and Markov Chain Monte Carlo methods are fundamental for the modeling and learning of TSG model.

Chapter 3: Statistical Induction of Tree Insertion Grammars In this chapter, we propose a probabilistic model of tree insertion grammars and an efficient learning algorithm to tackle the problem of optional-obligatory distinction. We apply our method to syntactic parsing and evaluate its performance.

Chapter 4: Statistical Induction of Symbol-Refined Tree Substitution Grammars In this chapter, we propose symbol-refined tree substitution grammars, a novel approach of incorporating TSG model into symbol refinement approach. We also propose a probabilistic model of the grammar and efficient

learning algorithm. We apply our method to syntactic parsing and evaluate its performance.

Chapter 5: Pseudo Blocked Subtree Sampler for Statistical Grammar Induction In this chapter, we focus on an inference problem of probabilistic grammars and propose an efficient learning algorithm based on Markov Chain Monte Carlo method for statistical grammar induction.

Chapter 6: Conclusion This chapter summarizes the thesis and discuss the future direction of the work.

Chapter 2

Preliminaries

This chapter provides the background knowledge to understand this thesis, specifically, context-free grammars, nonparametric Bayesian models, and Markov Chain Monte Carlo methods. Nonmarametric Bayesian models are required for modeling tree-structured data. Markov Chain Monte Carlo methods are inference algorithms used in the proposed methods.

2.1 Context-Free Grammars

Formally, a context-free grammar (CFG) is defined by a 4-tuple: $\mathcal{G} = (V_N, V_T, S, R)$ where

- V_N is a finite set of nonterminal symbols.
- V_T is a finite set of terminal symbols.
- $S \in V_N$ is the distinguished start symbol.

S	$\rightarrow$	NP	VP
NP	$\rightarrow$	I	
VP	$\rightarrow$	V	NP
NP	$\rightarrow$	DT	N
DT	$\rightarrow$	a	
N	$\rightarrow$	car	

Table 2.1: Context-free grammars of Figure 2.1.

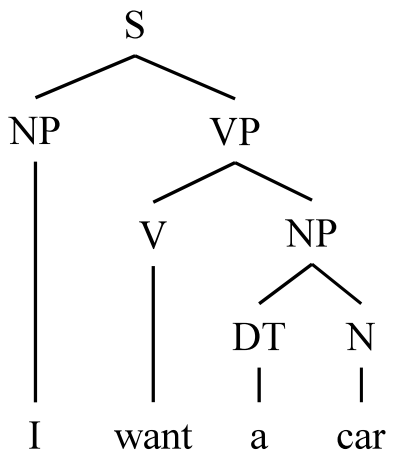


Figure 2.1: Example parse tree of "I want a car".

- R is a finite set of productions (a.k.a. rules).

Figure 2.1 shows an example parse tree and Table 2.1 shows its derivation rules based on context-free grammars. In a parse tree, every nodes is labeled with a nonterminal or terminal symbol. For example, S (sentence) stands for sentence, NP for noun phrase, and VP for verb phrase. The terminal symbols are usually lexical words such as "I", "have", and so on.

A *derivation* is a process of forming a parse tree with production rules of the grammar. It starts with the start symbol S and rewrites nonterminal symbols with a production rule until all leaf nodes become terminals. The production rules take the form of $X \to \alpha$ where $X \in V_N$ and $\alpha \in (V_N \cup V_T)^\star$. The asterisk represents the Kleene closure. It should be noted that given parse tree, the derivation of context-free grammars is uniquely determined by decomposing the parse tree as tree fragments of height = 1, while the derivation of other grammars such as Tree Substitution Grammars is not uniquely determined by a syntax tree since many derivations could produce the same parse tree.

2.1.1 Probabilistic Context-Free Grammars

A probabilistic context-free grammar (PCFG) assigns a probability to each grammatical rule. Table 2.2 shows an example of probabilistic context-free grammars. The probability of the rule $X \to \alpha$ is conditioned on the root symbol X as

Rule				Probability
S	$\rightarrow$	NP	VP	1.0
NP	$\rightarrow$	I		0.1
VP	$\rightarrow$	V	NP	0.3
NP	$\rightarrow$	DT	N	0.5
DT	$\rightarrow$	a		0.8
N	$\rightarrow$	car		0.1

Table 2.2: Probabilistic context-free grammars of Figure 2.1.

follows:

$$\Pr(X \rightarrow \alpha) = p(\alpha \,|X\,),$$

where

$$\begin{aligned} p(\alpha \,|X\,) &\geq 0, \\ \sum_{X \rightarrow \alpha \in R} p(\alpha \,|X\,) &= 1. \end{aligned}$$

The total probability of a derivation $\mathcal{D}$ is the product of the probabilities of its tree fragments as follows:

$$p(\mathcal{D}) = \prod_{X \rightarrow \alpha \in \mathcal{D}} p(\alpha \,|X\,). \tag{2.1}$$

Given a training data, the maximum likelihood estimate of

$$p_{\mathrm{MLE}}(\alpha \,|X\,) = \frac{\mathrm{count}(X \rightarrow \alpha)}{\sum_{\alpha'} \mathrm{count}(X \rightarrow \alpha')}. \tag{2.2}$$

2.2 Nonparametric Bayesian Models

This section provides an overview of nonparametric Bayesian models relevant to statistical grammar induction.

A "Bayesian" model is basically a probability model which assigns a probability distribution on its parameters. This distribution is called a *prior distribution*

and represents an uncertainness of the parameter. The parameter of a prior distribution is called a *hyperparameter*.

Let D be an observed data and $p(D|\Theta)$ be a probability distribution over D where Θ is a set of parameters of the distribution. A posterior distribution is calculated by Bayes' theorem as follows:

$$p(\Theta|D) = \frac{p(D|\Theta)\,p(\Theta)}{p(D)}, \tag{2.3}$$

where $p(\Theta)$ is a prior distribution.

A "nonparametric" model handles variable or possibly infinite number of parameters. This contrasts to a "parametric" model that handles fixed number of parameters. The nonparametric model has an advantage of making the model adaptable to the size of the observed data.

In statistical grammar induction, there are some reasons that nonparametric Bayesian model is preferable for modeling syntax trees. Firstly, most statistical-based grammatical models involve a probability distribution over tree-structured data. The number of tree fragments in the observed data is often unknown or possibly infinite, thus the nonparametric model is desirable. Secondly, Bayesian modeling enables us to encode a prior knowledge about grammars such as the number of grammatical rules should be as compact as possible even the data size increases.

2.2.1 Pitman-Yor Process

Pitman-Yor process [37] is a sort of nonparametric Bayesian model that is widely used in statistical grammar induction [11, 38, 41, 42, 48]. The Pitman-Yor process is a distribution over distributions over a probability space $\mathcal{X}$. We describe such distribution G over $\mathcal{X}$ as follows:

$$G \sim \mathrm{PYP}(d, \theta, H), \tag{2.4}$$

where d is a *discount* parameter with $0 \leq d < 1$, θ is a *strength* parameter with $\theta > -d$, and H is a *base distribution* over $\mathcal{X}$ with $\sum_x H(x) = 1$.

The draws from the Pitman-Yor process can be constructed by *Stick-breaking process* [40, 20]. This construction shows that G is a weighted sum of an infinite sequence of point masses. For $k = 1, 2, \ldots$, let us define v_k, π_k and ϕ_k as follows:

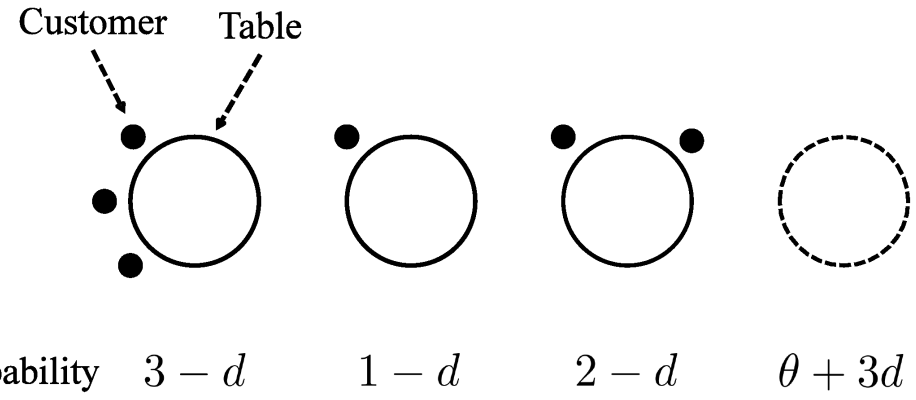


Figure 2.2: Illustration of Chinese restaurant process based on the Pitman-Yor process.

$$v_k \sim \text{Beta}\left(1-d, \theta+kd\right), \tag{2.5}$$

$$\pi_k = v_k \prod_{l=1}^{k-1}\left(1-v_l\right), \tag{2.6}$$

$$\phi_k \sim H. \tag{2.7}$$

Then G can be constructed as follows:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k}, \tag{2.8}$$

where δ_{ϕ_k} denotes the delta-Dirac mass located at ϕ. The distribution over $\pi = (\pi_1, \pi_2, \ldots)$is called the stick-breaking distribution and denoted as follows:

$$\pi \sim \text{GEM}\left(d, \theta\right).$$

Stick-breaking is a metaphor of recursively breaking a length-one stick at ratio v_k and calculating the actual length π_k based on the remaining stick.

2.2.1.1 Chinese Restaurant Process

An alternative way of representing Pitman-Yor process is the *Chinese restaurant process*. Unlike with the stick-breaking process, the Chinese restaurant pro-

cess describes the probability distribution over a sequence of draws $x_1, x_2, \ldots$ from G rather than G itself, which is achieved by integrating out G.

Consider the sequence of data $x_1, x_2, \ldots$ is generated according to G, and G follows the Pitman-Yor process as follows:

$$G \sim \mathrm{PYP}(d, \theta, H) \tag{2.9}$$

$$x \sim G \tag{2.10}$$

Integrating out G, the resulting distribution is

$$p(x_i = x \,|\mathbf{x}_{1:i-1}) = \frac{c_x - dt_x}{\theta + c_\cdot} + \frac{\theta + dt_\cdot}{\theta + c_\cdot} H(x) \tag{2.11}$$

Eq. 2.11 is interpreted as the "Chinese restaurant" metaphor. Figure 2.2 shows an illustration of the Chinese restaurant process. Consider a Chinese restaurant with unbounded number of tables. A customer $x_i\,(i = 1, 2, \ldots)$ visits the restaurant, and sits at a table one by one. The Chinese restaurant process assigns a probability distribution over the seating arrangement of the customers. The first customer always sits at the first table, and each subsequent customer sits at an already occupied table k with probability $c_k - d$ where c_k is the number of customers already sitting the table k, or a new table with probability $\theta + dt_\cdot$ where $t_\cdot$is the total number of occupied tables so far. If x_i sits at a new table, the table is labeled with the value x (called dish) according to the base distribution $H(x)$. t_x is the total number of tables labeled with x.

In summary, the probability of customer x_i choosing table k is

$$p(a(x_i) = k \,|a(\mathbf{x}_{1:i-1})) = \begin{cases} c_k - d & 1 \leq k \leq t_\cdot \\ \theta + dt_\cdot & k = t_\cdot + 1 \end{cases} \tag{2.12}$$

where $a(x)$ is the table assignment of the customer x. When a customer sits at a new table, a new x is drawn from the base distribution H, and the new table is labeled with x. The Pitman-Yor process can produce *rich get richer* statistics: a few types of x are generated many times while many are generated only a few times, which is empirically shown to be well-suited for natural language [44, 23].

In statistical grammar induction, Chinese restaurant process can be used for modeling a probability distribution over tree fragments. Let r be a CFG rule

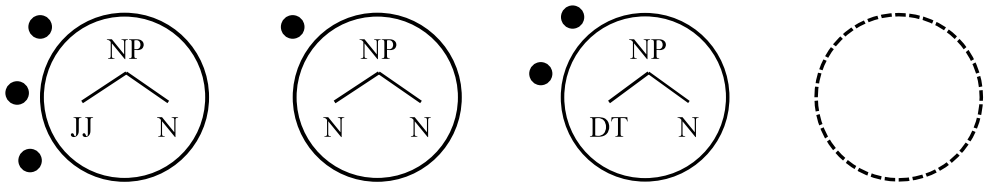


Figure 2.3: Illustration of Chinese restaurant process for CFG rules.

such as $A \rightarrow BC$. According to the eq. 2.11, the probability of generating r conditioned on the root symbol is given as follows:

$$p\left(r \left| \text{root}\left(r\right)\right.\right) = \frac{c_r - dt_r}{\theta + c_\cdot} + \frac{\theta + dt_\cdot}{\theta + c_\cdot} H\left(r \left| \text{root}\left(r\right)\right.\right) \tag{2.13}$$

where $\text{root}\left(r\right)$ is a root symbol of r and $H\left(r \left| A\right.\right)$ is a base probability of r. Figure 2.3 shows an illustration of Chinese restaurant process for CFG rules rooted with "NP". The probability of r depends on the c_r: the number of times r is previously generated, t_r: table counts labeled with r, and $H\left(r \left| \text{root}\left(r\right)\right.\right)$: the base probability of r. For example, when r is "NP→JJ NN" in Figure 2.3, $c_r = 3$, $t_r = 1$ and $c_\cdot = 6$. When the model generates r, a customer is added to either the table labeled with r or a new table. If the customer sits at the already occupied table, the values of c_r and $c_\cdot$ are incremented. If the customer sits at a new table, the table is labeled with r, c_r is set to be 1, and the values of t_r and $c_\cdot$ are incremented.

The base distribution H provides a backoff probability of r. One example of the base distribution is to decompose r into unary CFG rules as follows:

$$H\left(r \left| A\right.\right) = p\left(B \left| A\right.\right) \times p\left(C \left| A\right.\right)$$

where we assume r is a binary branching CFG rule: $A \rightarrow BC$. $p\left(B \left| A\right.\right)$ and $p\left(C \left| A\right.\right)$ may be simply defined as maximum likelihood estimate of $A \rightarrow B$ and $A \rightarrow C$, respectively.

2.2.2 Hierarchical Pitman-Yor Process

In natural language processing, probability smoothing is a critical technique for reliable estimate of probabilities since the observed data is sparse, which is known as a data sparseness problem. For example, in n-gram language models,

the probability distribution of the next word depends on the previous $n-1$ words. For any n-grams that have not explicitly been seen before gets the probability of 0.0, that is, those n-grams are estimated to be never happened in the future. To solve this zero-frequency problem, it is necessary to assign some of the total probability mass to unseen n-grams from the "back-off" models. Syntax modeling also requires probability smoothing from back-off models. To model syntax trees, probabilistic grammars consist of a distribution over tree fragments. It is often the case that large tree fragments appear only a few times in the observed data, thus the probability of tree fragment should be smoothed from the back-off distribution over smaller tree fragments.

Hierarchical Pitman-Yor process [44], an extension of Pitman-Yor process, encodes such back-off smoothing of probability distributions in a principled way. There are many variations to introduce dependencies among the Pitman-Yor processes in a hierarchical way, however, a simple approach is to assume a distribution is derived from the Pitman-Yor process and its base distribution is also derived from an another Pitman-Yor process as follows:

$$G_0 \sim \mathrm{PYP}(d_0, \theta_0, H) \tag{2.14}$$
$$G_1 \sim \mathrm{PYP}(d_1, \theta_1, G_0) \tag{2.15}$$
$$x \sim G_1 \tag{2.16}$$

As shown in eq. 2.11, the base distribution provides a smoothing probability for x.

2.2.2.1 Chinese Restaurant Franchise

The Chinese restaurant metaphor of Pitman-Yor process can be also extended to that of hierarchical Pitman-Yor process, called Chinese restaurant franchise. Figure 2.4 shows an illustration of Chinese restaurant franchise.

Consider hierarchical Chinese restaurants each of which has unbounded number of tables. A customer x enters the top-level Chinese restaurant and sits at some table. If he sits at a new table, the copy of customer is sent to the parent restaurant and he sits at some table at the parent restaurant. Otherwise, when he sits at already occupied table, no customer is sent to the parent restaurant. This process is recursively performed until the restaurant has no parent restaurants.

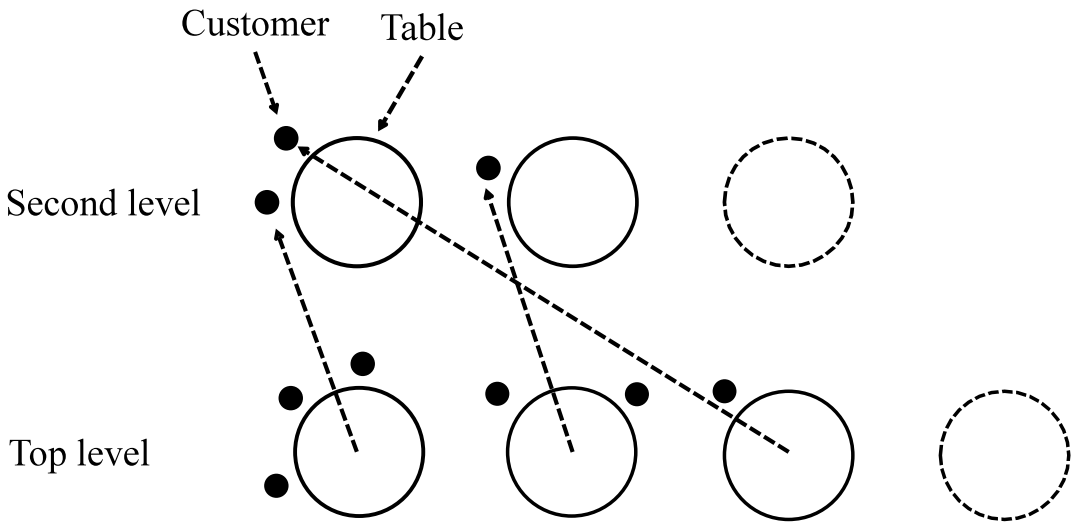


Figure 2.4: Illustration of Chinese restaurant franchise based on the Pitman-Yor process.

In Chinese restaurant franchise, every table at restaurants is associated with a customer in the parent restaurant. Thus, the number of tables at top-level restaurant equals to the number of customers sitting at the table labeled with x at the parent restaurant. That is,

$$t_x = \sum_i \delta_{x'_i = x} \tag{2.17}$$

where t_x is the number of tables labeled with x at top-level restaurant, and x'_i is a customer sent to the parent restaurant.

In statistical grammar induction, Chinese restaurant franchise can be used for modeling finer-grained probability distribution over tree fragments than single-layer Chinese restaurant process. Consider the distribution over tree fragments of arbitrary size rather than CFG rules (tree fragments of depth= 1). To estimate reliable probabilities of large tree fragments from limited amount of resources, we need a backoff scheme from large tree fragments to small ones. Figure 2.5 shows an illustration of Chinese restaurant franchise for large tree fragments rooted with "NP".

