Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing

Hiroyuki Shindo[†] **Yusuke Miyao**[‡] **Akinori Fujino**[†] **Masaaki Nagata**[†]

[†]NTT Communication Science Laboratories, NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto, Japan
{shindo.hiroyuki,fujino.akinori,nagata.masaaki}@lab.ntt.co.jp

[‡]National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
yusuke@nii.ac.jp

Abstract

We propose Symbol-Refined Tree Substitution Grammars (SR-TSGs) for syntactic parsing. An SR-TSG is an extension of the conventional TSG model where each nonterminal symbol can be refined (subcategorized) to fit the training data. We aim to provide a unified model where TSG rules and symbol refinement are learned from training data in a fully automatic and consistent fashion. We present a novel probabilistic SR-TSG model based on the hierarchical Pitman-Yor Process to encode backoff smoothing from a fine-grained SR-TSG to simpler CFG rules, and develop an efficient training method based on Markov Chain Monte Carlo (MCMC) sampling. Our SR-TSG parser achieves an F1 score of 92.4% in the Wall Street Journal (WSJ) English Penn Treebank parsing task, which is a 7.7 point improvement over a conventional Bayesian TSG parser, and better than state-of-the-art discriminative reranking parsers.

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 (Galley et al., 2004; DeNeefe and Knight, 2009), sentence compression (Cohn and Lapata, 2009; Yamangil and Shieber, 2010), and question answering (Wang et al., 2007). 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 freedom assumptions (Klein and Manning, 2003).

In recent years, there has been an increasing interest in tree substitution grammar (TSG) as an alternative to CFG for modeling syntax trees (Post and Gildea, 2009; Tenenbaum et al., 2009; Cohn et al., 2010). 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 (Cohn et al., 2010). Previous work on TSG parsing (Cohn et al., 2010; Post and Gildea, 2009; Bansal and Klein, 2010) 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 (Petrov et al., 2006) and the Charniak parser (Charniak and Johnson, 2005)). One major drawback of TSG is that the context freedom 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 (Johnson, 1998; Collins, 2003; Matsuzaki et al., 2005). Symbol refinement is a successful approach for weakening context freedom 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 (Zuidema, 2007; Bansal and Klein, 2010), large

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 440–448,
Jeju, Republic of Korea, 8-14 July 2012.

tree fragments and symbol refinement work complementarily for syntactic parsing. For example, Bansal and Klein (2010) 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 (Pitman and Yor, 1997), 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.

2 Background and Related Work

Our SR-TSG work is built upon recent work on Bayesian TSG induction from parse trees (Post and Gildea, 2009; Cohn et al., 2010). We firstly review the Bayesian TSG model used in that work, and then present related work on TSGs and symbol refinement.

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 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 1a shows an example parse tree and Figure 1b shows its example TSG derivation. Since different derivations may produce the same parse tree, recent work on TSG induction (Post and Gildea, 2009; Cohn et al., 2010) 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 (Ferguson, 1973) or the Pitman-Yor Process to encourage sparse and compact grammars.

Several studies have combined TSG induction and symbol refinement. An adaptor grammar (Johnson et al., 2007a) 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

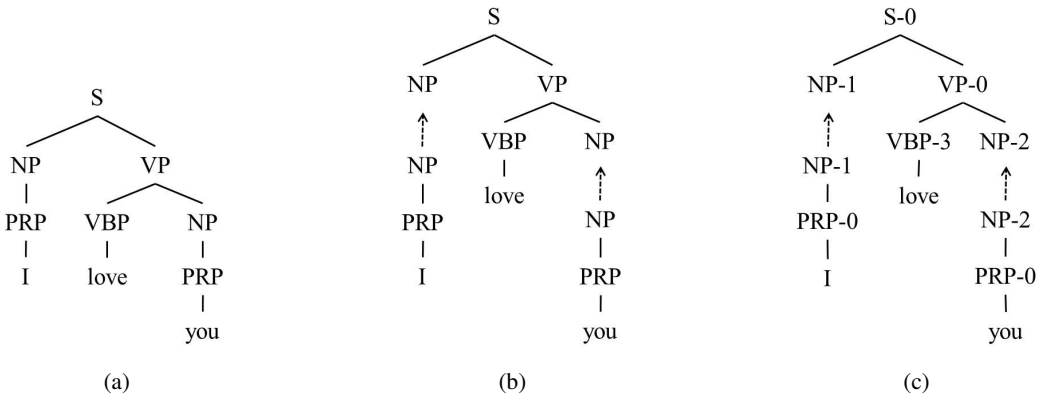

Figure 1: (a) Example parse tree. (b) Example TSG derivation of (a). (c) Example SR-TSG derivation of (a). The refinement annotation is hyphenated with a nonterminal symbol.

morphology analysis, word segmentation (Johnson and Goldwater, 2009), and dependency grammar induction (Cohen et al., 2010), rather than constituent syntax parsing.