Let e be a tree fragment of arbitrary size. The probability of generating e is given analogous to the Chinese restaurant process:

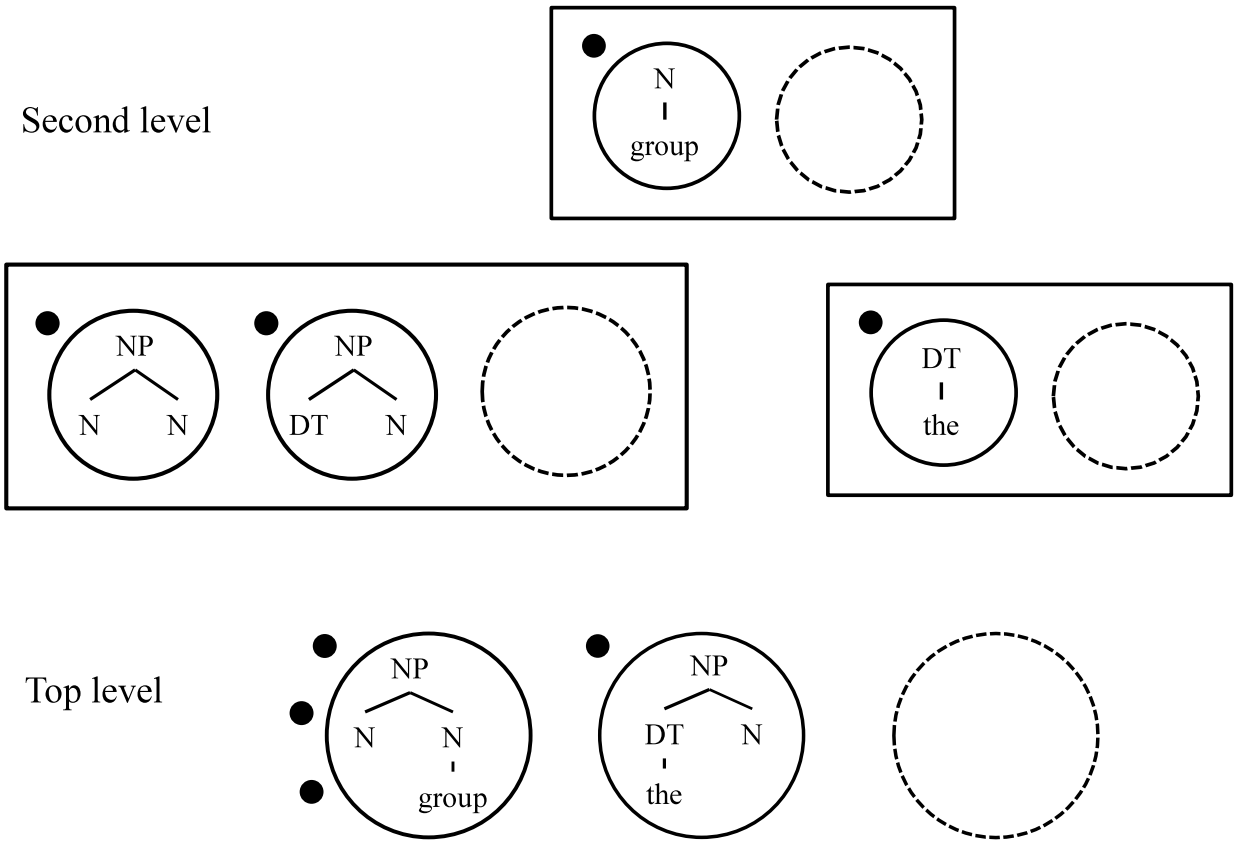


Figure 2.5: Illustration of Chinese restaurant franchise for tree fragments.

$$p\left(e\left|\text{root}\left(e\right)\right.\right) = \frac{c_e - dt_e}{\theta + c_\cdot} + \frac{\theta + dt_\cdot}{\theta + c_\cdot} G_1\left(e\left|\text{root}\left(e\right)\right.\right)$$

where G_1 is the base distribution over e. One example of the base probability is given by decomposition of e into CFG rules as follows:

$$G_1\left(e\left|\text{root}\left(e\right)\right.\right) = p_\$\left(\left|e\right|\right) \times \prod_{r \in \text{CFG}(e)} G_0\left(r\left|\text{root}\left(r\right)\right.\right)$$

where we assume $p_\$$ is a geometric distribution over the size of e, i.e., the number of CFG rules that e contains. $\text{CFG}\left(e\right)$ is a set of CFG rules that e contains. The base probability $G_0\left(r\left|\text{root}\left(r\right)\right.\right)$ is then given by eq. 2.13 in a hierarchical way.

2.3 Markov Chain Monte Carlo Methods

For probabilistic models of grammar induction, the exact inference is often intractable. In this section, we overview approximate inference techniques called Markov Chain Monte Carlo (MCMC) methods, which are particularly suitable for many Bayesian models. Markov Chain Monte Carlo methods are algorithms for sampling from a target probability distribution based on constructing a Markov chain that has the target distribution as its equilibrium distribution.

The basic idea of sampling methods is to obtain an i.i.d. set of samples $\left\{\mathbf{x}^{(i)}\right\}_{i=1}^{N}$ from the target distribution $p\left(\mathbf{x}\right)$, where $\mathbf{x}$ is a random variable defined on a multi-dimensional space. We can approximate the expectation of $\mathbf{x}$ by a finite sum:

$$\mathbb{E}\left[x\right] \simeq \frac{1}{N} \sum_{i=1}^{N} \delta\left(x = x^i\right),$$

where δ denotes the delta-Dirac mass located at $\mathbf{x}^{(i)}$. The accuracy of the estimator depends on the number of samples N, but does not depend on the dimensionality of $\mathbf{x}$. The variance of the estimator is given in a similar way:

$$\mathbb{V}\left[x\right] \simeq \frac{1}{N} \mathbb{E}\left[\left(x - \mathbb{E}\left[x\right]\right)^2\right].$$

The set of samples can also be used to obtain a maximum of the $p\left(\mathbf{x}\right)$ as follows:

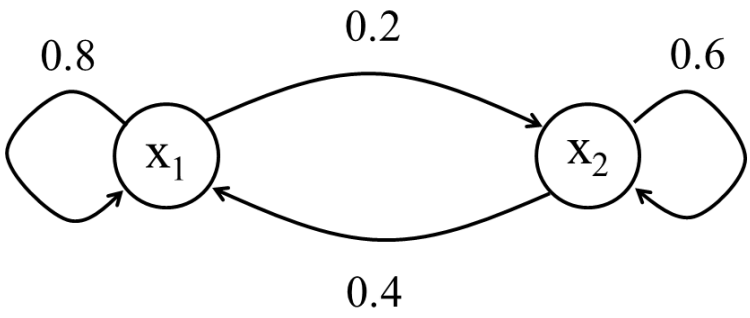


Figure 2.6: Example illustration of transition matrix for the Markov chain $\{\mathbf{x}_1, \mathbf{x}_2\}$.

$$\begin{aligned} \hat{\mathbf{x}} &= \arg\max_{\mathbf{x}} p(\mathbf{x}) \\ &\simeq \arg\max_{i=1,\ldots,N} p\left(\mathbf{x}^{(i)}\right). \end{aligned}$$

2.3.1 Markov Chain

Before introducing Markov Chain Monte Carlo methods in detail, we overview a concept of Markov chain. A stochastic process $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(i)}$ is called a (first-order) Markov chain if the following property holds.

$$\begin{aligned} p\left(\mathbf{x}^{(i)} \left| \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i-1)} \right.\right) &= p\left(\mathbf{x}^{(i)} \left| \mathbf{x}^{(i-1)} \right.\right) \\ &\equiv T\left(\mathbf{x}^{(i-1)}, \mathbf{x}^{(i)}\right), \end{aligned}$$

where $T\left(\mathbf{x}^{(i-1)}, \mathbf{x}^{(i)}\right)$ is called a *transition probability* from $\mathbf{x}^{(i-1)}$ to $\mathbf{x}^{(i)}$.

A Markov chain is called *homogeneous* if $T\left(\mathbf{x}^{(i-1)}, \mathbf{x}^{(i)}\right)$ is invariant for all i. That is, the transition probabilities are fixed as $T\left(\mathbf{x}^{(i-1)}, \mathbf{x}^{(i)}\right) = T\left(\mathbf{x}, \mathbf{x}'\right)$ through the stochastic process. For example, consider a Markov chain with two states $\{\mathbf{x}_1, \mathbf{x}_2\}$ that has the following transition matrix

$$T = \begin{bmatrix} 0.8 & 0.2 \\ 0.6 & 0.4 \end{bmatrix}.$$

The transition matrix above represents

$$\begin{aligned} p\left(\mathbf{x}^{(i)} = \mathbf{x}_1 \left| \mathbf{x}_1^{(i-1)} = \mathbf{x}_1 \right.\right) &= 0.8, \\ p\left(\mathbf{x}^{(i)} = \mathbf{x}_2 \left| \mathbf{x}^{(i-1)} = \mathbf{x}_1 \right.\right) &= 0.2, \end{aligned}$$

and so on, which is illustrated in Figure 2.6. In this example, after several iterations of the Markov process, it converges to $p(\mathbf{x}) = (0.75, 0.25)$ irrespective of the choice of initial point $\mathbf{x}^{(0)}$. It should be noted that for any state of the Markov chain, there must be a positive probability of visiting all other states. In other words, there must not be an isolated state in Figure 2.6.

A sufficient, but not neccessary condition to ensure that the distribution $p(\mathbf{x})$ is invariant is to satisfy the following property called *detailed balance*:

$$p(\mathbf{x})\, T(\mathbf{x}, \mathbf{x}') = p(\mathbf{x}')\, T(\mathbf{x}', \mathbf{x}).$$

A Markov chain that respects detailed balance is referred to as *reversible*. Summing both sides over $\mathbf{x}'$, we obtain

$$p(\mathbf{x}) = \sum_{\mathbf{x}'} T(\mathbf{x}, \mathbf{x}')\, p(\mathbf{x}) = \sum_{\mathbf{x}'} T(\mathbf{x}', \mathbf{x})\, p(\mathbf{x}').$$

If the distribution $p\left(\mathbf{x}^{(i)}\right)$ converges to the invariant distribution $p(\mathbf{x})$ for $i \to \infty$ irrespective of the choice of initial point $\mathbf{x}^{(\mathbf{0})}$, the Markov chain is said to be *ergodic.*, and then the invariant distribution is called the *equilibrium* distribution. Further discussion of ergodic Markov chains is described in [**?**].

The general strategy of Markov Chain Monte Carlo methods is to construct a Markov chain that has the target distribution as its equilibrium distribution and generate samples from the target distribution. Here we present several algorithms of Markov Chain Monte Carlo methods that are used in this work.

2.3.2 Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm [32, 18] is a widely used Markov chain Monte Carlo method for obtaining samples from a target distribution. The basic idea of Metropolis-Hastings algorithm is to generate a candidate sample $\mathbf{x}^\star$

given the current value $\mathbf{x}$ from a distribution $q(\mathbf{x}^\star|\mathbf{x})$ and then accept it with the probability $A(\mathbf{x}, \mathbf{x}^\star)$, where

$$A(\mathbf{x}, \mathbf{x}^\star) = \min\left(1, \frac{p(\mathbf{x}^\star)\, q(\mathbf{x}|\mathbf{x}^\star)}{p(\mathbf{x})\, q(\mathbf{x}^\star|\mathbf{x})}\right). \tag{2.18}$$

The distribution $q(\mathbf{x}^\star|\mathbf{x})$ is referred to as a *proposal* distribution. The pseudo-code of Metropolis-Hastings algorithm is shown in algorithm 1.

The distribution $p(\mathbf{x})$ can be shown as an invariant distribution of the Markov chain by showing the detailed balance as follows:

$$\begin{aligned} p(\mathbf{x})\, q(\mathbf{x}^\star|\mathbf{x})\, A(\mathbf{x}, \mathbf{x}^\star) &= \min\left(p(\mathbf{x})\, q(\mathbf{x}^\star|\mathbf{x}), p(\mathbf{x}^\star)\, q(\mathbf{x}|\mathbf{x}^\star)\right) \\ &= \min\left(p(\mathbf{x}^\star)\, q(\mathbf{x}|\mathbf{x}^\star), p(\mathbf{x})\, q(\mathbf{x}^\star|\mathbf{x})\right) \\ &= p(\mathbf{x}^\star)\, q(\mathbf{x}|\mathbf{x}^\star)\, A(\mathbf{x}^\star, \mathbf{x}). \end{aligned}$$

Some properties of the Metropolis-Hastings algorithm should be noted. First, the choice of proposal distribution has a great impact on the performance of the Metropolis-Hastings algorithm. If the variance of proposal distribution is small, the acceptance rate will be high, however, it results in high correlation among samples. On the other hand, if it is large, the acceptance rate will be low. Thus, careful design of proposal distribution is required for fast convergence of the Markov chain. Second, the normalizing constant of the target distribution is unnecessary since it is canceled out in both the numerator and denominator of eq. 2.18. Therefore, the Metropolis-Hastings algorithm is useful for the model whose normalizing constant is intractable.

Metropolis-Hastings algorithm has been used for statistical grammar induction. For example, the work in [24, 10] uses a sentence-level blocked Markov Chain Monte Carlo method for sampling parse trees given words in an unsupervised way. They wish to sample a derivation from the posterior distribution over the current parse tree t in a corpus given words $\mathbf{w}$ and other trees $\mathbf{t}_-$:

$$p(t|\mathbf{w}, \mathbf{t}_-; \Theta),$$

where Θ is a set of parameters. If the prior distribution of the model is based on Dirichlet process or Pitman-Yor process, direct sampling is difficult due to "rich get richer" property of the prior.

Algorithm 1: Metropolis-Hastings algorithm.

Input : number of iterations: N, target distribution: $p(\mathbf{x})$, proposal distribution: $q(\mathbf{x}|\mathbf{x}^\star)$.

Output: samples from the target distribution: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$

Initialize $\mathbf{x}^{(0)}$.
for $i = 1$ *to* N **do**
  Sample $u \sim \mathcal{U}(0, 1)$.
  Sample $\mathbf{x}^\star \sim q\left(\mathbf{x}^\star \middle| \mathbf{x}^{(i-1)}\right)$.
  if $u < A\left(\mathbf{x}^{(i-1)}, \mathbf{x}^\star\right)$ **then**
    $\mathbf{x}^{(i)} = \mathbf{x}^\star$
  else
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)}$
  end
end

Specifically, their method consists of the following three steps. For each sentence,

1. Run the Inside-Outside algorithm for given words and grammatical rules. The parameters are fixed during the algorithm.

2. Generate a sample of derivation by expanding each nonterminal in a top-down fashion.

3. Accept or reject the sample by Metropolis-Hastings test.

The proposal distribution is simply set as follows:

$$q(t^\star|\mathbf{t}_-) = p\left(t^\star \middle| \mathbf{t}_-, \Theta^{\mathrm{MAP}}\right),$$

where Θ^{MAP} is the maximum a posteriori (MAP) estimate of the parameters, which is computed from $\mathbf{t}_-$. The proposal probability differs from the true probability only when multiple grammarical rules are used in the derivation $t^\star$.

2.3.3 Gibbs Sampling

Gibbs sampling [17] is a widely applicable Markov chain Monte Carlo algorithm for sampling from a target distribution. Unlike with Metropolis-Hastings algorithm, a user-defined proposal distribution is unnecessary for Gibbs sampling.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_M)$ be a multi-dimensional vector and $p(\mathbf{x})$ be a target distribution we wish to sample. The Gibbs sampler picks one variable at a time and generates a sample from the conditional probability: $p(x_m | \mathbf{x}_{-m})$, where $\mathbf{x}_{-m}$ represents $\mathbf{x}$ excluding the mth component x_m. The procedure of Gibbs sampling always accepts the generated sample and update the value of x_m. Indeed, as described later, the Gibbs sampling can be viewed as a special case of the Metropolis- Hastings algorithm of acceptance rate being equal to one. The procedure is repeated through the variables either in some particular order or at random. Figure 2 shows the pseudo-code of Gibbs sampling.

The Gibbs sampling can be viewed as a special case of Metropolis-Hastings algorithm whose proposal distribution is given by $q(\mathbf{x}^\star | \mathbf{x}) = p(x_m^\star | \mathbf{x}_{-m})$. Besides, $\mathbf{x}_{-m}^\star = \mathbf{x}_{-m}$ holds since $\mathbf{x}_{-m}$ remains fixed during the sampling of x_m. Overall, eq. 2.18 is computed as follows:

$$\begin{aligned} A(\mathbf{x}, \mathbf{x}^\star) &= \min\left(1, \frac{p(\mathbf{x}^\star)\, q(\mathbf{x} | \mathbf{x}^\star)}{p(\mathbf{x})\, q(\mathbf{x}^\star | \mathbf{x})}\right) \\ &= \min\left(1, \frac{p(x_m^\star | \mathbf{x}_{-m}^\star)\, p(\mathbf{x}_{-m}^\star)\, p(x_m | \mathbf{x}_{-m}^\star)}{p(x_m | \mathbf{x}_{-m})\, p(\mathbf{x}_{-m})\, p(x_m^\star | \mathbf{x}_{-m})}\right) \\ &= \min(1, 1) \\ &= 1. \end{aligned}$$

Thus the Metropolis-Hastings steps are always accepted in the Gibbs sampling.

Next, we show that the distribution $p(\mathbf{x})$ is invariant through the iterations of Gibbs sampling. Since $p(\mathbf{x}_{-m})$ remains invariant during the procedure, the conditional distribution of one variable given all others is proportional to the joint distribution:

$$p(x_m | \mathbf{x}_{-m}) = \frac{p(\mathbf{x})}{p(\mathbf{x}_{-m})} \propto p(\mathbf{x}).$$

Therefore, the distribution $p(\mathbf{x})$ is invariant through the iterations of Gibbs sampling.

Algorithm 2: Gibbs sampling.

Input : number of iterations: N, target distribution: $p(\mathbf{x})$.
Output: samples from the target distribution: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$

Initialize $\mathbf{x}^{(0)}$.
for $i = 1$ *to* N **do**
  Sample $x_1^{(i)} \sim p\left(x_1 \middle| x_2^{(i-1)}, x_3^{(i-1)}, \ldots, x_M^{(i-1)}\right)$.
  Sample $x_2^{(i)} \sim p\left(x_2 \middle| x_1^{(i)}, x_3^{(i-1)}, \ldots, x_M^{(i-1)}\right)$.
  $\vdots$
  Sample $x_M^{(i)} \sim p\left(x_M \middle| x_1^{(i)}, x_2^{(i)}, \ldots, x_{M-1}^{(i)}\right)$.
end

A blocked Gibbs sampling improves search efficiency by grouping two or more variables together and sampling from their joint distribution conditioned on all other variables, rather than sampling one variable at a time. To calculate candidate probabilities of multiple variables efficiently, it is often combined with dynamic programming techniques such as forward-backward algorithm for sequential data and inside-outside algorithm for tree-structured data.

2.3.4 Slice Sampling

Slice sampling [33] is a Markov chain Monte Carlo method that utilizes an auxiliary variable. It has advantages over the Metropolis-Hastings algorithm in that the proposal distribution is not required since the step size of the sampling is automatically adapted to match the characteristics of the target distribution. Besides, the normalized constant of target distribution is also unnecessary. Slice sampling is widely used in grammar induction for the inference of hyperparameters of grammatical models.

The key observation of slice sampling is that one can sample from a distribution by sampling uniformly from the region under the graph of its distribution. A Markov chain that converges to this uniform distribution can be constructed by sampling a variable in the vertical direction and horizontal direction alternately.

Consider a simple single-variable case as an example. Let $p(x)$ be one-dimensional probability distribution and $f(x)$ be the unnormalized distribution of $p(x)$. Our

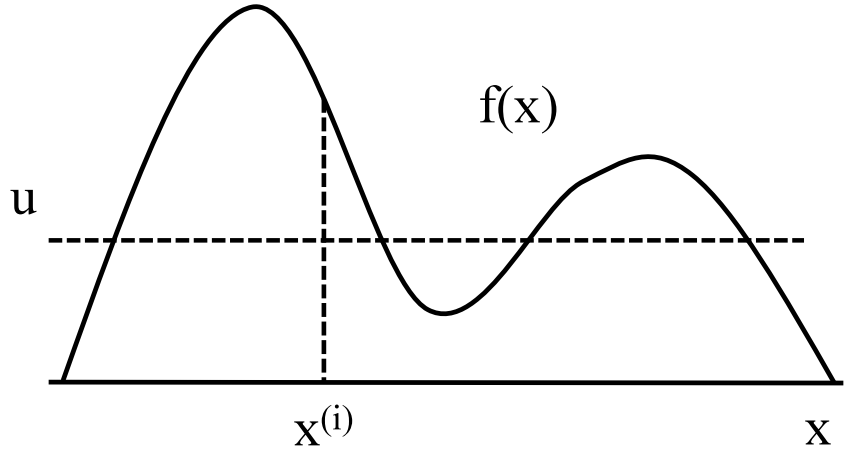


Figure 2.7: Illustration of slice sampling.

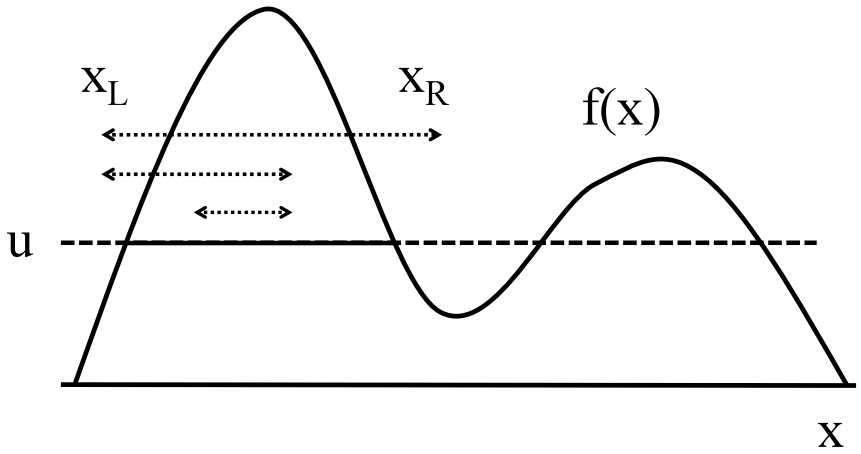


Figure 2.8: Illustration of the "doubling" procedure.

goal is to sample from a target distribution $p(x)$. The basic idea of slice sampling is as follows. First, sample an auxiliary variable u uniformly at random. Second, sample x uniformly at random from $p(x,u)$, a joint distribution over x and u, which is defined as follows:

$$U = \{(x,u) \,|0 < u < f(x)\}. \tag{2.19}$$

That is, the joint distribution over (x,u) is given by

$$p(x,u) = \begin{cases} \frac{1}{Z_p} & \text{if } 0 < u < f(x) \\ 0 & \text{otherwise,} \end{cases} \tag{2.20}$$

where $Z_p = \int f(x)\,dx$. The marginal distribution over x is then given by

$$p(x) = \int p(x,u)\,du = \int_0^{f(x)} \frac{1}{Z_p} du = \frac{f(x)}{Z_p}. \tag{2.21}$$

Therefore, we can sample from the target distribution $p(x)$ by alternating sampling u in the vertical direction and sampling x in the horizontal direction from the following region S:

$$S = \{x \,|u < f(x)\}. \tag{2.22}$$

Figure 2.7 shows an illustration of slice sampling and the algorithm 3 shows the pseudo-code of slice sampling.

In practice, it is often difficult to sample directly from the slice S. Instead, we can sample from the alternative region: $\{x \,|x_L \leq x, x \leq x_R\}$ as long as it is ensured that the sampling from the range leaves the uniform distribution over S invariant. We wish to set x_L and x_R so as to encompass as much of the slice S as possible for large moves, while having as little as possible of this region lying outside the slice.

The work in [33] proposes several procedures for finding the range (x_L, x_R). One approach is called a "doubling" procedure, which is illustrated in Figure 2.8. The doubling procedure first initialize the range that contains x, then expand the interval twice the size of the previous one until both x_L and x_R are outside of the range, or a predetermined limit is reached. The expanded side (left or right) are chosen at random for each iteration. The correctness of this algorithm is described in [33]. Slice sampling can be applied to multivariate distributions

Algorithm 3: Single-variable slice sampling.

Input : number of iterations: N, unnormalized target distribution: $f(x)$.
Output: samples from the target distribution: $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$

Initialize $x^{(0)}$.
for $i = 1$ *to* N **do**
    Sample $u \sim \text{Uniform}\left(0, f\left(x^{(i-1)}\right)\right)$.
    Find the range (x_L, x_R) that contains $x^{(i-1)}$ and much of the slice $S = \{x \,|u < f(x)\}$.
    Sample $x^{(i)} \sim \text{Uniform}(x_L, x_R)$.
end

by repeatedly sampling each variable $\mathbf{x} = (x_1, x_2, \ldots, x_M)$ in turn, in the same way of Gibbs sampling.

Chapter 3

Statistical Induction of Tree Insertion Grammars

3.1 Introduction

Many statistical models for syntactic parsing are based on Context Free Grammars (CFG). Although learning a probabilistic CFGs directly from a treebank data via maximum likelihood estimation results in poor performance, it has been shown that dividing coarse treebank symbols (e.g. NP and VP) into subcategories such as parent annotation [22], head lexicalization [5, 12], and automatic refinement [31, 35] significantly improve parsing accuracy. However, CFG is not powerful enough to capture natural language features such as predicate-argument structure and argument-adjunct distinction, thus more expressive grammars are desirable for deep understanding of natural language.

Tree Insertion Grammar (TIG) [39] is an alternative grammar formalism that is similar to well-known Tree Adjoining Grammar (TAG) [25]. Some variants of TIG and TAG have been already applied to syntactic parsing [8], semantic role labeling [29], and machine translation [4, 13]. TIG differs from CFG in that TIG allows nonterminal symbols to be replaced or inserted by tree fragments of arbitrary size, while CFG allows nonterminal symbols to be replaced by minimal tree fragments. Since an insertion operator is helpful to distinguish arguments from adjuncts, we can expect the TIG to capture natural language phenomena.

One of the problems of obtaining TIGs from parse trees is that standard treebank data such as Penn Treebank [30] is not annotated with TIG *derivations*, i.e., how tree fragments are combined to form parse trees. As different deriva-

tions might produce the same parse tree, it is necessary to extract TIGs from parse trees in an unsupervised fashion. Previous work on TIG and TAG induction [47, 8, 7] has relied on heuristic rules and maximum likelihood estimators to extract grammatical rules. For example, the work described in [8] extracts grammatical rules in a heuristic way by using the information of heads, arguments, and adjuncts for each nonterminal node. As noted in [38], such language-specific heuristics and maximum likelihood estimators lead to an overfit of the training data.

Instead of using such *ad hoc* approaches, we present a statistical method that automatically induces TIG derivations from parse trees. Statistical approach has an advantage that it is independent of language and annotation style of dataset. Our probabilistic TIG model is based on a nonparametric Bayesian prior to handle an arbitrarily large number of subtrees and keep the grammar size as compact as possible. We also develop an efficient inference technique based on the Markov Chain Monte Carlo (MCMC) method for our TIG model.

Recent work on statistical grammar induction has extended CFG to Tree Substitution Grammars (TSG), a restricted variant of TIG without a tree insertion operator [**?**]. We expect that the tree insertion operator of TIG will help us model the data more accurately, and make the grammar more compact than TSG.

We applied the proposed method to the English Penn Treebank data and obtained better parsing results than TSG for a small dataset and comparable results for a large dataset, making the grammar rules much smaller than TSG as we expected. We also confirmed that our model automatically captured adjuncts as an insertion operator and other elements as a substitution operator without any linguistic knowledge.

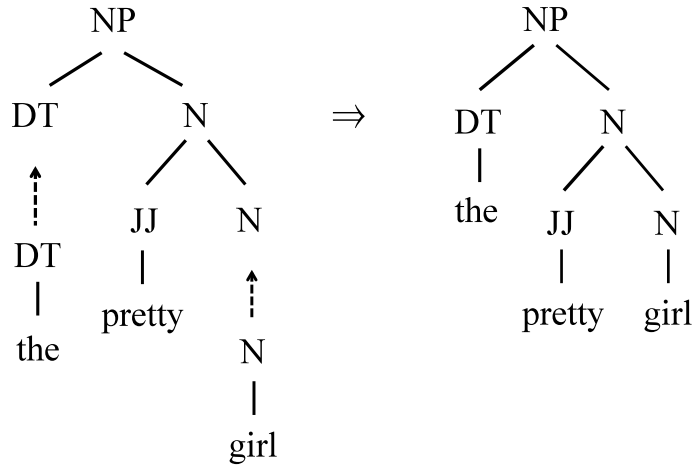
3.2 Tree Insertion Grammars

Formally, TIG is defined by a 5-tuple $\mathcal{G} = (V_N, V_T, S, I, A)$ where V_N is a finite set of nonterminal symbols, V_T is a finite set of terminal symbols, $S \in V_N$ is the start symbol, I is a finite set of *initial trees* and A is a finite set of *auxiliary trees*. The trees in the set $I \cup A$ are referred to as *elementary trees*.

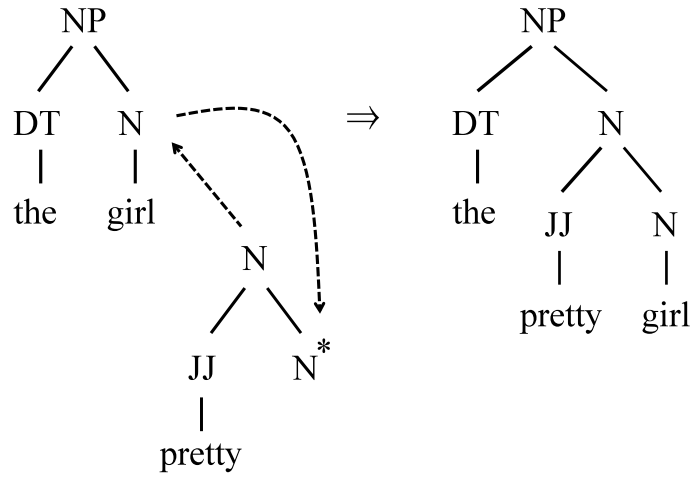
Figure 3.1 shows examples of initial and auxiliary trees. Initial trees consist of nonterminal and terminal nodes. The nonterminal leaf node in initial trees is referred to as a *frontier* node. Auxiliary trees also consist of nonterminal and

(a) (b)

Figure 3.1: Examples of elementary trees. (a) Intial tree. (b) Auxiliary tree.



(a)



(b)

Figure 3.2: Example of (a) substitution and (b) insertion (dotted line).

terminal nodes, however, they contain a special nonterminal leaf node labeled with the same symbol as the root node. This leaf node is referred to as a *foot* node (marked with the subscript " * "). We refer to nodes that are neither leaf nor root nonterminal nodes in elementary trees as *internal* nodes.

A derivation starts with an initial tree whose root node is labeled with the start symbol, combining elementary trees with two types of operations: *substitution* and *insertion*. Figure 3.2a shows an example of substitution. Substitution replaces a frontier node with another initial tree whose root node is labeled with the same symbol as the frontier node.

Figure 3.2b shows an example of insertion. Auxiliary trees can be inserted into another tree at an internal node when the root node of the auxiliary tree and the internal node are the same symbol. The children of the internal node are combined with the foot node.

To avoid expensive computational cost, we constrain auxiliary trees to be *simple auxiliary trees*, i.e., auxiliary trees whose root node must generate a foot node as an immediate child. For example, " (N (JJ pretty) N*) " in Figure 3.2b is a simple auxiliary tree, while " (S (NP) (VP (V think) S*)) " is not. We place no restriction on the initial trees.

Let us refer to some differences between TAG, the original TIG [39] and our TIG variant. TIG is a strict subset of TAG since it does not allow wrapping adjunction in TAG. TIG generates context-free languages and the parsing complexity is $O(n^3)$ where n is a sentence length. Our TIG variant prohibits neither wrapping adjunction in TAG nor simultaneous adjunction in the original TIG, and allows only simple auxiliary trees. The expressive power and computational complexity of our formalism is identical to the original TIG, however, simple auxiliary trees do not increase the computational cost of the grammar induction since the probability distribution over simple auxiliary trees can be defined as having the same form as that of initial trees. This ensures that we can make use of an efficient dynamic programming for training our model, which we describe the detail later.

3.3 Probabilistic Tree Insertion Grammars

In this section, we propose a probabilistic model of TIG. Recall that our task is the induction of TIG derivations from parse trees in treebank data. As we

mentioned in the introduction, this is nontrivial since different derivations might produce the same parse tree. For example, Figure 3.2a and 3.2b produce the same parse tree. For this reason, we employ a probabilistic model of elementary trees given a parse tree, $p(\mathbf{e}\,|t)$, where $\mathbf{e}$ is a set of elementary trees (i.e., initial tree and auxiliary tree) and t is a parse tree. $p(\mathbf{e}\,|t)$ can be computed using Bayes' rule:

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

where the value of $P(t\,|\mathbf{e})$ is either 1 (if t is consistent with $\mathbf{e}$) or 0 (otherwise). Therefore, our task turns out to be to model the prior distribution $P(\mathbf{e})$.

3.3.1 Substitution Model

We follow the Bayesian TSG models [11, 38] for probabilistic modeling of initial trees. Let G_X be the probability distribution over initial trees whose root nonterminal symbol is X. We employ a Pitman-Yor process (PYP) [37] as a prior on G_X:

$$\begin{aligned} e\,|X &\sim G_X \\ G_X\,|d_X, \theta_X &\sim \mathrm{PYP}(d_X, \theta_X, P_0(\cdot\,|X)) \end{aligned}$$

where $P_0(\cdot\,|X)$ is a *base distribution* over the infinite space of initial trees rooted with X. d_X and θ_X are hyperparameters to control the model's behavior. Integrating out all possible values of G_X, the resulting distribution is

$$\begin{aligned} p(e_i\,|\mathbf{e}_{-i}, X, d_X, \theta_X) &= \alpha_{e_i,X} + \beta_X P_0(e_i\,|X) \qquad (3.1) \\ \alpha_{e_i,X} &= \frac{n^{-i}_{e_i,X} - d_X \cdot t_{e_i,X}}{\theta_X + n^{-i}_{\cdot,X}} \\ \beta_X &= \frac{\theta_X + d_X \cdot t_{\cdot,X}}{\theta_X + n^{-i}_{\cdot,X}} \end{aligned}$$

where $\mathbf{e}_{-i} = e_1, \ldots, e_{i-1}$ are previously generated initial trees, and $n^{-i}_{e_i,X}$ is the number of times e_i has been used in $\mathbf{e}_{-i}$. $t_{e_i,X}$ is the number of *tables* labeled

with e_i, which we describe below. $n^{-i}_{\cdot,X} = \sum_e n^{-i}_{e,X}$ and $t_{\cdot,X} = \sum_e t_{e,X}$ are the total counts of initial trees and tables, respectively.

The base probability of an initial tree, $P_0(e\,|X)$, is given as follows.

$$\begin{aligned} P_0(e\,|X) &= \prod_{r \in \mathrm{CFG}(e)} P_{\mathrm{MLE}}(r) \times \prod_{A \in \mathrm{LEAF}(e)} s_A \\ &\times \prod_{B \in \mathrm{INTER}(e)} (1 - s_B), \end{aligned} \tag{3.2}$$

where $\mathrm{CFG}(e)$ is a set of decomposed CFG productions of e, $P_{\mathrm{MLE}}(r)$ is a maximum likelihood estimate (MLE) of r. $\mathrm{LEAF}(e)$ and $\mathrm{INTER}(e)$ are sets of leaf and internal symbols of e, respectively. s_X is a *stopping probability* defined for each X.

The process for generating an initial tree from the base distribution is as follows:

1. Generate a CFG rule r given X with probability $P_{\mathrm{MLE}}(X \to r)$.
2. For each expanded nonterminal Y in r, decide whether to generate child nodes with probability $(1 - s_Y)$, or stop with probability s_Y. Go back to 1 if Y generates child nodes.

Here $P_{\mathrm{MLE}}(X \to r)$ is a MLE of r given X, which is obtained from parse trees in the training data. s_X is a stopping probability defined for each nonterminal X.

Eq. 3.1 is known as the "Chinese restaurant" metaphor. Consider a Chinese restaurant with unbounded number of tables. Customers (initial trees in our case) visit the restaurant, and sit at a table one by one. The first customer always sits at the first table, and each subsequent customer sits at an already occupied table with probability $\alpha_{e_i,X}$, or a new table with probability β_X. When a customer sits at a new table, an initial tree is drawn from P_0, and the new table is labeled with the initial tree. The PYP produces *rich get richer* statistics: a few initial trees are used many times for derivation while many are used only a few times, which is empirically shown to be well-suited for natural language [44, 23].

3.3.2 Insertion Model

We model a probability distribution over simple auxiliary trees as having the same form as eq. 3.1, that is,

$$p\left(e'_i \,|\mathbf{e}'_{-i}, X, d'_X, \theta'_X\right) = \alpha'_{e_i,X} + \beta'_X P'_0\left(e'_i, |X\right) \tag{3.3}$$

where e'_i is an auxiliary tree, d'_X and θ'_X are hyperparameters of the insertion model, and the definition of $\left(\alpha'_{e_i,X}, \beta'_X\right)$ is the same as that of $(\alpha_{e_i,X}, \beta_X)$ in eq. 3.1.

We need to modify the base distribution over simple auxiliary trees, $P'_0\left(e\,|X\right)$, as follows, so that all probabilities of the simple auxiliary trees sum to one.

$$\begin{aligned} P'_0\left(e'\,|X\right) &= P'_{\mathrm{MLE}}\left(\mathrm{TOP}\left(e'\right)\right) \times \prod_{r \in \mathrm{INTER\_CFG}(e')} P_{\mathrm{MLE}}\left(r\right) \\ &\times \prod_{A \in \mathrm{LEAF}(e')} s_A \times \prod_{B \in \mathrm{INTER}(e')} \left(1 - s_B\right) \end{aligned} \tag{3.4}$$

where $\mathrm{TOP}\left(e'\right)$ is the CFG production that starts with the root node of e'. For example, $\mathrm{TOP}\left(\mathrm{N\ (JJ\ pretty)\ (N*)}\right)$ returns " $\mathrm{N} \rightarrow \mathrm{JJ\ N*}$ ". $\mathrm{INTER\_CFG}\left(e'\right)$ is a set of CFG productions of e' excluding $\mathrm{TOP}\left(e'\right)$. $P'_{\mathrm{MLE}}\left(r'\right)$ is a modified MLE for simple auxiliary trees, which is given by

$$\begin{cases} \frac{count(r')}{count(X \rightarrow X^* Y) + count(X \rightarrow Y X^*)} & \text{if } r' \text{ includes a foot node} \\ 0 & \text{otherwise} \end{cases}$$

where $count\left(r'\right)$ is the number of times r' occurs in parse trees. It is ensured that $P'_0\left(e'\,|X\right)$ generates a foot node as an immediate child.

In short, we define the probability distribution over both initial trees and simple auxiliary trees with a PYP prior. The base distribution over initial trees is defined as $P_0\left(e\,|X\right)$, and the base distribution over simple auxiliary trees is defined as $P'_0\left(e'\,|X\right)$. An initial tree e_i replaces a frontier node with probability $p\left(e_i\,|\mathbf{e}_{-i}, X, d_X, \theta_X\right)$. On the other hand, a simple auxiliary tree e'_i inserts an internal node with probability $a_X \times p'\left(e'_i \,|\mathbf{e}'_{-i}, X, d'_X, \theta'_X\right)$, where a_X is an insertion probability defined for each X. The stopping probabilities are common to both initial and auxiliary trees.

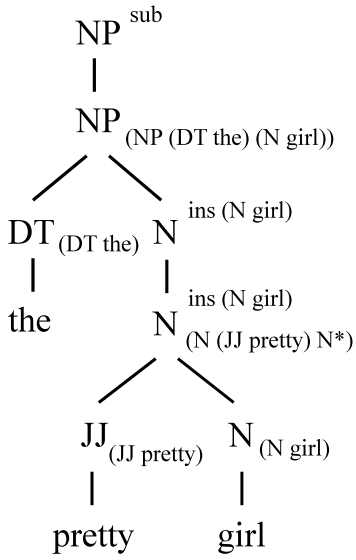


Figure 3.3: Derivation of Fig. 3.2b transformed by grammar decomposition.

3.4 Inference

3.4.1 Grammar Decomposition

To sample derivations from eqs. 3.1 and 3.3, we develop a *grammar decomposition* technique, which is an extension of work [10] on the Bayesian TSG model. The motivation behind our grammar decomposition is that it is difficult to consider all possible elementary trees explicitly since our base distribution assigns non-zero probability to an infinite number of unseen elementary trees. Alternatively, we transform TIG into the equivalent CFG to sample derivations efficiently. The key idea of grammar decomposition is:

1. Differentiate the first term of eqs. 3.1 and 3.3 from the second term. The first term corresponds to the sample from already generated trees, and the second term corresponds to the sample from the base distribution.

2. Embed all of derivational information in nonterminal symbols of a parse tree. We define six types of nonterminal symbols as follows.

Nonterminal symbols for initial trees

	rule	probability
$\mathrm{NP}^{\mathrm{sub}}$	$\rightarrow \mathrm{NP}_{(\mathrm{NP\ (DT\ the)\ (N\ girl)})}$	$\alpha_{e=(\mathrm{NP\ (DT\ the)\ (N\ girl)}),\mathrm{NP}}$
$\mathrm{NP}_{(\mathrm{NP\ (DT\ the)\ (N\ girl)})}$	$\rightarrow \mathrm{DT}_{(\mathrm{DT\ the})}\mathrm{N}^{\mathrm{ins\ (N\ girl)}}$	$\left(1 - a_{(X=\mathrm{DT})}\right) \times a_{(X=\mathrm{N})}$
$\mathrm{DT}_{(\mathrm{DT\ the})}$	$\rightarrow$ the	1
$\mathrm{N}^{\mathrm{ins\ (N\ girl)}}$	$\rightarrow \mathrm{N}^{\mathrm{ins\ (N\ girl)}}_{(\mathrm{N\ (JJ\ pretty)\ N*})}$	$\alpha'_{e=(\mathrm{N\ (JJ\ pretty)\ N*}),\mathrm{N}}$
$\mathrm{N}^{\mathrm{ins\ (N\ girl)}}_{(\mathrm{N\ (JJ\ pretty)\ N*})}$	$\rightarrow \mathrm{JJ}_{(\mathrm{JJ\ pretty})}\mathrm{N}_{(\mathrm{N\ girl})}$	$\left(1 - a_{(X=\mathrm{JJ})}\right) \times 1$
$\mathrm{JJ}_{(\mathrm{JJ\ pretty})}$	$\rightarrow$ pretty	1
$\mathrm{N}_{(\mathrm{N\ girl})}$	$\rightarrow$ girl	1

Table 3.1: The rules and probabilities of grammar decomposition for Figure 3.3.

- $X_{(\mathrm{base})}$ is a root or internal symbol of initial trees sampled from the base distribution P_0.

- $X_{(e)}$ is a root or internal symbol of initial trees sampled from already generated trees e (the first term of eq. 3.1). $X_{(e)}$ must generate child nodes according to e. For example, let X be NP and e be "NP (DT) (N girl)" (Figure 3.1a). Then, $\mathrm{NP}_{(\mathrm{NP\ (DT\ )\ (N\ girl)})}$ generates two children: $\mathrm{DT}^{\mathrm{sub}}$ and $\mathrm{N}_{(\mathrm{N\ girl})}$. $\mathrm{N}_{(\mathrm{N\ girl})}$ generates the terminal node " girl ".