An all-fragments grammar (Bansal and Klein, 2010) 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 (Klein and Manning, 2003) 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.

3 Symbol-Refined Tree Substitution Grammars

In this section, we propose Symbol-Refined Tree Substitution Grammars (SR-TSGs) for syntactic parsing. 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 1c shows an example of SR-TSG derivation. As with previous 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.

3.1 Probabilistic Model

We define a probabilistic model of an SR-TSG based on the Pitman-Yor Process (PYP) (Pitman and Yor, 1997), 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 (Teh, 2006b), and TSG induction (Cohn et al., 2010). 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 a hierarchical PYP to encode a backoff scheme from the SR-TSG rules to simpler CFG rules, inspired by recent work on dependency parsing (Blunsom and Cohn, 2010).

Our model consists of a three-level hierarchy. Table 1 shows an example of the SR-TSG rule and its backoff tree fragments as an illustration of this three-level hierarchy. The topmost level of our model is a distribution over the SR-TSG rules as follows.

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

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: $(d_{x_k}, \theta_{x_k}, P^{\text{sr-tsg}})$. $P^{\text{sr-tsg}}\left(\cdot\,|x_k\right)$

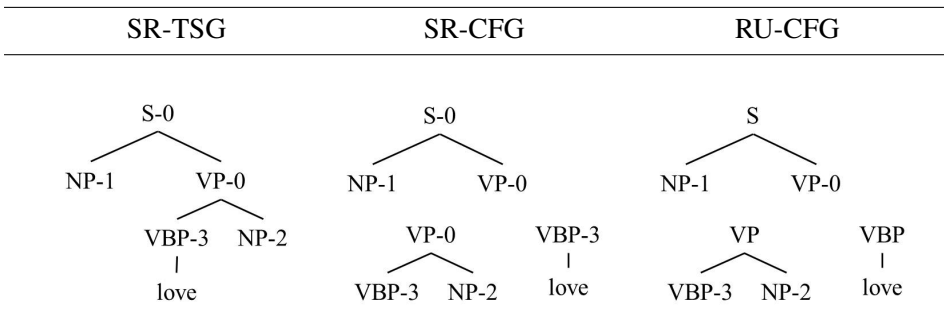

Table 1: Example three-level backoff.

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

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

where $F(e)$ is a set of frontier nonterminal nodes and $I(e)$ 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 1. The function cfg-rules$\,(e\,|x_k)$ returns the SR-CFG rules that e contains, which take the form of $x_k \rightarrow \alpha$. 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 $x \rightarrow \alpha$. 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 1. 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.

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

where $|x \rightarrow \cdot|$ 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 (Blunsom and Cohn, 2010; Cohen et al., 2010), 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 Section 5.

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, $p\left(\mathbf{e}\,|\mathbf{t}, \mathbf{d}, \boldsymbol{\theta}, \mathbf{s}\right)$.

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.

4.1 Inference of Symbol Subcategories

For the inference of latent symbol subcategories, we adopt split and merge training (Petrov et al., 2006) 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 Metroporil-Hastings (MH) sampler and the *tree-level* blocked Gibbs sampler, while (Petrov et al., 2006) 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 (Johnson et al., 2007b; Cohn et al., 2010). 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 (Lari and Young, 1991) in a bottom-up manner, 2) sample a derivation tree in a top-down manner, and 3) accept or reject the derivation sample by using the MH test. See (Cohn et al., 2010) 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 (Johnson and Goldwater, 2009), 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.

4.2 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 (Cohn and Lapata, 2009; Post and Gildea, 2009). 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 (Cohn et al., 2010) 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.

4.3 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: $d \sim \mathrm{Beta}\,(1,1)$, $\theta \sim \mathrm{Gamma}\,(1,1)$. The values of d and θ are optimized with the auxiliary variable technique (Teh, 2006a).

5 Experiment

5.1 Settings

5.1.1 Data Preparation

We ran experiments on the Wall Street Journal (WSJ) portion of the English Penn Treebank data set (Marcus et al., 1993), 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 (Matsuzaki et al., 2005) 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 (Petrov et al., 2006). We also split off all the function tags and eliminated empty nodes from the data set, following (Johnson, 1998).

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 (Petrov et al., 2006) 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 2: Comparison of parsing accuracy with the small and full training sets. *Our reimplementation of (Cohn et al., 2010).