- X^{sub} is a frontier node at which substitution occurs. The superscript " sub " indicates substitution. X^{sub} chooses the sample of initial trees from either base distribution (second term of eq. 3.1) or already generated trees (first term of eq. 3.1). Therefore, we set X^{sub} to generate $X_{(\mathrm{base})}$ (with probability β_X) or $X_{(e)}$ (with probability $\alpha_{e,X}$) as a child.

Nonterminal symbols for auxiliary trees

- $X^{\mathrm{ins}(type)}$ is a node at which insertion occurs at internal node $X_{(type)}$, where *type* is either " base " or " e ". The superscript " ins " indicates insertion. Similar to X^{sub}, $X^{\mathrm{ins}(type)}$ chooses the sample of auxiliary trees from either the base distribution or already generated trees, i.e., $X^{\mathrm{ins}(type)}$ generates $X^{\mathrm{ins}(type)}_{\mathrm{base}}$ (with probability β'_X) or $X^{\mathrm{ins}(type)}_{(e)}$ (with probability $\alpha'_{e,X}$), where $X^{\mathrm{ins}(type)}_{\mathrm{base}}$ is a root symbol of auxiliary trees sampled from the base

distribution, and $X_{(e)}^{\mathrm{ins}(type)}$ is a root symbol of auxiliary trees sampled from already generated auxiliary trees e. $X_{\mathrm{base}}^{\mathrm{ins}(type)}$ and $X_{(e)}^{\mathrm{ins}(type)}$ must generate $X_{(type)}$ as the foot node symbol.

Figure 3.3 shows an example derivation employing the grammar decomposition. As shown in Figure 3.3, all of the derivational information is embedded in refined nonterminal symbols. It should be noted that determining each nonterminal symbol X in a parse tree as any of the above six types specifies a unique TIG derivation. Thus, instead of dealing with an insertion operator explicitly, we embed all of derivational information in nonterminal symbols and deal with the refined parse tree as standard CFG.

The rules and probabilities of grammar decomposition for Figure 3.3 are shown in Table 3.1. The generative process of our TIG model begins with the frontier start symbol $\mathrm{S}^{\mathrm{sub}}$, and generates child nodes acording to the grammar decomposition rules. $\mathrm{S}^{\mathrm{sub}}$ generates either $\mathrm{S}_{(\mathrm{base})}$ (with probability β_{S}) or $\mathrm{S}_{(e)}$ (with probability $\alpha_{e,\mathrm{S}}$). Then, $\mathrm{S}_{(\mathrm{base})}$ generates one of three nodes as a child: a frontier node, an internal node, or a root node of some auxiliary tree because of adjunction. That is, $\mathrm{S}_{(\mathrm{base})}$ is expanded to any of three symbols: X^{sub} (with probability $P_{\mathrm{MLE}}(\mathrm{S} \to X) \times s_X$), $X_{(\mathrm{base})}$ (with probability $P_{\mathrm{MLE}}(\mathrm{S} \to X) \times (1 - s_X) \times (1 - a_X)$), or $X^{\mathrm{ins}(\mathrm{base})}$ (with probability $P_{\mathrm{MLE}}(\mathrm{S} \to X) \times (1 - s_X) \times a_X$).

On the other hand, $\mathrm{S}_{(e)}$ generates either the child of e (when insertion does not occur) or a root symbol of some auxiliary tree (when insertion occurs). That is, $\mathrm{S}_{(e)}$ is expanded to $X_{(\mathrm{child}(e))}$ (with probability $(1 - a_X)$), or $X^{\mathrm{ins}(\mathrm{child}(e))}$ (with probability a_X), where $\mathrm{child}(e)$ is the direct child of e. The sampling stops when all leaf nodes become terminals. We tabularize the rules and probabilities of our grammar decomposition in Tables 3.2 and 3.3..

3.4.2 Training and Parsing

We use a sentence-level Metropolis-Hastings (MH) algorithm [10] to train our model. The MH algorithm is one of the Markov chain Monte Carlo (MCMC) methods for obtaining random samples from a probability distribution. In our case, we wish to obtain derivation samples from $p(\mathbf{e} \,|\mathbf{t}, \mathbf{d}, \boldsymbol{\theta}, \mathbf{s}, \mathbf{a})$.

The MH algorithm consists of the following three steps. For each sentence,

rule			probability
X^{sub}	$\rightarrow$	$X_{(\text{base})}$	β_X
X^{sub}	$\rightarrow$	$X_{(e)}$	$\alpha_{e,X}$
$X^{\text{adj}(type)}$	$\rightarrow$	$X^{\text{adj}(type)}_{(\text{base})}$	β'_X
$X^{\text{adj}(type)}$	$\rightarrow$	$X^{\text{adj}(type)}_{(e)}$	$\alpha'_{e,X}$

(a)

rule			probability		
$X_{(e)} \rightarrow$	w		1	(if $w = \text{child}\,(e)$)	
$X_{(e)} \rightarrow$	$Y_{(\text{child}(e))}$		$(1 - a_Y)$		
$X_{(e)} \rightarrow$	$Y^{\text{adj}(\text{child}(e))}$		a_Y		
$X_{(e)} \rightarrow$	$Y_{(\text{childL}(e))}$	$Z_{(\text{childR}(e))}$	$(1 - a_Y)$	$\times$	$(1 - a_Z)$
$X_{(e)} \rightarrow$	$Y_{(\text{childL}(e))}$	$Z^{\text{adj}(\text{childR}(e))}$	$(1 - a_Y)$	$\times$	a_Z
$X_{(e)} \rightarrow$	$Y^{\text{adj}(\text{childL}(e))}$	$Z_{(\text{childR}(e))}$	a_Y	$\times$	$(1 - a_Z)$
$X_{(e)} \rightarrow$	$Y^{\text{adj}(\text{childL}(e))}$	$Z^{\text{adj}(\text{childR}(e))}$	a_Y	$\times$	a_Z

(b)

Table 3.2: Grammar decomposition for (a) X^{sub} and $X^{\text{ins}(type)}$. X^{sub} and $X^{\text{ins}(type)}$always produce a unary child labeled with the same symbol. (b) $X_{(e)}$. childL (e) and childR (e) are the left and right child of e, respectively. w is a terminal symbol.

rule			probability			
$X_{(\text{base})} \rightarrow$	w		$P_{\text{MLE}}(X \rightarrow w)$			
$X_{(\text{base})} \rightarrow$	Y^{sub}		$P_{\text{MLE}}(X \rightarrow Y) \times$	s_Y		
$X_{(\text{base})} \rightarrow$	$Y_{(\text{base})}$		$P_{\text{MLE}}(X \rightarrow Y) \times$	$(1 - s_Y)(1 - a_Y)$		
$X_{(\text{base})} \rightarrow$	$Y^{\text{adj}(type)}$		$P_{\text{MLE}}(X \rightarrow Y) \times$	$(1 - s_Y)\, a_Y$		
$X_{(\text{base})} \rightarrow$	Y^{sub}	Z^{sub}	$P_{\text{MLE}}(X \rightarrow YZ) \times$	s_Y	$\times$	s_Z
$X_{(\text{base})} \rightarrow$	Y^{sub}	$Z_{(\text{base})}$	$P_{\text{MLE}}(X \rightarrow YZ) \times$	s_Y	$\times$	$(1 - s_Z)(1 - a_Z)$
$X_{(\text{base})} \rightarrow$	Y^{sub}	$Z^{\text{adj}(type)}$	$P_{\text{MLE}}(X \rightarrow YZ) \times$	s_Y	$\times$	$(1 - s_Z)\, a_Z$
$X_{(\text{base})} \rightarrow$	$Y_{(\text{base})}$	Z^{sub}	$P_{\text{MLE}}(X \rightarrow YZ) \times$	$(1 - s_Y)(1 - a_Y)$	$\times$	s_Z
$X_{(\text{base})} \rightarrow$	$Y_{(\text{base})}$	$Z_{(\text{base})}$	$P_{\text{MLE}}(X \rightarrow YZ) \times$	$(1 - s_Y)(1 - a_Y)$	$\times$	$(1 - s_Z)(1 - a_Z)$
$X_{(\text{base})} \rightarrow$	$Y_{(\text{base})}$	$Z^{\text{adj}(type)}$	$P_{\text{MLE}}(X \rightarrow YZ) \times$	$(1 - s_Y)(1 - a_Y)$	$\times$	$(1 - s_Z)\, a_Z$
$X_{(\text{base})} \rightarrow$	$Y^{\text{adj}(type)}$	Z^{sub}	$P_{\text{MLE}}(X \rightarrow YZ) \times$	$(1 - s_Y)\, a_Y$	$\times$	s_Z
$X_{(\text{base})} \rightarrow$	$Y^{\text{adj}(type)}$	$Z_{(\text{base})}$	$P_{\text{MLE}}(X \rightarrow YZ) \times$	$(1 - s_Y)\, a_Y$	$\times$	$(1 - s_Z)(1 - a_Z)$
$X_{(\text{base})} \rightarrow$	$Y^{\text{adj}(type)}$	$Z^{\text{adj}(type)}$	$P_{\text{MLE}}(X \rightarrow YZ) \times$	$(1 - s_Y)\, a_Y$	$\times$	$(1 - s_Z)\, a_Z$

Table 3.3: Grammar decomposition for $X_{(\text{base})}$. The rules and probabilities for $X^{\text{ins}(type)}_{(\text{base})}$ and $X^{\text{ins}(type)}_{(e)}$ are defined in the same manner.

1. Calculate the inside probability [27] in a bottom-up manner using the grammar decomposition.

2. Sample a derivation tree in a top-down manner

3. Accept or reject the derivation sample by using the MH test

The MH algorithm is described in detail in [10]. In a training step, the inside algorithm considers only the derivations that match the parsed tree.

In a parsing step, we apply the inside algorithm in a similar manner to that in training to obtain derivation samples. The only difference between training and parsing is that the inside algorithm considers any possible derivations in a parsing step since parse trees are unobservable. After sampling derivations, we use the MAX-RULE-SUM algorithm [35] to obtain the final parsing results.

We deal with four kinds of hyperparameters, $(\mathbf{d}, \boldsymbol{\theta}, \mathbf{s}, \mathbf{a})$, as random variables and update their values for every MH iteration. We place a prior on hyperparameters as follows.

- d_X and $d'_X \sim \text{Beta}\,(1, 1)$
- θ_X and $\theta'_X \sim \text{Gamma}\,(0.1, 10.0)$
- $s_X \sim \text{Beta}\,(1, 1)$
- $a_X \sim \text{Beta}\,(b_1, b_2)$

We adopt various values of (b_1, b_2) in the experiments to investigate the effect of insertion. The hyperparameters of $\mathbf{d}$ and $\boldsymbol{\theta}$ are updated with the auxiliary variable technique [43].

3.5 Experiment

We ran experiments on the Wall Street Journal (WSJ) portion of the English Penn Treebank and the British National Corpus (BNC) Treebank[1]. The WSJ dataset consists of 24 sections and contains approximately 40k sentences. Section 22 is usually assigned as a development set for parameter tuning, but we do not

[1] http://nclt.computing.dcu.ie/~jfoster/resources/

corpus	method	F1
BNC	CFG	54.08
	TSG	67.73
	TIG (proposed)	**69.06**
WSJ	CFG	64.99
	TSG	77.19
	TIG (proposed)	**78.54**
	[35]	77.93[3]
	[10]	78.40

Table 3.4: Small dataset experiments.

use a development set since our model automatically updates the hyperparameters for every iterations.

The treebank data is binarized with " CENTER-HEAD " method [31] to make grammars with only unary and binary productions. We replace lexical words with count ≤ 1 in the training set with one of 3 unknown words using lexical features.

We trained our model using a training set, and then sampled 10k derivations for each sentence in a test set. Parsing results were obtained with the MAX-RULE-SUM algorithm using the 10k derivation samples. We show the bracketing F1 score of predicted parse trees evaluated by EVALB[2], averaged over three independent runs.

3.5.1 Small dataset

In small dataset experiments, we used BNC (1k sentences, 90% for training and 10% for testing) and WSJ (section 2 for training and section 22 for testing). This was a small-scale experiment, but large enough to be relevant for low-resource languages. We trained the model with an MH sampler for 1k iterations. Table 3.4 shows the parsing results for the test set. We compared our model with standard CFG and Bayesian TSG models implemented by us. The TSG results are obtained by setting $a_X = 0$ in our TIG model for all X, following the previous work.

[2]http://nlp.cs.nyu.edu/evalb/

[3]Results from [10].

method	(b_1, b_2)	# iterations	# rules (# aux. trees)	F1
CFG	-	-	5957 (-)	64.99
TSG	-	1k	9268 (0)	77.19
TIG (proposed)	(1,1)	1k	7988 (2)	**78.54**
	(100,1)		7462 (16)	75.44
	(100,100)		8057 (15)	77.98
TSG	-	5k	9433 (0)	77.34
TIG (proposed)	(1,1)	5k	8158 (2)	76.78
	(100,1)		7438 (17)	75.02
	(100,100)		8143 (15)	76.03

Table 3.5: Detailed comparison between CFG, TSG and TIG on small dataset.

As shown in Table 3.4, our TIG model successfully outperformed CFG and TSG. Obviously, simple CFG is not powerful for modeling syntax trees. TSG strongly outperformed CFG, however, the TIG model obtained the best F1 score, and reduced the grammar size by approximately 14% compared with TSG. Therefore, adding an insertion operator is helpful for modeling syntax trees accurately. This suggests that adding an insertion operator is helpful for modeling syntax trees accurately. The TSG model described in [10] is similar to ours. They reported an F1 score of 78.40 (the score of our TSG model was 77.19). We speculate that the performance gap is due to data preprocessing such as the treatment of rare words.

Table 3.5 shows the results of a detailed comparison between CFG, TSG and TIG. In Table 3.4, we can see that only a small number of auxiliary trees have a great impact on reducing the grammar size. Surprisingly, there are many fewer auxiliary trees than initial trees. We believe this to be due to the model becoming stuck during the training and our restricted assumption of simple auxiliary trees.

The prior value (b_1, b_2) controls insertion probability indirectly. Compared with the results of $(b_1, b_2) = (1, 1)$ with $(100, 1)$, a higher b_1 value encouraged the insertion operator and further reduced the grammar size, but the performance worsened. However, our model with a proper prior value achieved comparable results to those of the state-of-the-art parsers [35] for a small dataset.

We tried training the model with an MH sampler for 1k and 5k iterations. As shown in Table 3.5, we found that training the model for longer iterations

	# rules (# aux. trees)	F1
CFG	35374 (-)	71.0
TSG	80026 (0)	85.0
TIG (proposed)	65099 (25)	84.8
[38]	-	82.6[4]
[10]	-	85.3

Table 3.6: Full treebank data experiments.

increased the likelihood, but the performance deteriorated. This suggests that our model overfit the training data for long iterations.

3.5.2 Full dataset

We applied the model to the standard WSJ Penn Treebank setting (section 2-21 for training and section 23 for testing). The parsing results are shown in Table 3.6. We initialized the model with the final sample of the small data experiments, and trained the model with an MH sampler for 3.5k iterations. We set $(b_1, b_2) = (100, 100)$.

For full treebank dataset, the TIG model obtained nearly identical results to the TSG model, making the grammar size approximately 19% smaller than TSG. Compared with conventional Bayesian TSG models [38, 10], our model outperformed [38] and obtained comparable results to [10]. As noted in [10], there is a mixing problem with the MH sampler, i.e., the more the data size increases, the more difficult it becomes for our sampler to escape the mode since elementary tree count increases. We believe that more sophisticated inference techniques are needed if we are to obtain better results.

Table 3.7 shows examples of lexicalized auxiliary trees obtained by the TIG model for full treebank data. We can see that punctuation ("–", ",", ";", and ":") and adverbs (ADVP and RB) tend to be inserted in other trees. Punctuation and adverbs appear in various positions in English sentences. Our results suggest that rather than treat those words as substitutions, it is more reasonable to consider these words as an "insertion", which is intuitively understandable.

[4]Results on length ≤ 40.

(N$\bar{\text{P}}$ (N$\bar{\text{P}}$) (: –))
(NP (NP) (ADVP (RB aloft)))
(N$\bar{\text{P}}$ (N$\bar{\text{P}}$) (ADVP (RB respectively)))
(P$\bar{\text{P}}$ (P$\bar{\text{P}}$) (, ,))
(FR$\bar{\text{A}}$G (, ,) (FR$\bar{\text{A}}$G))
(V$\bar{\text{P}}$ (V$\bar{\text{P}}$) (RB then))
(V$\bar{\text{P}}$ (V$\bar{\text{P}}$) (RB not))
(Q$\bar{\text{P}}$ (Q$\bar{\text{P}}$) (IN of))
(SB$\bar{\text{A}}$R (SB$\bar{\text{A}}$R) (RB not))
($\bar{\text{S}}$ ($\bar{\text{S}}$) (RB so))
($\bar{\text{S}}$ ($\bar{\text{S}}$) (: ;))
(ADVP (ADVP) (PP (-NONE- *EMPTY*)))

Table 3.7: Examples of lexicalized auxiliary trees obtained from our model in full treebank data experiments. Nonterminal symbols created by binarization are shown with an over-bar.

3.6 Related Work

Our work is based on recent TSG studies [11, 38]. There are other variants of TSG models to be noted. All-Fragments grammar [2] is a variant of TSG, which maps TSG to an implicit representation of grammar rules rather than using all possible tree fragments explicitly. They combined a symbol refinement approach with TSG and obtained nearly state-of-the-art parsing results. Adaptor grammars [21] is a basic framework for Bayesian TSG, but different from ours. Adoptor grammars permit only terminal symbols at a leaf node, but our model permits nonterminal leaf nodes.

3.7 Summary

We developed a model that automatically induces TIG from a treebank, instead of using heuristic extraction rules. We achieved this by incorporating an probabilistic insertion operator into the conventional TSG model. We also developed grammar decomposition rules to transform TIG derivation into equivalent CFG for efficient training. For a small dataset, our model outperformed CFG and TSG. For a large dataset, our model achieved comparable parsing results to the TSG model, making the number of grammars much smaller than TSG.

Chapter 4

Statistical Induction of Symbol-Refined Tree Substitution Grammars

4.1 Introduction

Syntactic parsing has played a central role in natural language processing. The resulting syntactic analysis can be used for various applications such as machine translation [16, 13], sentence compression [11, 49], and question answering [46]. Probabilistic context-free grammar (PCFG) underlies many statistical parsers, however, it is well known that the PCFG rules extracted from treebank data via maximum likelihood estimation do not perform well due to unrealistic context-free assumptions [26].

In recent years, there has been an increasing interest in tree substitution grammar (TSG) as an alternative to CFG for modeling syntax trees [38, 45, 10]. TSG is a natural extension of CFG in which nonterminal symbols can be rewritten (substituted) with arbitrarily large tree fragments. These tree fragments have great advantages over tiny CFG rules since they can capture non-local contexts explicitly such as predicate-argument structures, idioms and grammatical agreements [10]. Previous work on TSG parsing [10, 38, 2] has consistently shown that a probabilistic TSG (PTSG) parser is significantly more accurate than a PCFG parser, but is still inferior to state-of-the-art parsers (e.g., the Berkeley parser [35] and the Charniak parser [6]). One major drawback of TSG is that the context-free assumptions still remain at substitution sites, that is, TSG tree fragments are

generated that are conditionally independent of all others given root nonterminal symbols. Furthermore, when a sentence is unparsable with large tree fragments, the PTSG parser usually uses naive CFG rules derived from its backoff model, which diminishes the benefits obtained from large tree fragments.

On the other hand, current state-of-the-art parsers use *symbol refinement* techniques [22, 12, 31]. Symbol refinement is a successful approach for weakening context-free assumptions by dividing coarse treebank symbols (e.g. NP and VP) into subcategories, rather than extracting large tree fragments. As shown in several studies on TSG parsing [51, 2], large tree fragments and symbol refinement work complementarily for syntactic parsing. For example, the work presented in [2] have reported that deterministic symbol refinement with heuristics helps improve the accuracy of a TSG parser.

In this paper, we propose Symbol-Refined Tree Substitution Grammars (SR-TSGs) for syntactic parsing. SR-TSG is an extension of the conventional TSG model where each nonterminal symbol can be refined (subcategorized) to fit the training data. Our work differs from previous studies in that we focus on a unified model where TSG rules and symbol refinement are learned from training data in a fully automatic and consistent fashion. We also propose a novel probabilistic SR-TSG model with the hierarchical Pitman-Yor Process [37], namely a sort of nonparametric Bayesian model, to encode backoff smoothing from a fine-grained SR-TSG to simpler CFG rules, and develop an efficient training method based on blocked MCMC sampling.

Our SR-TSG parser achieves an F1 score of 92.4% in the WSJ English Penn Treebank parsing task, which is a 7.7 point improvement over a conventional Bayesian TSG parser, and superior to state-of-the-art discriminative reranking parsers.

4.2 Background

A TSG consists of 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 *start nonterminal symbol* and R is a set of productions (a.k.a. rules). The productions take the form of *elementary trees* i.e., tree fragments of height ≥ 1. The root and internal nodes of the elementary trees are labeled with nonterminal symbols, and leaf nodes are labeled with either terminal or nonterminal symbols. Nonterminal

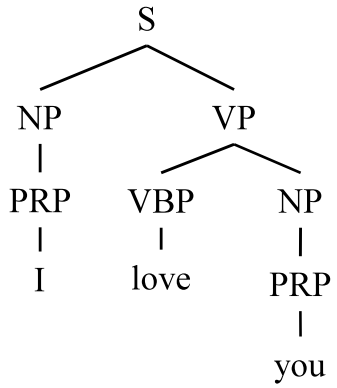


Figure 4.1: Example parse tree.

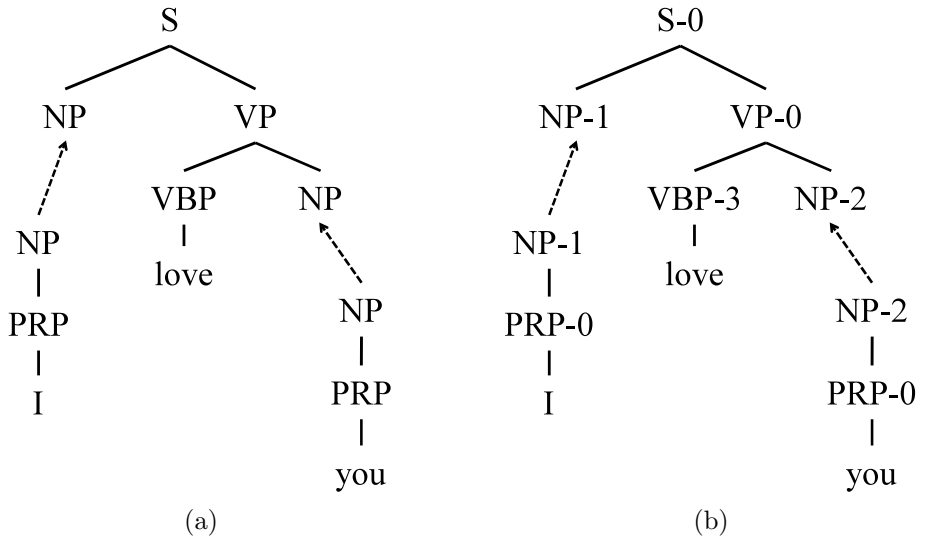


Figure 4.2: (a) Example TSG derivation. (b) Example SR-TSG derivation. The refinement annotation is hyphenated with a nonterminal symbol.

leaves are referred to as *frontier nonterminals*, and form the substitution sites to be combined with other elementary trees.

A *derivation* is a process of forming a parse tree. It starts with a root symbol and rewrites (substitutes) nonterminal symbols with elementary trees until there are no remaining frontier nonterminals. Figure 4.1 shows an example parse tree and Figure 4.2a shows its example TSG derivation. Since different derivations may produce the same parse tree, recent work on TSG induction [38, 10] employs a probabilistic model of a TSG and predicts derivations from observed parse trees in an unsupervised way.

A Probabilistic Tree Substitution Grammar (PTSG) assigns a probability to each rule in the grammar. The probability of a derivation is defined as the product of the probabilities of its component elementary trees as follows.

$$p(\mathbf{e}) = \prod_{x \to e \in \mathbf{e}} p(e \,|x)$$

where $\mathbf{e} = (e_1, e_2, \ldots)$ is a sequence of elementary trees used for the derivation, $x = root(e)$ is the root symbol of e, and $p(e \,|x)$ is the probability of generating e given its root symbol x. As in a PCFG, e is generated conditionally independent of all others given x.

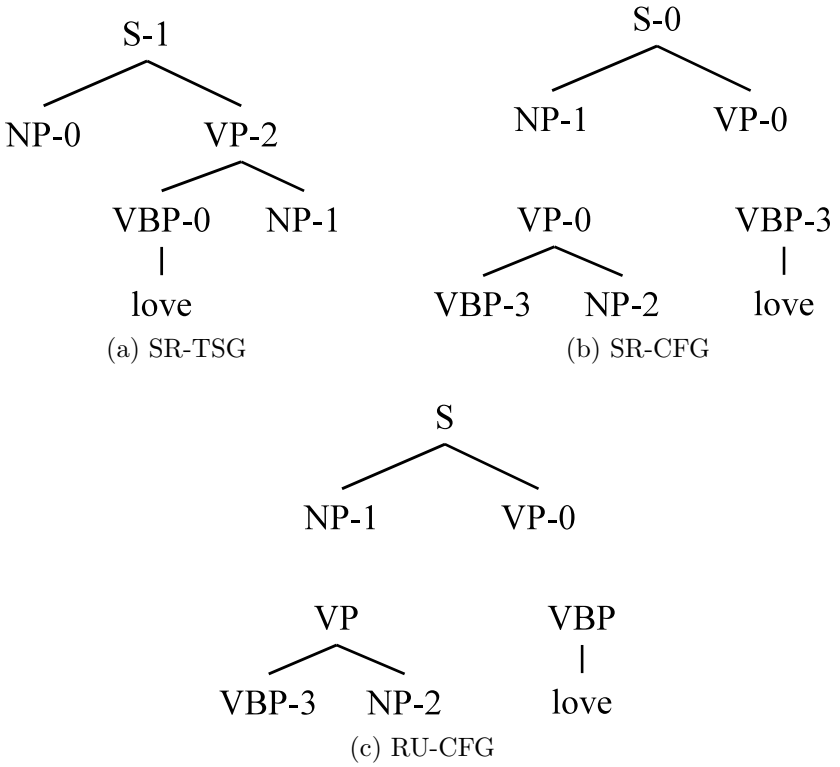
The posterior distribution over elementary trees given a parse tree t can be computed by using the Bayes' rule:

$$p(\mathbf{e} \,|t) \propto p(t \,|\mathbf{e})\, p(\mathbf{e})$$

where $p(t \,|\mathbf{e})$ is either equal to 1 (when t and $\mathbf{e}$ are consistent) or 0 (otherwise). Therefore, the task of TSG induction from parse trees turns out to consist of modeling the prior distribution $p(\mathbf{e})$. Recent work on TSG induction defines $p(\mathbf{e})$ as a nonparametric Bayesian model such as the Dirichlet Process [14] or the Pitman-Yor Process to encourage sparse and compact grammars.

4.3 Symbol-Refined Tree Substitution Grammars

In this section, we propose Symbol-Refined Tree Substitution Grammars (SR-TSGs). Our SR-TSG model is an extension of the conventional TSG model where every symbol of the elementary trees can be refined to fit the training data. Figure 4.2b shows an example of SR-TSG derivation. As with previous



(a) SR-TSG (b) SR-CFG

(c) RU-CFG

Figure 4.3: Example three-level backoff.

work on TSG induction, our task is the induction of SR-TSG derivations from a corpus of parse trees in an unsupervised fashion. That is, we wish to infer the symbol subcategories of every node and substitution site (i.e., nodes where substitution occurs) from parse trees. Extracted rules and their probabilities can be used to parse new raw sentences.

4.3.1 Probabilistic Model

We define a probabilistic model of an SR-TSG based on the Pitman-Yor Process (PYP) [37], namely a sort of nonparametric Bayesian model. The PYP produces power-law distributions, which have been shown to be well-suited for such uses as language modeling [44], and TSG induction [10]. One major issue as regards modeling an SR-TSG is that the space of the grammar rules will be very sparse since SR-TSG allows for arbitrarily large tree fragments and also an arbitrarily large set of symbol subcategories. To address the sparseness problem, we employ

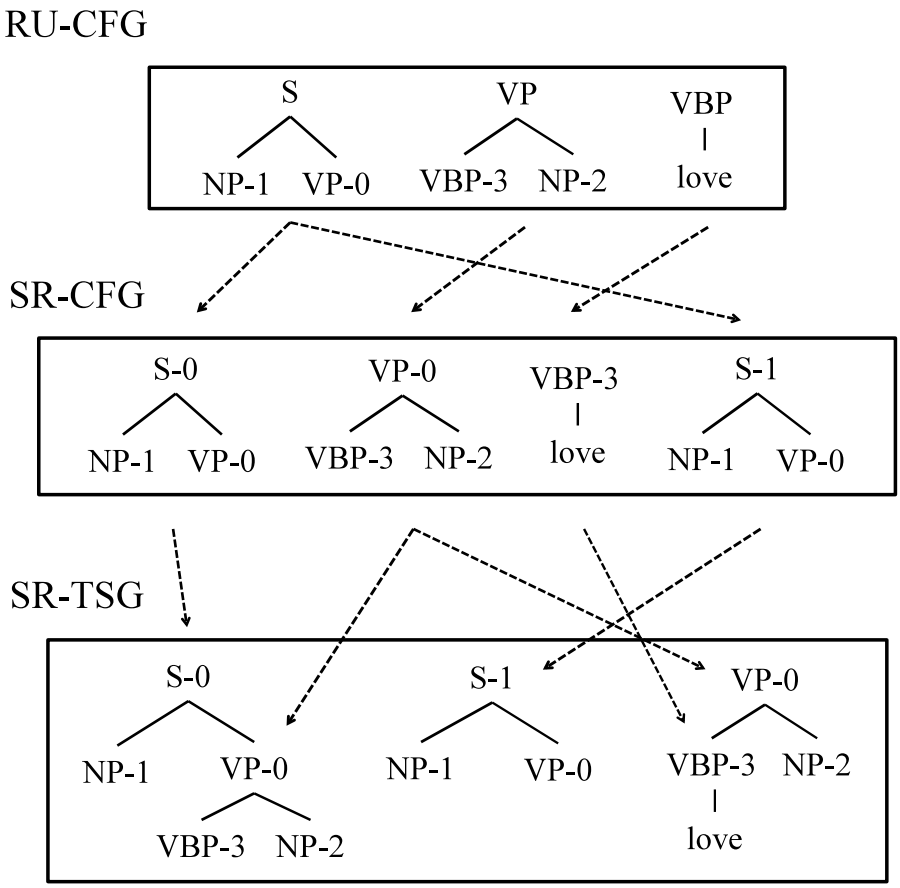


Figure 4.4: Three-level hierarchy of SR-TSG.

a hierarchical PYP to encode a backoff scheme from the SR-TSG rules to simpler CFG rules, inspired by recent work on dependency parsing [3].

Our model consists of a three-level hierarchy. Figure 4.3 shows an example of the SR-TSG rule and its backoff tree fragments. Figure 4.3 shows the relation between our three-level hierarchy.

The topmost level of our model is a distribution over the SR-TSG rules as follows.

$$\begin{aligned} e\,|x_k &\sim G_{x_k} \\ G_{x_k} &\sim \mathrm{PYP}\left(d_{x_k}, \theta_{x_k}, P^{\text{sr-tsg}}\left(\cdot\,|x_k\right)\right) \end{aligned}$$

where x_k is a refined root symbol of an elementary tree e, while x is a raw nonterminal symbol in the corpus and $k = 0, 1, \ldots$ is an index of the symbol subcategory. Suppose x is NP and its symbol subcategory is 0, then x_k is NP_0. The PYP has three parameters: $(d_{x_k}, \theta_{x_k}, P^{\text{sr-tsg}})$. $P^{\text{sr-tsg}}\left(\cdot\,|x_k\right)$ is a *base distribution* over infinite space of symbol-refined elementary trees rooted with x_k, which provides the backoff probability of e. The remaining parameters d_{x_k} and θ_{x_k} control the strength of the base distribution.

The backoff probability $P^{\text{sr-tsg}}\left(e\,|x_k\right)$ is given by the product of *symbol-refined CFG (SR-CFG)* rules that e contains as follows.

$$\begin{aligned} P^{\text{sr-tsg}}\left(e\,|x_k\right) &= \prod_{f \in F(e)} s_{c_f} \times \prod_{i \in I(e)} \left(1 - s_{c_i}\right) \\ &\times H\left(\text{cfg-rules}\left(e\,|x_k\right)\right) \\ \alpha\,|x_k &\sim H_{x_k} \\ H_{x_k} &\sim \mathrm{PYP}\left(d_x, \theta_x, P^{\text{sr-cfg}}\left(\cdot\,|x_k\right)\right), \end{aligned}$$

where $F\left(e\right)$ is a set of frontier nonterminal nodes and $I\left(e\right)$ is a set of internal nodes in e. c_f and c_i are nonterminal symbols of nodes f and i, respectively. s_c is the probability of stopping the expansion of a node labeled with c. SR-CFG rules are CFG rules where every symbol is refined, as shown in Table 4.3. The function $\text{cfg-rules}\left(e\,|x_k\right)$ returns the SR-CFG rules that e contains, which take the form of $x_k \to \alpha$. Each SR-CFG rule α rooted with x_k is drawn from the backoff distribution H_{x_k}, and H_{x_k} is produced by the PYP with parameters:

$\left(d_x, \theta_x, P^{\text{sr-cfg}}\right)$. This distribution over the SR-CFG rules forms the second level hierarchy of our model.

The backoff probability of the SR-CFG rule, $P^{\text{sr-cfg}}(\alpha \,|x_k)$, is given by the *root-unrefined CFG (RU-CFG)* rule as follows,

$$\begin{aligned} P^{\text{sr-cfg}}(\alpha \,|x_k) &= I\left(\text{root-unrefine}(\alpha \,|x_k)\right) \\ \alpha \,|x &\sim I_x \\ I_x &\sim \text{PYP}\left(d'_x, \theta'_x, P^{\text{ru-cfg}}(\cdot \,|x)\right), \end{aligned}$$

where the function root-unrefine$(\alpha \,|x_k)$ returns the RU-CFG rule of α, which takes the form of $x \to \alpha$. The RU-CFG rule is a CFG rule where the root symbol is unrefined and all leaf nonterminal symbols are refined, as shown in Table 4.3. Each RU-CFG rule α rooted with x is drawn from the backoff distribution I_x, and I_x is produced by a PYP. This distribution over the RU-CFG rules forms the third level hierarchy of our model. Finally, we set the backoff probability of the RU-CFG rule, $P^{\text{ru-cfg}}(\alpha \,|x)$, so that it is uniform as follows.

$$P^{\text{ru-cfg}}(\alpha \,|x) = \frac{1}{|x \to \cdot|}.$$

where $|x \to \cdot|$ is the number of RU-CFG rules rooted with x. Overall, our hierarchical model encodes backoff smoothing consistently from the SR-TSG rules to the SR-CFG rules, and from the SR-CFG rules to the RU-CFG rules. As shown in [3, 9], the parsing accuracy of the TSG model is strongly affected by its backoff model. The effects of our hierarchical backoff model on parsing performance are evaluated in the experiment section.

4.4 Inference

We use Markov Chain Monte Carlo (MCMC) sampling to infer the SR-TSG derivations from parse trees. MCMC sampling is a widely used approach for obtaining random samples from a probability distribution. In our case, we wish to obtain derivation samples of an SR-TSG from the posterior distribution, $p(\mathbf{e} \,|\mathbf{t}, \mathbf{d}, \boldsymbol{\theta}, \mathbf{s})$.

The inference of the SR-TSG derivations corresponds to inferring two kinds of latent variables: latent symbol subcategories and latent substitution sites.

We first infer latent symbol subcategories for every symbol in the parse trees, and then infer latent substitution sites stepwise. During the inference of symbol subcategories, every internal node is fixed as a substitution site. After that, we unfix that assumption and infer latent substitution sites given symbol-refined parse trees. This stepwise learning is simple and efficient in practice, but we believe that the joint learning of both latent variables is possible, and we will deal with this in future work. Here we describe each inference algorithm in detail.

4.4.1 Inference of Symbol Subcategories

For the inference of latent symbol subcategories, we adopt split and merge training [35] as follows. In each split-merge step, each symbol is split into at most two subcategories. For example, every NP symbol in the training data is split into either NP_0 or NP_1 to maximize the posterior probability. After convergence, we measure the loss of each split symbol in terms of the likelihood incurred when removing it, then the smallest 50% of the newly split symbols as regards that loss are merged to avoid overfitting. The split-merge algorithm terminates when the total number of steps reaches the user-specified value.

In each splitting step, we use two types of blocked MCMC algorithm: the *sentence-level* blocked Metropolis-Hastings (MH) sampler and the *tree-level* blocked Gibbs sampler, while [35] use a different MLE-based model and the EM algorithm. Our sampler iterates sentence-level sampling and tree-level sampling alternately.

The sentence-level MH sampler is a recently proposed algorithm for grammar induction [21, 10]. In this work, we apply it to the training of symbol splitting. The MH sampler consists of the following three steps:

For each sentence,

1. Calculate the inside probability [27] in a bottom-up manner.
2. Sample a derivation tree in a top-down manner.
3. Accept or reject the derivation sample by using the MH test. See [10] for details. This sampler simultaneously updates blocks of latent variables associated with a sentence, thus it can find MAP solutions efficiently.

The tree-level blocked Gibbs sampler focuses on the type of SR-TSG rules and simultaneously updates all root and child nodes that are annotated with the same

SR-TSG rule. For example, the sampler collects all nodes that are annotated with $\mathrm{S}_0 \rightarrow \mathrm{NP}_1\mathrm{VP}_2$, then updates those nodes to another subcategory such as $\mathrm{S}_0 \rightarrow \mathrm{NP}_2\mathrm{VP}_0$ according to the posterior distribution. This sampler is similar to table label resampling [23], but differs in that our sampler can update multiple table labels simultaneously when multiple tables are labeled with the same elementary tree. The tree-level sampler also simultaneously updates blocks of latent variables associated with the type of SR-TSG rules, thus it can find MAP solutions efficiently.

4.4.2 Inference of Substitution Sites

After the inference of symbol subcategories, we use Gibbs sampling to infer the substitution sites of parse trees as described in [11, 38]. We assign a binary variable to each internal node in the training data, which indicates whether that node is a substitution site or not. For each iteration, the Gibbs sampler works by sampling the value of each binary variable in random order. See [10] for details.

During the inference, our sampler ignores the symbol subcategories of internal nodes of elementary trees since they do not affect the derivation of the SR-TSG. For example, the elementary trees " $(\mathrm{S}_0\ (\mathrm{NP}_0\ \mathrm{NNP}_0)\ \mathrm{VP}_0)$ "and " $(\mathrm{S}_0\ (\mathrm{NP}_1\ \mathrm{NNP}_0)\ \mathrm{VP}_0)$ "are regarded as being the same when we calculate the generation probabilities according to our model. This heuristics is helpful for finding large tree fragments and learning compact grammars.

4.4.3 Hyperparameter Estimation

We treat hyperparameters $\{\mathbf{d}, \boldsymbol{\theta}\}$ as random variables and update their values for every MCMC iteration. We place a prior on the hyperparameters as follows.

- $d \sim \mathrm{Beta}\,(1.0, 1.0)$
- $\theta \sim \mathrm{Gamma}\,(1.0, 1.0)$

The values of d and θ are optimized with the auxiliary variable technique [43].

4.5 Experiment

4.5.1 Settings

4.5.1.1 Data Preparation

We ran experiments on the Wall Street Journal (WSJ) portion of the English Penn Treebank data set [30], using a standard data split (sections 2–21 for training, 22 for development and 23 for testing). We also used section 2 as a *small* training set for evaluating the performance of our model under low-resource conditions. Henceforth, we distinguish the small training set (section 2) from the full training set (sections 2-21). The treebank data is right-binarized [31] to construct grammars with only unary and binary productions. We replace lexical words with count ≤ 5 in the training data with one of 50 unknown words using lexical features, following [35]. We also split off all the function tags and eliminated empty nodes from the data set, following [22].

4.5.1.2 Training and Parsing

For the inference of symbol subcategories, we trained our model with the MCMC sampler by using 6 split-merge steps for the full training set and 3 split-merge steps for the small training set. Therefore, each symbol can be subdivided into a maximum of $2^6 = 64$ and $2^3 = 8$ subcategories, respectively. In each split-merge step, we initialized the sampler by randomly splitting every symbol in two subcategories and ran the MCMC sampler for 1000 iterations. After that, to infer the substitution sites, we initialized the model with the final sample from a run on the small training set, and used the Gibbs sampler for 2000 iterations. We estimated the optimal values of the stopping probabilities $\mathbf{s}$ by using the development set.

We obtained the parsing results with the MAX-RULE-PRODUCT algorithm [35] by using the SR-TSG rules extracted from our model. We evaluated the accuracy of our parser by bracketing F1 score of predicted parse trees. We used EVALB[1] to compute the F1 score. In all our experiments, we conducted ten independent runs to train our model, and selected the one that performed best on the development set in terms of parsing accuracy.

[1] http://nlp.cs.nyu.edu/evalb/

Model	F1 (small)	F1 (full)
CFG	61.9	63.6
*TSG	77.1	85.0
SR-TSG ($P^{\text{sr-tsg}}$)	73.0	86.4
SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$)	79.4	89.7
SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$, $P^{\text{ru-cfg}}$)	**81.7**	**91.1**

Table 4.1: Comparison of parsing accuracy with the small and full training sets. *Our reimplementation of [10].

4.5.2 Results and Discussion

4.5.2.1 Comparison of SR-TSG with TSG

We compared the SR-TSG model with the CFG and TSG models as regards parsing accuracy. We also tested our model with three backoff hierarchy settings to evaluate the effects of backoff smoothing on parsing accuracy. Table 4.1 shows the F1 scores of the CFG, TSG and SR-TSG parsers for small and full training sets. In Table 4.1, SR-TSG ($P^{\text{sr-tsg}}$) denotes that we used only the topmost level of the hierarchy. Similary, SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$) denotes that we used only the $P^{\text{sr-tsg}}$ and $P^{\text{sr-cfg}}$ backoff models.

Our best model, SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$, $P^{\text{ru-cfg}}$), outperformed both the CFG and TSG models on both the small and large training sets. This result suggests that the conventional TSG model trained from the vanilla treebank is insufficient to resolve structural ambiguities caused by coarse symbol annotations in a training corpus. As we expected, symbol refinement can be helpful with the TSG model for further fitting the training set and improving the parsing accuracy.

The performance of the SR-TSG parser was strongly affected by its backoff models. For example, the simplest model, $P^{\text{sr-tsg}}$, performed poorly compared with our best model. This result suggests that the SR-TSG rules extracted from the training set are very sparse and cannot cover the space of unknown syntax patterns in the testing set. Therefore, sophisticated backoff modeling is essential for the SR-TSG parser. Our hierarchical PYP modeling technique is a successful way to achieve backoff smoothing from sparse SR-TSG rules to simpler CFG rules, and offers the advantage of automatically estimating the optimal backoff

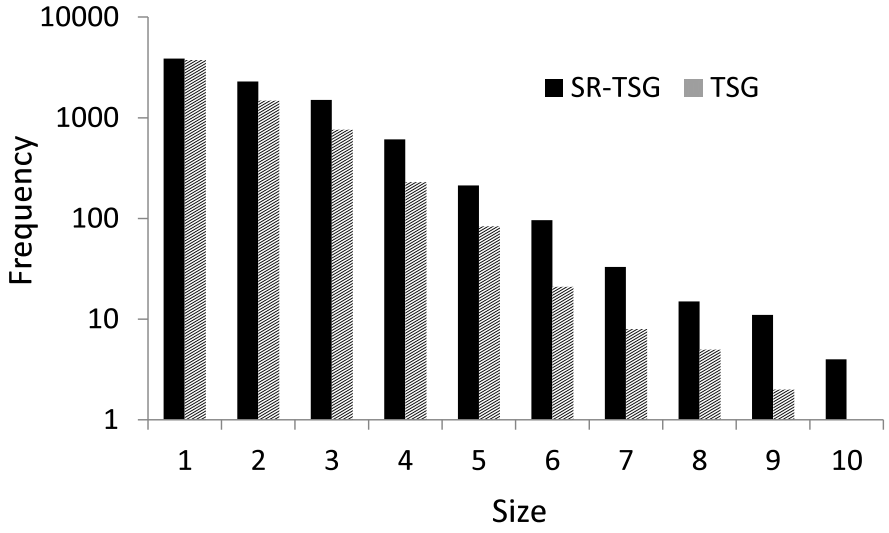


Figure 4.5: Histogram of SR-TSG and TSG rule sizes on the small training set. The size is defined as the number of CFG rules that the elementary tree contains.

probabilities from the training set.

We compared the rule sizes and frequencies of SR-TSG with those of TSG. The rule sizes of SR-TSG and TSG are defined as the number of CFG rules that the elementary tree contains. Figure 4.5 shows a histogram of the SR-TSG and TSG rule sizes (by *unrefined* token) on the small training set. For example, SR-TSG rules: $S_1 \rightarrow NP_0VP_1$ and $S_0 \rightarrow NP_1VP_2$ were considered to be the same token. As shown in Figure 4.5, there are almost the same number of SR-TSG rules and TSG rules with size $= 1$. However, there are more SR-TSG rules than TSG rules with size ≥ 2. This shows that an SR-TSG can use various large tree fragments depending on the context, which is specified by the symbol subcategories.

4.5.2.2 Comparison of SR-TSG with Other Models

We compared the accuracy of the SR-TSG parser with that of conventional high-performance parsers. Table 4.2 shows the F1 scores of an SR-TSG and conventional parsers with the full training set. In Table 4.2, SR-TSG (single) is a standard SR-TSG parser, and SR-TSG (multiple) is a combination of sixteen independently trained SR-TSG models, following the work of [34].

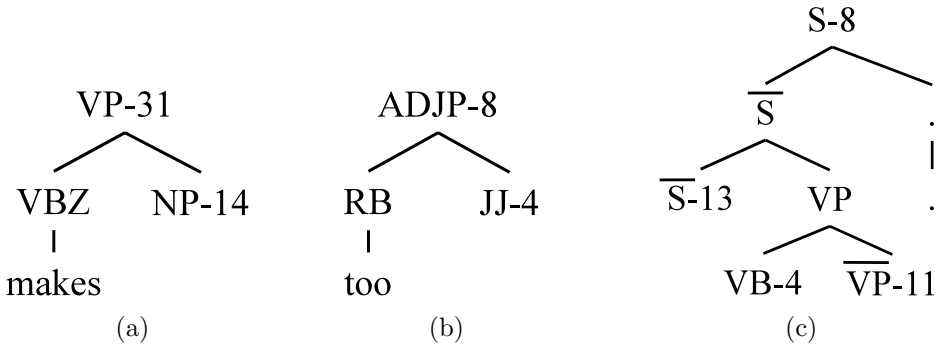
Our SR-TSG (single) parser achieved an F1 score of 91.1%, which is a 6.4 point

Model	F1 (≤ 40)	F1 (all)
TSG (no symbol refinement)		
Post and Gildea (2009)	82.6	-
Cohn et al. (2010)	85.4	84.7
TSG with Symbol Refinement		
Zuidema (2007)	-	*83.8
Bansal et al. (2010)	88.7	88.1
SR-TSG (single)	**91.6**	**91.1**
SR-TSG (multiple)	**92.9**	**92.4**
CFG with Symbol Refinement		
Collins (1999)	88.6	88.2
Petrov and Klein (2007)	90.6	90.1
Petrov (2010)	-	91.8
Discriminative		
Carreras et al. (2008)	-	91.1
Charniak and Johnson (2005)	92.0	91.4
Huang (2008)	92.3	91.7

Table 4.2: Our parsing performance for the testing set compared with those of other parsers. *Results for the development set (≤ 100).

improvement over the conventional Bayesian TSG parser reported by [10]. Our model can be viewed as an extension of Cohn's work by the incorporation of symbol refinement. Therefore, this result confirms that a TSG and symbol refinement work complementarily in improving parsing accuracy. Compared with a symbol-refined CFG model such as the Berkeley parser [35], the SR-TSG model can use large tree fragments, which strengthens the probability of frequent syntax patterns in the training set. Indeed, the few very large rules of our model memorized full parse trees of sentences, which were repeated in the training set.

The SR-TSG (single) is a pure generative model of syntax trees but it achieved results comparable to those of discriminative parsers. It should be noted that



(a) (b) (c)

Figure 4.6: Examples of SR-TSG rules obtained from full treebank data. Nonterminal symbols created by binarization are shown with an over-bar.

discriminative reranking parsers such as [6] and [19] are constructed on a generative parser. The reranking parser takes the k-best lists of candidate trees or a packed forest produced by a baseline parser (usually a generative model), and then reranks the candidates using arbitrary features. Hence, we can expect that combining our SR-TSG model with a discriminative reranking parser would provide better performance than SR-TSG alone.

Recently, [34] has reported that combining multiple grammars trained independently gives significantly improved performance over a single grammar alone. We applied his method (referred to as a TREE-LEVEL inference) to the SR-TSG model as follows. We first trained sixteen SR-TSG models independently and produced a 100-best list of the derivations for each model. Then, we erased the subcategory information of parse trees and selected the best tree that achieved the highest likelihood under the product of sixteen models. The combination model, SR-TSG (multiple), achieved an F1 score of 92.4%, which is a state-of-the-art result for the WSJ parsing task. Compared with discriminative reranking parsers, combining multiple grammars by using the product model provides the advantage that it does not require any additional training. Several studies [15, 50] have proposed different approaches that involve combining k-best lists of candidate trees. We will deal with those methods in future work.

4.5.2.3 Extracted Grammars

Figure 4.6 shows examples of SR-TSG rules obtained from full treebank data. The symbol subcategories of internal nodes are marginalized. Figure 4.6a shows

an example lexicalized tree fragment rooted with a verb phrase (VP). In our model, the nonterminal symbol "NP-14" generates lexical words such as "financing", "changes", "impeachment", "good will", "plan" and "representation", which are frequently appeared with "makes". Similarly, Figure 4.6b shows an example lexicalized tree fragment rooted with an adjective phrase (ADJP). In our model, the nonterminal symbol "JJ-4" generates lexical words such as "steady", "dogged", "timely", "limited" and "high", which are frequently appeared with "too". From the above, it turns our that our SR-TSG model successfully clusters similar words by symbol splitting. Figure 4.6c shows an example unlexicalized tree fragment rooted with "S" (sentence). In our model, the nonterminal symbol "VB-4" generates lexical words such as "comment", "vote", "invest", "work" and "play". These large tree fragments with rich context information are unable to be obtained by previous SR-CFG and TSG models. The SR-TSG model incorporates the strength of symbol refinement with that of large tree fragments.

Let us note the relation between SR-CFG, TSG and SR-TSG. TSG is weakly equivalent to CFG and generates the same set of strings. For example, the TSG rule " S $\rightarrow$ (NP NNP) VP "with probability p can be converted to the equivalent CFG rules as follows: " S $\rightarrow$ $\mathrm{NP}^{\mathrm{NNP}}$ VP "with probability p and " $\mathrm{NP}^{\mathrm{NNP}}$ $\rightarrow$ NNP "with probability 1. From this viewpoint, TSG utilizes surrounding symbols (NNP of $\mathrm{NP}^{\mathrm{NNP}}$ in the above example) as latent variables with which to capture context information. The search space of learning a TSG given a parse tree is $\mathcal{O}\left(2^n\right)$ where n is the number of internal nodes of the parse tree. On the other hand, an SR-CFG utilizes an arbitrary index such as $0, 1, \ldots$ as latent variables and the search space is larger than that of a TSG when the symbol refinement model allows for more than two subcategories for each symbol. Our experimental results confirm that jointly modeling both latent variables using our SR-TSG assists accurate parsing.

Our experiments primarily focus on evaluating parsing accuracy, however, we remark on the decoding time of our SR-TSG parser for the future work. Since the number of SR-TSG rules learned from the training set is much larger than that of SR-CFG and TSG rules, the parsing time for test set increases substantially. Indeed, our SR-TSG parser took more than 2800 minutes to parse 2416 sentences in test set, run on Xeon 5600 3.33GHz. Previous work on CFG with symbol-refinement has introduced coarse-to-fine approach to prune unneccesary grammar rules and acceralate the parsing speed [36]. The basic idea of coarse-to-fine

pruning is as follows. First, parse the sentence with coarse (no symbol-refined) grammar rules. The spans of input sentence that have low posterior probabilities are pruned away. Then reparse the sentence with finer grammar rules considering only unpruned spans. This procedure is iterated from the coarsest grammar rules to the finest ones. Surprisingly, Petrov et al. [36] showed that their SR-CFG parser achieved more than 100 times parsing speed up by the coarse-to-fine pruning with no loss in test set accuracy, which made their parser practical. We believe that our SR-TSG parser could be benefit from the coarse-to-fine pruning for parsing speed up and leave it to the future work.

Another direction is a *lexicalization* of tree substitution grammars. Lexicalized TSG is defined as TSG where each tree fragment has at least one lexical word as a leaf node. Since every tree fragments in the grammars has an anchor word, we can easily detect unneccesary tree fragments for parsing the input sentence. Besides, supertagging is a widely used technique for acceralating parsing speed of lexicalized grammars. It assigns candidate lexicalized tree fragments to the input words as a sequence tagging. Tree fragments that have low poeterior probabilities are discarded, which leads to substantially faster parsing times. We could model lexicalized SR-TSG and apply supertagging technique for decoding.

4.6 Related Work

Several studies have combined TSG induction and symbol refinement. An adaptor grammar [24] is a sort of nonparametric Bayesian TSG model with symbol refinement, and is thus closely related to our SR-TSG model. However, an adaptor grammar differs from ours in that all its rules are *complete*: all leaf nodes must be terminal symbols, while our model permits nonterminal symbols as leaf nodes. Furthermore, adaptor grammars have largely been applied to the task of unsupervised structural induction from raw texts such as morphology analysis, word segmentation [23], and dependency grammar induction [9], rather than constituent syntax parsing.

An all-fragments grammar [2] is another variant of TSG that aims to utilize all possible subtrees as rules. It maps a TSG to an implicit representation to make the grammar tractable and practical for large-scale parsing. The manual symbol refinement described in [26] was applied to an all-fragments grammar and this improved accuracy in the English WSJ parsing task. As mentioned in the

introduction, our model focuses on the automatic learning of a TSG and symbol refinement without heuristics.

4.7 Summary

We have presented an SR-TSG, which is an extension of the conventional TSG model where each symbol of tree fragments can be automatically subcategorized to address the problem of the conditional independence assumptions of a TSG. We proposed a novel backoff modeling of an SR-TSG based on the hierarchical Pitman-Yor Process and sentence-level and tree-level blocked MCMC sampling for training our model. Our best model significantly outperformed the conventional TSG and achieved state-of-the-art result in a WSJ parsing task. Future work will involve examining the SR-TSG model for different languages and for unsupervised grammar induction.

Chapter 5

Pseudo Blocked Subtree Sampler for Statistical Grammar Induction

5.1 Introduction

Gibbs sampling [17] is a widely used technique for grammatical inference from treebank data. It is one of the Markov Chain Monte Carlo methods, that is, it generates samples from the given probability distribution based on constructing a Markov Chain that has the target distribution as its equilibrium distribution. In statistical grammar induction, the sampler generates a set of grammatical rules from the posterior distribution given parse trees to find the optimal set that has the highest posterior probability [11, 41, 42].

Gibbs sampler is characterized by sampling one variable at a time from the conditional distribution of each variable. Therefore, it has a great advantage when sampling from joint distribution is computationally expensive. However, it has been shown that Gibbs sampler for grammar induction tends to get stuck in local optima due to strong dependency among variables of tree structure [10].

A blocked sampling is a well-known technique to improve sampling efficiency by grouping two or more variables together and sampling from their joint distribution conditioned on all other variables, rather than sampling one variable at a time. However, previous work on blocked MCMC sampling for grammar induction is restricted to be applicable to only particular models, e.g., every variables are assumed to be binary.

To tackle this problem, we propose a *pseudo* blocked subtree sampler for statistical grammar induction. Our aim is to provide a blocked sampler which is widely applicable to many grammatical models similar to Gibbs sampler and also able to simultaneously update variables as many as possible. Our proposed method differs from standard blocked samplers in that we construct a blocked sampler and a standard MCMC sampler independently, and then run both samplers alternately. Besides, the proposed method automatically determines the appropriate size of blocks and updates all variables in the block simultaneously, instead of giving the blocks manually as in previous work. Therefore, our method is independent from particular grammatical models and able to handle variables each of which has multiple value. It should be noted that our method is referred to as a *pseudo* blocked subtree sampler since it does not construct an exact Markov chain that has the target distribution as its equilibrium distribution, thus it is an approximate sampler from the posterior distribution.

We applied the proposed method to the induction of symbol-refined context-free grammars on the English Penn Treebank data. Our method obtained better results than standard Gibbs sampling and blocked Gibbs sampling regardless of the amount of data.

This chapter is organized as follows. In Section 5.2, we give an overview of symbol-refined context-free grammars and Gibbs sampling used in our method. In Section 5.3, we describe the theory and algorithm of our pseudo blocked subtree sampling. In Section 5.4, we show experimental results of our method. In Section 5.5we present related work, and Section 5.6 concludes the chapter.

5.2 Background

5.2.1 Symbol-Refined Context-Free Grammars

Symbol-Refined Context-Free Grammar (SR-CFG) [31, 35] is an extension of standard CFG where each symbol (e.g. NP and VP) is automatically divided into subcategories such as NP-0 and NP-1 to fit training data, rather than extracting large tree fragments. Our proposed method is applicable to many probabilistic grammars, however, we use SR-CFG as an example since it underlies current state-of-the-art parsers [34, 42]. Figure 5.1 shows an example parse tree and Figure 5.2 shows an example derivation of SR-CFG.

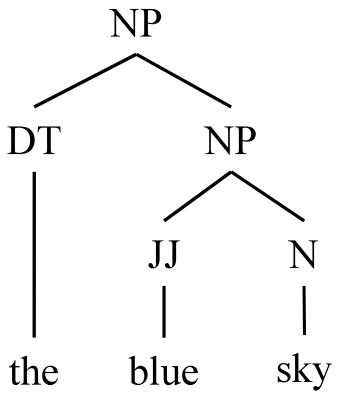


Figure 5.1: Example parse tree.

Figure 5.2: Example SR-CFG derivation. Dotted line denotes a process of rewriting nonterminal symbol.

A SR-CFG rule e takes the form of $A_x \rightarrow B_y C_z$, where A, B and C are nonterminal symbols, and x, y and z are symbol subcategories, respectively.

We design a probabilistic model of SR-CFG based on Pitman-Yor process [37] as follows.

$$\begin{aligned} e\,|A_x &\sim G_{A_x} \\ G_{A_x} &\sim \mathrm{PYP}\left(d_{A_x}, \theta_{A_x}, P_0\left(\cdot\,|A_x\right)\right) \end{aligned}$$

where A_x is a root nonterminal symbol of a SR-CFG rule e and $x = 0, 1, \ldots$ is an index of the symbol subcategory. The Pitman-Yor process has three parameters: $(d_{A_x}, \theta_{A_x}, P_0)$. $P_0\left(\cdot\,|A_x\right)$ is a *base distribution* over a space of SR-CFG rules rooted with A_x, which provides the backoff probability of e. The remaining parameters d_{x_k} and θ_{x_k} control the strength of the base distribution.

We set the base distribution as follows.

$$P_0\left(e\,|A_x\right) = P_{\mathrm{MLE}}\left(A \rightarrow BC\right) \times \frac{1}{|y|} \times \frac{1}{|z|}$$

where $A \rightarrow BC$ is an *unrefined* CFG rule of e and $P_{\mathrm{MLE}}\left(e\right)$ is a maximum-likelihood estimate of e, which is obtained from the training corpus. $|y|$ and $|z|$ are the number of subcategories of y and z, respectively.

In case of a SR-CFG rule that has a single child node, i.e., $e : A_x \rightarrow B_y$, the base distribution is similarly defined as follows.

$$P_0\left(e\,|A_x\right) = P_{\mathrm{MLE}}\left(A \rightarrow B\right) \times \frac{1}{|y|}$$

We also define the base distribution for a lexical rule $e : A_x \rightarrow w$ as follows.

$$P_0\left(e\,|A_x\right) = P_{\mathrm{MLE}}\left(A \rightarrow w\right)$$

where w is a lexical word.

Integrating out all possible values of G_X, the resulting distribution is computed as follows.

$$\begin{aligned} P\left(e_i\,|\mathbf{e}_{1:i-1}, A_x, d_{A_x}, \theta_{A_x}\right) &= \alpha_{e_i,A_x} + \beta_{A_x} P_0\left(e_i\,|A_x\right) \\ \alpha_{e_i,A_x} &= \frac{n_{e_i,A_x} - d_{A_x} \cdot t_{e_i,A_x}}{\theta_{A_x} + \Sigma_e n_{e,A_x}} \\ \beta_{A_x} &= \frac{\theta_{A_x} + d_{A_x} \cdot \Sigma_e t_{e,A_x}}{\theta_{A_x} + \Sigma_e n_{e,A_x}} \end{aligned}$$

where $\mathbf{e}_{1:i-1} = e_1, \ldots, e_{i-1}$ are previously generated SR-CFG rules, and $n_{e_i,X}$ is the number of times e_i has been used in $\mathbf{e}_{1:i-1}$. $t_{e_i,X}$ is the number of *tables* labeled with e_i in the *Chinese Restaurant*. $n_{\cdot,X} = \sum_e n_{e,X}$ and $t_{\cdot,X} = \sum_e t_{e,X}$ are the total counts of rules and tables, respectively.

The SR-CFG model is assumed to be context-free. Therefore, the probability of generating a parse tree is simply the product of the rule probabilities as follows.

$$P\left(\mathbf{e}\right) = \prod_{j=1}^{|\mathbf{e}|} P\left(e_j\right)$$

where $\mathbf{e}$ is a set of SR-CFG rules used in the derivation.

5.2.2 Statistical SR-CFG Induction by Gibbs Sampling

Recall that our task is an induction of grammatical rules from treebank data. The posterior distribution over SR-CFG rules $\mathbf{e}$ given a parse tree t is computed by using the Bayes' rule:

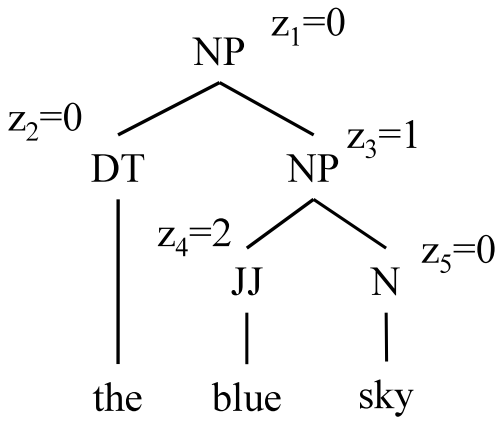


Figure 5.3: Alternative representation of Figure 5.2 which assigns latent variables to each node.

$$p\left(\mathbf{e}\,|t\right) \propto p\left(t\,|\mathbf{e}\right) p\left(\mathbf{e}\right)$$

where $p\left(t\,|\mathbf{e}\right)$ is either equal to 1 (when t and $\mathbf{e}$ are consistent) or 0 (otherwise).

Rather than dealing with $\mathbf{e}$ directly, we introduce a latent variable for each node in a parse tree to represent a SR-CFG derivation. Figure 5.3 shows the alternative representation of Figure 5.2 which assigns a latent variable to each node in a parse tree. In Figure 5.3, each latent variable represents symbol subcategory. It should be noted that other probabilistic grammars such as Tree Substitution Grammars (TSG) and Tree Insertion Grammars (TIG) can be represented as the above latent variable representation [11, 41, 48].

The optimal set of grammatical rules and parameters: $\hat{\mathbf{z}}, \hat{\Theta}$ are obtained by maximizing the posterior probability given parse trees $\{t\}$ as follows.

$$\hat{\mathbf{z}}, \hat{\Theta} = \underset{\mathbf{z},\Theta}{\operatorname{argmax}}\, P\left(\mathbf{z}\,|\{t\}\,;\Theta\right) P\left(\Theta\right)$$

where Θ is a set of parameters of the SR-CFG model. The derivation of SR-CFG is uniquely determined from $\hat{\mathbf{z}}$.

Gibbs sampling is widely used for the inference of latent variables and parameters. As described above, Gibbs sampler performs one variable at a time from the conditional distribution of each variable. The basic procedure is listed as follows:

1. Set initial variables $\mathbf{z}_0$ and initial parameters Θ_0.

2. Iterate over the following processes by reaching the user-defined repeat count:

   (a) Pick $z \in \mathbf{z}$ at random.

   (b) Sample the value of z according to $P\left(z \,|\mathbf{z} \setminus z, \Theta\right)$ and update the value of z.

   (c) Sample the value of $\theta \in \Theta$ according to $P\left(\theta \,|\mathbf{z}, \Theta \setminus \theta\right)$ and update the value of θ.

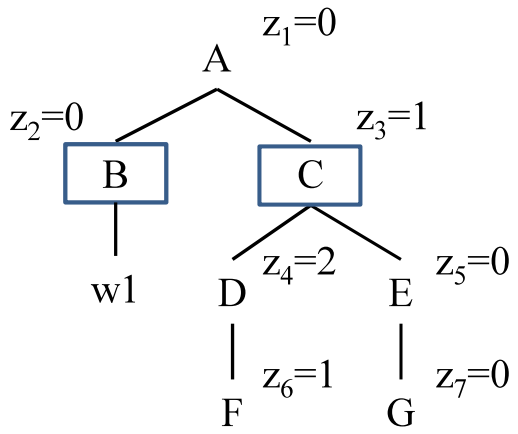
Since a SR-CFG rule is composed of several variables, it might take several steps to replace a SR-CFG rule e with an another rule e'. Thus, if there exists a low-likelihood pass from e to e', it is difficult to reach e from e' within a finite number of iterations. A blocked Gibbs sampler improves the search efficiency by sampling many variables at a time, however, it also gets stuck in a local optimum as the training corpus becomes large.

5.3 Proposed Method

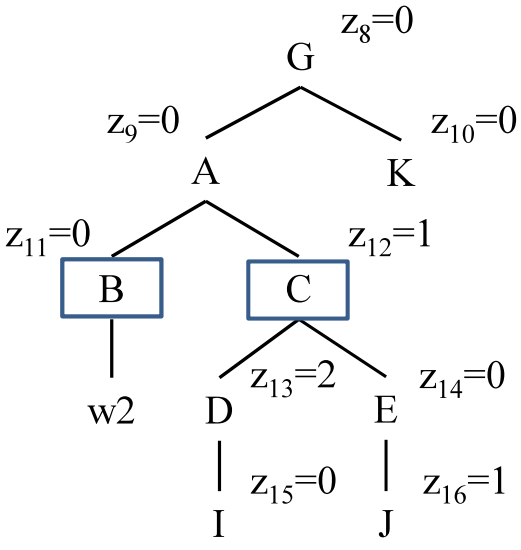
In this section, we propose a *pseudo* blocked subtree sampling for statistical grammar induction. Our aim is to provide a blocked sampler which is widely applicable to many probabilistic grammars and also able to simultaneously samples variables as many as possible. For this reason, we construct a pseudo blocked sampler and a standard MCMC sampler independently, then run both samplers alternately. Besides, the proposed method automatically determines the appropriate size of blocks and updates all variables in the block simultaneously, instead of giving the blocks manually as in previous work. Therefore, our method is independent from particular probabilistic grammars and able to handle variables each of which has multiple value.

5.3.1 Pseudo Blocked Subtree Sampling

Our proposed method constructs a block of common subtrees across parse trees in a training corpus. It then samples from their joint distribution conditioned on all other variables.



(a)



(b)

Figure 5.4: Example parse trees and the block $B_s = \{\{z_2, z_3\}, \{z_{11}, z_{12}\}\}$.

Let $\mathbf{z} = \{z\}$ be a set of latent variables and $tree\,(\mathbf{z})$ be the subtree that $\mathbf{z}$ indicates. For example, suppose $\mathbf{z} = \{z_1 = 0, z_2 = 0, z_3 = 1\}$ in Figure 5.3, then $tree\,(\mathbf{z})$ = (NP-0 (DT-0 NP-1)). We define a block B_s as follows.

$$B_s \equiv \{internal\,(\mathbf{z})\,|tree\,(\mathbf{z}) = s \wedge \cap\mathbf{z} = \varnothing\}\,, \tag{5.1}$$

where $internal\,(\mathbf{z})$ is a set of all variables in $tree\,(\mathbf{z})$ except the root and leaf nonterminal nodes. The block B_s is constructed based on the value of z for each sampling iteration. Figure 5.4 shows two example parse trees and the block of variables in the example parse trees. In Figure 5.4, the block $B_s = \{\{z_2, z_3\}, \{z_{11}, z_{12}\}\}$ corresponds to the common subtrees: s = (A-0 (B-0 (C-1 (D-2 E-0))))). A-0, D-2, E-0 are root or leaf nonterminal nodes in the subtree s. We exclude those nodes from the block since changing their values affects the subtree of surrounding nodes, e.g. (G-1 (A-0 K-0)). B-0 is a leaf node of s, however, its child node has a terminal symbol (lexical word). Thus, it is included in the block since changing the value of B-0 does not affect the surrounding subtrees.

After constructing blocks, our method samples all variables in the block according to the joint probability conditioned on all other variables as follows.

$$P\left(\{\mathbf{z}\}_{\mathbf{z}\in B_s}\,\middle|\mathbf{z}^-, \Theta\right), \tag{5.2}$$

where $\{\mathbf{z}\}_{\mathbf{z}\in B_s}$ denotes a set of variables in B_s, and $\mathbf{z}^-$ denotes all variables in the model except the variables in $\{\mathbf{z}\}_{\mathbf{z}\in B_s}$.

Let c be the number of possible values z takes. The calculation cost of eq. 5.2 is $O\left(c^{|\mathbf{z}|\times|B_s|}\right)$, which becomes computationally infeasible in practice as the size of blocks, $|B_s|$, increases.

We tackle this problem by restrict corresponding variables to be the same values before and after sampling (pseudo blocked sampling). Taking Figure 5.4 as an example, suppose every z is a binary variable, our pseudo sampler considers only the following four possibilities:

$$(z_2, z_3, z_{11}, z_{12}) = (0,0,0,0)\,, (0,1,0,1)\,, (1,0,1,0)\,, (1,1,1,1)$$

rather than total sixteen possibilities:

$$(z_2, z_3, z_{11}, z_{12}) = (0,0,0,0)\,, (0,0,0,1)\,, \ldots (1,1,1,0)\,, (1,1,1,1)\,.$$

Thanks to this restriction, the calculation cost of our pseudo sampling reduces to $O\left(c^{|\mathbf{z}|}\right)$, which is independent from the block size $|B_s|$. However, the sampler does not construct an exact Markov chain that has the posterior distribution as its equilibrium distribution. Therefore, in this work, we run a standard MCMC sampler such as Gibbs sampler and our pseudo blocked subtree sampler alternately. The method of constructing blocks and the algorithm of pseudo blocked subtree sampling are presented in the following sections.

5.3.2 Block Construction

Our method needs to search common subtrees with latent variables in each sampling iteration to construct the blocks $\mathbf{B} = \{B_s\}$. To archive this, we take a pattern mining approach to enumerate common subtrees in a training corpus. The main procedure is listed as follows.

1. Initialize the candidate subtree as a minimal tree, that is, a tree that has only single node.
2. Expand the candidate subtree by appending a node as a child. Calculate the frequency of every candidate subtree.
3. If the frequency becomes one or the number of nodes reaches the user-defined maximum, the algorithm terminates. Else go back to 2.

This algorithm is essentially the same as the frequent tree pattern mining algorithm such as FREQT [1]. After the method finds a set of common subtree patterns, it randomly picks a subtree s, then construct the block B_s until every variable in the corpus belongs to any blocks.

5.3.3 Proposed Algorithm

As described above, our proposed method runs a standard MCMC sampler and the pseudo blocked subtree sampler alternately. The procedure of our method is shown in algorithm 4. The input of algorithm 4 is the number of iterations: I, parse trees: $\{t\}$, and the frequency of our pseudo blocked subtree sampling: f. We will discuss the frequency f later.

The algorithm starts with the standard Gibbs sampling (line 4). Other MCMC samplers can be used instead of Gibbs sampler. Then, it runs the pseudo blocked subtree sampling when the current iteration i satisfies the user-defined condition regarding the frequency f. For example, suppose $f = 10$, the sampler runs the pseudo blocked sampling once every ten times of Gibbs sampling. We introduce the frequency f to enable us to adjust the trade-off between calculation cost and search efficiency. The effect of the frequency f is evaluated in our experiment.

The pseudo blocked subtree sampler firstly finds common subtree patterns by pattern mining approach (line 9). Then, it constructs blocks $\mathbf{B}$. The variable Z is a storage of already assigned variable z.

The procedure is that it randomly picks a subtree s, then construct B_s sequentially (line 15). Then, every variables in the block are removed from Z (line 17). Z becomes empty set when every variable z belongs to any blocks, and the block construction is completed.

The pseudo blocked subtree sampler secondly updates the value of variables. Specifically, pick a block B_s randomly from $\mathbf{B}$, then calculate the probabilities of candidate values according to eq. 5.2 and generate samples (line 21). Finally, after the sampling procedures, recover SR-CFG rules $\hat{\mathbf{e}}$ from the estimated values $\hat{\mathbf{z}}$ (line 26).

5.4 Experiment

5.4.1 Setting

We conducted experiments on the Wall Street Journal (WSJ) portion of the English Penn Treebank data set [30]. We prepared two types of data set: Penn-A (sections 2 for training, 1989 sentences) and Penn-B (section 2-11 for training, 18581 sentences) to evaluate the effect of the data size. The treebank data is right-binarized [31] to construct grammars with only unary and binary productions. We replace lexical words with count ≤ 1 in the training data with the special word: "UNKNOWN". We also split off all the function tags and eliminated empty nodes from the data set, following [22].

We used the SR-CFG model based on Pitman-Yor process described in the section 5.2. We initialized the latent variables randomly and ran our algorithm

for 5000 iterations. In all our experiments, we conducted five independent runs to train our model and evaluated the likelihood of our model.

5.4.2 Results and Discussion

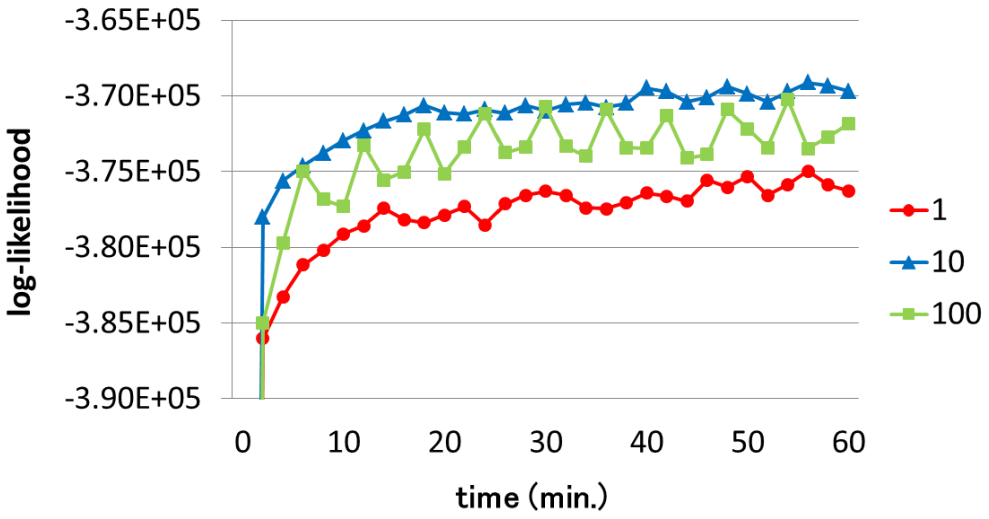
5.4.2.1 Comparison of the Frequency of Pseudo Sampling

We evaluated the effect of our pseudo sampling frequency f on the search efficiency. In this experiment, the number of subcategories is set to be 2, and every latent variable is initialized to be 0 or 1 randomly. Figure 5.5 shows the comparison of log-likelihood of our model with various values of f on Penn-A and Penn-B data set. Since the number of sampling iterations depends on the frequency, the vertical axis indicates running time (minutes) rather than the number of sampling iterations.

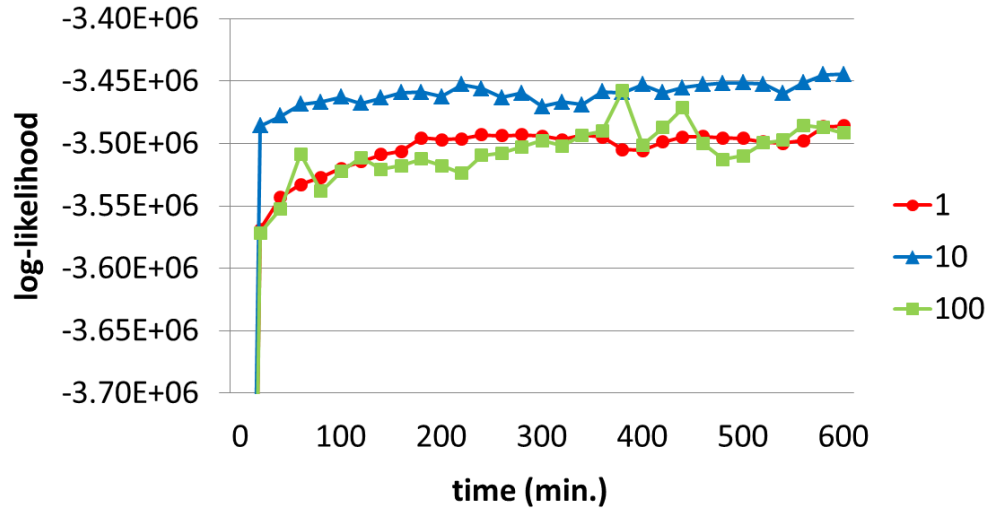
As shown in Figure 5.5, search efficiency, that is, running time of arriving high-likelihood, differs among the frequencies. In particular, running the pseudo blocked sampling for every 10 or 100 Gibbs iterations was better than every 1 Gibbs iteration. This is because the calculation cost of our pseudo blocked sampling is much higher than that of Gibbs sampling. Compared the results on Penn-A data set with that on Penn-B data set, the setting of $f = 10$ performs the best on both data set. However, the result of $f = 100$ is better than that of $f = 1$ on Penn-A data set, which contrasts with the result on Penn-B data set. We speculate that the performance gap is because the pseudo blocked sampling becomes relatively effective for finding high-likelihood values as the data size increases.

5.4.2.2 Comparison of Other Methods

We compared our pseudo blocked sampler with conventional Gibbs sampler and blocked Gibbs sampler. The blocked Gibbs sampler we used is a method of jointly sampling variables of parent and child nodes in a parse tree that forms a SR-CFG rule, instead of sampling one variable at a time. For example, in Figure 5.3, the blocked Gibbs sampler constructs three blocks: $B_1 = \{z_1, z_2, z_3\}, B_2 = \{z_4\}, B_3 = \{z_5\}$ and samples the values of blocked variables from the joint probability. Figure 5.6 shows the comparison of our method with conventional methods. In this experiment, the number of subcategories is set to

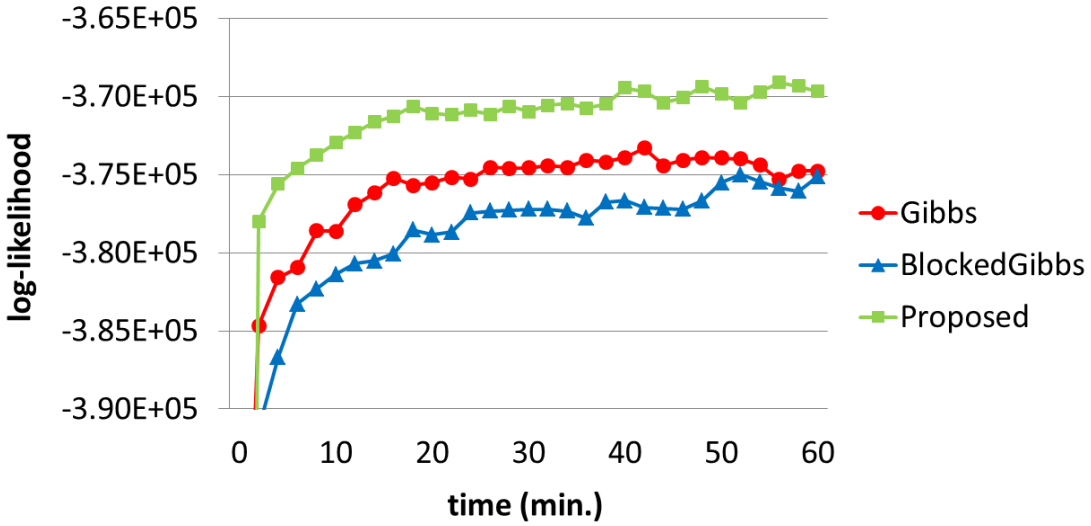


(a) Result of Penn-A data set.

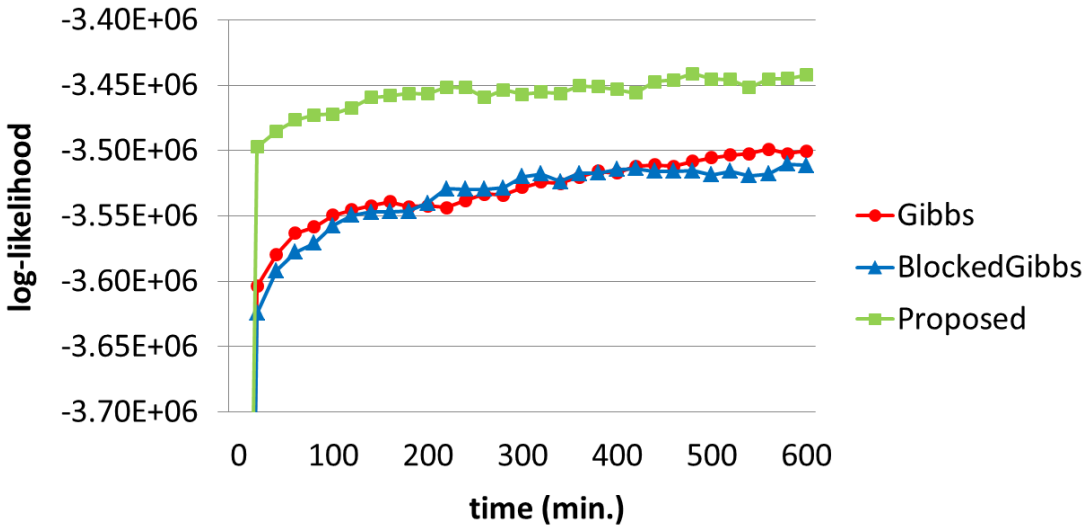


(b) Result of Penn-B data set.

Figure 5.5: Comparison of log-likelihood of our model among various values of f.

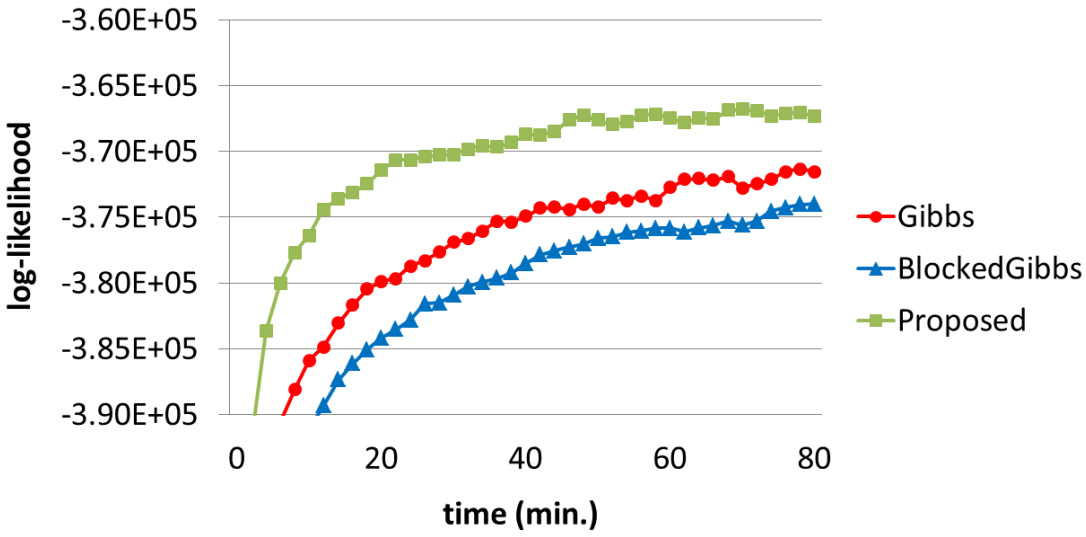


(a) Result of Penn-A data set.

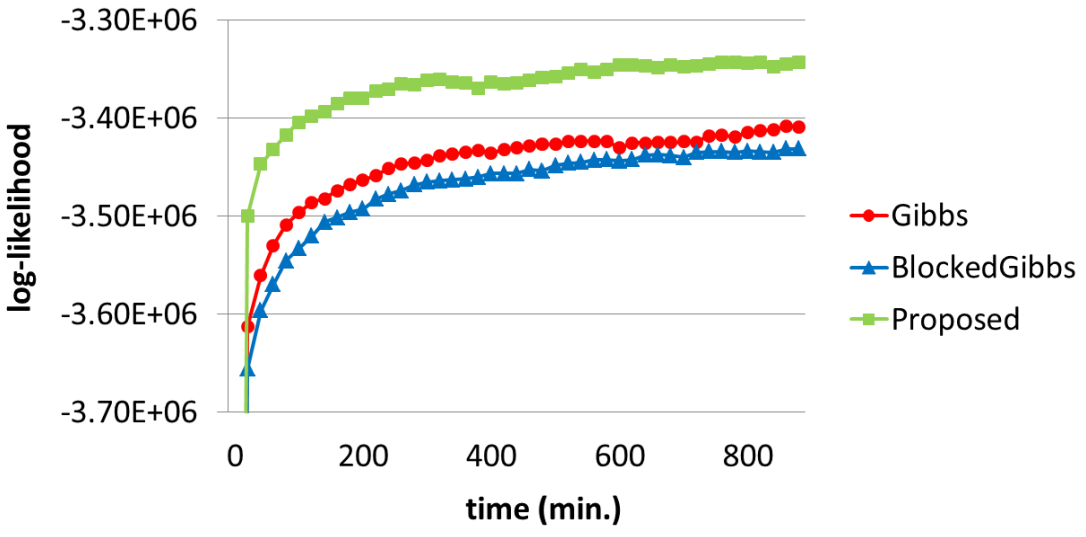


(b) Result of Penn-B data set.

Figure 5.6: Comparison of the pseudo blocked sampler with other methods. The number of subcategories is 2.



(a)



(b)

Figure 5.7: Comparison of the pseudo blocked sampler with other methods. The number of subcategories is 2.

be 2, and every latent variable is initialized to be 0 or 1 randomly. Following the previous experiment, the frequency f of our sampler is set to be 10.

As shown in Figure 5.6, our proposed method performs the best on both Penn-A and Penn-B data set. For Penn-A data set, the blocked Gibbs sampling took longer time to achieve high-likelihood than standard Gibbs sampling despite that the blocked Gibbs sampling updated several variables at a time. This result indicates that iterating low-cost sampler many times may be more efficient than calculating joint probability for blocked sampler on small data set. However, in Figure 5.6a, the difference between the log-likelihood of standard Gibbs sampling and that of blocked Gibbs sampling gradually decreases and then becomes almost equivalent at approximately 60 minutes of running time. This suggests that the standard Gibbs sampler gets stuck when it passes about 30 minutes, while the blocked Gibbs sampler is not likely to get stuck because of updating several variables in a parse tree at a time. On the other hand, our pseudo sampler jointly updates many variables across parse trees, thus performs better than standard and blocked Gibbs samplers.

For Penn-B data set, standard Gibbs sampling and blocked Gibbs sampling performs almost the same. Compared with Penn-A data set, the blocked sampling gets an advantage over standard Gibbs sampling since standard Gibbs sampling becomes likely to get stuck into a local optimum as the data size increases. Our pseudo blocked sampling is confirmed to perform extremely well even the data size increases.

We next set the number of subcategories to be 3 and compared our method with conventional methods. Figure 5.7 shows the comparison of our method with conventional methods. Following the previous experiments, every latent variable is initialized to be 0 , 1 and 2 randomly. As shown in Figure 5.7, our method performs the best on both Penn-A and Penn-B data set. Compared with the setting of 2 subcategories, the blocked Gibbs sampling underperforms standard Gibbs sampling. The reason is that the cost of computing the joint probability increases exponentially as the number of subcategories increases. In contrast, our method reduces the calculation cost by performing the blocked sampling with an appropriate frequency, thus perform efficiently even the number of subcategories increases.

5.5 Related Work

Type-based sampling [28] is an approach for improving search efficiency of Gibbs sampler by sampling the same type of variables at a time. If variables and those surrounding variables (parent and child nodes) have the same values, they are called as the same *type*. It has been shown that type-based sampling performs more efficient than Gibbs sampling, however, it is assumed that the probability model is based on Dirichlet process prior and every variables in the model is binary. Therefore, the type-based sampling is unable to be applied to our Pitman-Yor based SR-CFG model.

Table label resampling [23] is a blocked subtree sampling for an inference of adaptor grammars [21]. It is similar to our method in that table label resampling and standard MCMC sampling is constructed independently and runs both samplers alternately. However, adaptor grammar is restricted that all leaf nodes of the rules in adaptor grammars must be terminal symbols. Furthermore, adaptor grammars have largely been applied to the task of unsupervised structural induction from raw texts. Therefore, table label resampling is a blocked sampling method for sequential data rather than tree-structured data.

5.6 Summary

We proposed a pseudo blocked subtree sampling to improve search efficiency of statistical grammar induction. Our method differs from standard blocked samplers in that we construct a blocked sampler and a standard MCMC sampler independently, then run both samplers alternately. Besides, the proposed method automatically determines the appropriate size of blocks and updates all variables in the block simultaneously, instead of giving the blocks manually as in previous work. Therefore, our method is independent from particular grammatical models and able to handle variables each of which has multiple value. Our method generates samples from an approximate posterior distribution over grammatical rules, however, the experimental results show that our method obtained better results than standard Gibbs sampling and blocked Gibbs sampling regardless of the amount of data.

Algorithm 4: Pseudo blocked subtree sampling

Input : number of iterations: I, parse trees: $\{t\}$, frequency of pseudo blocked subtree sampling: f

Output: estimated elementary trees: $\hat{\mathbf{e}}$, estimated parameters: $\hat{\Theta}$

for $i = 1, \ldots, I$ **do**
  Initialize $\mathbf{z}, \Theta$
  `// Gibbs sampling`
  foreach z *in random order* **do**
    Generate z' according to $P\left(z \,|\mathbf{z} \setminus z, \Theta\right)$
    $z \leftarrow z'$
  end
  Update parameters Θ
  if i *mod* $f = 0$ **then**
    `// Construct block`
    Find subtree patterns S by subtree expansion method
    $Z \leftarrow \mathbf{z}$
    $\mathbf{B} \leftarrow \emptyset$
    while $Z \neq \emptyset$ **do**
      Pick subtree $s \in S$ at random
      Construct B_s
      $\mathbf{B} \leftarrow \mathbf{B} \cup B_s$
      foreach z *in* B_s **do**
        $Z \leftarrow Z \setminus z$
      end
    end
    `// Pseudo Blocked subtree sampling`
    foreach $B_s \in \mathbf{B}$ *in random order* **do**
      Generate $\{\mathbf{z}\}'$ according to $P\left(\{\mathbf{z}\}_{\mathbf{z} \in B_s} \,|\mathbf{z}^{-}, \Theta\right)$
      $\{\mathbf{z}\} \leftarrow \{\mathbf{z}\}'$
    end
  end
end
Recover $\hat{\mathbf{e}}$ from $\hat{\mathbf{z}}$

Chapter 6

Conclusion

This chapter summarizes the thesis and gives further directions we intend to explore.

6.1 Summary

This thesis aimed to construct an accurate parser based on the statistical modeling of rich, powerful, and linguistically-motivated tree-generating grammars. We argued two major problems of conventional TSG models for syntactic parsing and proposed more expressive grammars to overcome the problems.

In Chapter 3, we tackled the problem of optional-obligatory distinction by proposing to incorporate an insertion operator into the conventional TSG model, i.e., Tree Insertion Grammar (TIG). We developed a probabilistic model that automatically induces TIG from a treebank, instead of using heuristic extraction rules. We also developed grammar decomposition rules to transform TIG derivation into equivalent CFG for efficient training. For a small dataset, our TIG model outperformed CFG and TSG. For a large dataset, our model achieved comparable parsing results to the TSG model, making the number of grammars much smaller than TSG.

In Chapter 4, we presented an SR-TSG, which is an extension of the conventional TSG model where each symbol of tree fragments can be automatically subcategorized to address the problem of the conditional independence assumptions of a TSG. We proposed a novel backoff modeling of an SR-TSG based on the hierarchical Pitman-Yor Process and sentence-level and tree-level blocked MCMC

sampling for training our model. Our best model significantly outperformed the conventional TSG and achieved state-of-the-art result in a WSJ parsing task.

In Chapter 5, we proposed a pseudo blocked subtree sampling to improve search efficiency of statistical grammar induction. Our method differs from standard blocked samplers in that we construct a blocked sampler and a standard MCMC sampler independently, then run both samplers alternately. Besides, the proposed method automatically determines the appropriate size of blocks and updates all variables in the block simultaneously, instead of giving the blocks manually as in previous work. Therefore, our method is independent from particular grammatical models and able to handle variables each of which has multiple value. Our method generates samples from an approximate posterior distribution over grammatical rules, however, the experimental results show that our method obtained better results than standard Gibbs sampling and blocked Gibbs sampling regardless of the amount of data.

6.2 Future Directions

There are several open problems we intend to explore.

First, this thesis focused primarily on modeling and learning of probabilistic grammars. Decoding is an another part to be considered for making the parser practical. Rich but complex grammars contain a large number of grammar rules and thus the parsing (decoding) time becomes seriously slow. Therefore, efficient decoding algorithms of the grammars are also important for practical use. The pruning algorithms such as coarse-to-fine decoding [36] is one possibility to be explored for fast decoding when the grammar rules are large.

Second, our grammars require a large amount of human-annotated resources of syntax trees. However, a large amount of human-annotated resources are not readily available to many languages since human-annotation is costly and time-consuming. Therefore, we will explore semi-supervised and unsupervised approaches for statistical grammar induction. Both approaches utilize unannotated data for an inference of probabilistic grammars. One issue is that the search space over parameters is large since there is no correct tree structure in the training data. Efficient learning algorithms such as parallelization might be required.

Third, we will apply our parser to the NLP systems such as machine translation

and summarization to validate the efficiency of our statistical parsers. Besides, we will explore to apply our grammars and parsing algorithms to other fields such as Bioinformatics and Music Information Retrieval.

Bibliography

[1] Kenji Abe, Shinji Kawasoe, Tatsuya Asai, Hiroki Arimura, and Setsuo Arikawa. Optimized substructure discovery for semi-structured data. In *Proceedings of 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 1–14, 2002.

[2] Mohit Bansal and Dan Klein. Simple, accurate parsing with an all-fragments grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1098–1107, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[3] Phil Blunsom and Trevor Cohn. Unsupervised induction of tree substitution grammars for dependency parsing. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1204–1213, 2010.

[4] Xavier Carreras and Michael Collins. Non-projective parsing for statistical machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 200–209, Singapore, August 2009. Association for Computational Linguistics.

[5] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the First Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139, Seattle, Washington, April 29 - May 04 2000. Morgan Kaufmann Publishers, San Francisco, CA, USA.

[6] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL)*, 1:173–180, 2005.

[7] J. Chen, S. Bangalore, and K. Vijay-Shanker. Automated extraction of tree-adjoining grammars from treebanks. *Natural Language Engineering*, 12(03):251–299, 2006.

[8] David Chiang. *Statistical Parsing with an Automatically Extracted Tree Adjoining Grammar*, chapter 16, pages 299–316. CSLI Publications, 2003.

[9] Shay B Cohen, David M Blei, and Noah A Smith. Variational inference for adaptor grammars. In *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 564–572, 2010.

[10] 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. Association for Computational Linguistics.

[11] 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 (HLT-NAACL)*, pages 548–556, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[12] Michael Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637, 2003.

[13] Steve DeNeefe and Kevin Knight. Synchronous tree adjoining machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 727–736, Singapore, August 2009. Association for Computational Linguistics.

[14] Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *Annals of Statistics*, 1:209–230, 1973.

[15] Victoria Fossum and Kevin Knight. Combining constituent parsers. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers (HLT-NAACL)*, pages 253–256, 2009.

[16] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, volume 4, pages 273–280. Boston, 2004.

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

[18] W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[19] Liang Huang. Forest reranking : Discriminative parsing with non-local features. pages 586–594, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[20] Hemant Ishwaran and Lancelot F James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173, 2001.

[21] M. Johnson, T.L. Griffiths, and S. Goldwater. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. *Advances in Neural Information Processing Systems (NIPS)*, 19:641, 2007.

[22] Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

[23] 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 (HLT-NAACL)*, pages 317–325, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[24] 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; Proceedings of the Main Conference (HLT-NAACL), pages 139–146, Rochester, New York, April 2007. Association for Computational Linguistics.

[25] A.K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions. *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, pages 206–250, 1985.

[26] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL)*, 1:423–430, 2003.

[27] K. Lari and S.J. Young. Applications of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, 5(3):237–257, 1991.

[28] 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 (NAACL)*, pages 573–581, Los Angeles, California, June 2010. Association for Computational Linguistics.

[29] Yudong Liu and Anoop Sarkar. Experimental evaluation of LTAG-based features for semantic role labeling. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 590–599, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[30] Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19:313–330, 1993.

[31] T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL)*, pages 75–82. Association for Computational Linguistics, 2005.

[32] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[33] Radford M Neal. Slice sampling. *Annals of statistics*, pages 705–741, 2003.

[34] Slav Petrov. Products of random latent variable grammars. *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 19–27, 2010.

[35] 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 the 44th Annual Meeting of the Association for Computational Linguistics (ICCL-ACL)*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics.

[36] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 404–411. Association for Computational Linguistics, April 2007.

[37] J. Pitman and M. Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2):855–900, 1997.

[38] 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. Association for Computational Linguistics.

[39] Y. Schabes and R.C. Waters. Tree insertion grammar: a cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Fuzzy Sets and Systems*, 76(3):309–317, 1995.

[40] Jayaram Sethuraman. A constructive definition of dirichlet priors. *Statistica Sinica*, 4:639–650, 1994.

[41] Hiroyuki Shindo, Akinori Fujino, and Masaaki Nagata. Insertion operator for bayesian tree substitution grammars. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*, pages 206–211, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[42] Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 440–448, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[43] Y. W. Teh. A Bayesian interpretation of interpolated Kneser-Ney. Technical Report TRA2/06, School of Computing, National University of Singapore, 2006.

[44] Y. W. Teh. A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ICCL-ACL)*, pages 985–992, 2006.

[45] J Tenenbaum, TJ O'Donnell, and ND Goodman. Fragment grammars: Exploring computation and reuse in language. *MIT Computer Science and Artificial Intelligence Laboratory Technical Report Series*, 2009.

[46] Mengqiu Wang, Noah A Smith, and Teruko Mitamura. What is the jeopardy model ? a quasi-synchronous grammar for QA. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, 2007.

[47] Fei Xia. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS)*, pages 398–403, 1999.

[48] Elif Yamangil and Stuart Shieber. Estimating compact yet rich tree insertion grammars. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 110–114, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[49] Elif Yamangil and Stuart M Shieber. Bayesian synchronous tree-substitution grammar induction and its application to sentence compression. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 937–947, 2010.

[50] Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. K-best combination of syntactic parsers. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1552–1560, 2009.

[51] 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 (EMNLP-CoNLL)*, pages 551–560, 2007.

List of Publications

Journal Papers

1. Hiroyuki Shindo, Yuji Matsumoto and Masaaki Nagata. “Blocked Subtree Sampler for Statistical Grammar Induction”. IPSJ Transactions on Mathematical Modeling and Its Applications (TOM), to appear. (in Japanese)

2. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. “A Probabilistic Model of Tree Insertion Grammars Based on Pitman-Yor Processes”. IPSJ Transactions on Mathematical Modeling and Its Applications (TOM), Vol. 5, No. 3, pp. 95-106, 2012. (in Japanese)

Conference Papers

1. Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino and Masaaki Nagata. “Statistical Parsing with Symbol-Refined Tree Substitution Grammars”. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 3082-3086, August 2013.

2. Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino and Masaaki Nagata. “Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing”. In Proceedings of the Association for Computational Linguistics (ACL), pp. 440-448, July 2012. (Best Paper Award)

3. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. “Insertion Operator for Bayesian Tree Substitution Grammars”. In Proceedings of the Association for Computational Linguistics (ACL), pp. 206-211, June 2011.

Awards

1. Computer Science Research Award for Young Scientists, Information Processing Society of Japan, "A Probabilistic Model of Tree Insertion Grammars Based on Pitman-Yor Processes".

2. Best Paper Award, the 50th Annual Meeting of the Association for Computational Linguistics, "Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing".

3. Best Paper Award, the 18th Annual Meeting of the Association for Natural Language Processing (domestic conference), "Symbol-Refined Tree Substitution Grammars based on Pitman-Yor Processes and its Application to Syntactic Parsing".

Other Publications

1. Hiroyuki Shindo, Yuji Matsumoto and Masaaki Nagata. "Blocked Subtree Sampler for Statistical Grammar Induction". IPSJ SIG Technical Report, MPS-93, No. 6, pp. 1-6, 2013. (in Japanese)

2. Hiroyuki Shindo, Tsutomu Hirao, Jun Suzuki, Akinori Fujino and Masaaki Nagata. "TripleEye: Mining Closed Itemsets with Minimum Length Thresholds based on Ordered Inclusion Tree". IPSJ Transactions on Databases (TOD), Vol. 5, No. 3, pp. 15-25, September 2012.

3. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. "A Probabilistic Model of Tree Insertion Grammars Based on Pitman-Yor Processes". IPSJ SIG Technical Report, MPS-88, No. 12, pp. 1-6, 2012. (in Japanese)

4. Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino and Masaaki Nagata. "Symbol-Refined Tree Substitution Grammars based on Pitman-Yor Processes and its Application to Syntactic Parsing". In Proceedings of the 18th Annual Meeting of the Association for Natural Language Processing (NLP), 2012. (in Japanese)

5. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. “Word Alignment with Synonym Information”. IPSJ Transactions on Mathematical Modeling and Its Applications (TOM), Vol. 4, No. 2, pp. 13-22, 2011. (in Japanese)

6. Sanae Fujita, Kevin Duh, Akinori Fujino, Hirotoshi Taira and Hiroyuki Shindo. “Effectiveness of Automatic Expansion of Training Data for Japanese Word Sense Disambiguation”. Journal of natural language processing Vol. 18, No. 3, pp. 273-291, 2011. (in Japanese)

7. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. “Bayesian Induction of Tree Adjoining Grammars”. In Proceedings of the 17th Annual Meeting of the Association for Natural Language Processing (NLP), 2011. (in Japanese)

8. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. “Word Alignment with Synonym Regularization”. In Proceedings of the Association for Computational Linguistics (ACL), pp. 137-141, July 2010.

9. Hiroyuki Shindo, Akinori Fujino and Masaaki Nagata. “Word Alignment with Synonym Information”. IPSJ SIG Technical Report, MPS-80, No. 9, pp. 1-6, 2010. (in Japanese)