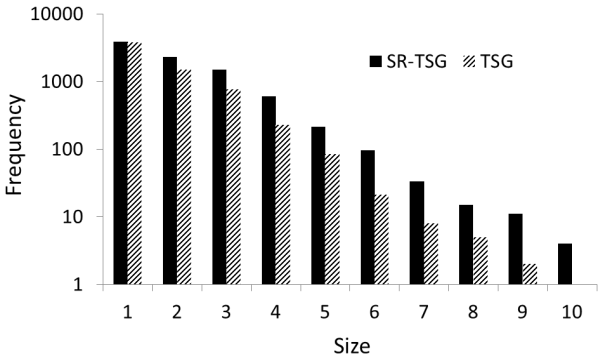

Figure 2: Histogram of SR-TSG and TSG rule sizes on the small training set. The size is defined as the number of CFG rules that the elementary tree contains.

5.2 Results and Discussion

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 2 shows the F1 scores of the CFG, TSG and SR-TSG parsers for small and full training sets. In Table 2, 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

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

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 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 2 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. In Figure 2, we can see that 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.

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 3 shows the F1 scores of an SR-TSG and conventional parsers with the full training set. In Table 3, 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 (Petrov, 2010).

Our SR-TSG (single) parser achieved an F1 score of 91.1%, which is a 6.4 point improvement over the conventional Bayesian TSG parser reported by (Cohn et al., 2010). 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 (Petrov et al., 2006), 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 discriminative reranking parsers such as (Charniak and Johnson, 2005) and (Huang, 2008) 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, (Petrov, 2010) 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 (Fossum and Knight, 2009; Zhang et al., 2009) have proposed different approaches that involve combining k-best lists of candidate trees. We will deal with those methods in future work.

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 $\rightarrow$ NP^{NNP} VP " with probability p and "NP^{NNP} $\rightarrow$ NNP" with probability 1. From this viewpoint, TSG utilizes surrounding symbols (NNP of NP^{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 comfirm that jointly modeling both latent variables using our SR-TSG assists accurate parsing.

6 Conclusion

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.

Acknowledgements

We would like to thank Liang Huang for helpful comments and the three anonymous reviewers for thoughtful suggestions. We would also like to thank Slav Petrov and Hui Zhang for answering our questions about their parsers.

References

Mohit Bansal and Dan Klein. 2010. Simple, Accurate Parsing with an All-Fragments Grammar. In *In Proc. of ACL*, pages 1098–1107.

Phil Blunsom and Trevor Cohn. 2010. Unsupervised Induction of Tree Substitution Grammars for Dependency Parsing. *In Proc. of EMNLP*, pages 1204–1213.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. *In Proc. of ACL*, 1:173–180.

Shay B Cohen, David M Blei, and Noah A Smith. 2010. Variational Inference for Adaptor Grammars. In *In Proc. of HLT-NAACL*, pages 564–572.

Trevor Cohn and Mirella Lapata. 2009. Sentence Compression as Tree Transduction. *Journal of Artificial Intelligence Research*, 34:637–674.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing Tree-Substitution Grammars. *Journal of Machine Learning Research*, 11:3053–3096.

Michael Collins. 2003. Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, 29:589–637.

Steve DeNeefe and Kevin Knight. 2009. Synchronous Tree Adjoining Machine Translation. *In Proc. of EMNLP*, page 727.

Thomas S Ferguson. 1973. A Bayesian Analysis of Some Nonparametric Problems. *Annals of Statistics*, 1:209–230.

Victoria Fossum and Kevin Knight. 2009. Combining Constituent Parsers. *In Proc. of HLT-NAACL*, pages 253–256.

Michel Galley, Mark Hopkins, Kevin Knight, Daniel Marcu, Los Angeles, and Marina Del Rey. 2004. What's in a Translation Rule? *Information Sciences*, pages 273–280.

Liang Huang. 2008. Forest Reranking : Discriminative Parsing with Non-Local Features. *In Proc. of ACL*, 19104:0.

Mark Johnson and Sharon Goldwater. 2009. Improving nonparameteric Bayesian inference: experiments on unsupervised word segmentation with adaptor grammars. In *In Proc. of HLT-NAACL*, pages 317–325.

Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. 2007a. Adaptor Grammars : A Framework for Specifying Compositional Nonparametric Bayesian Models. *Advances in Neural Information Processing Systems 19*, 19:641–648.

Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. 2007b. Bayesian Inference for PCFGs via Markov chain Monte Carlo. In *In Proc. of HLT-NAACL*, pages 139–146.

Mark Johnson. 1998. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24:613–632.

Dan Klein and Christopher D Manning. 2003. Accurate Unlexicalized Parsing. *In Proc. of ACL*, 1:423–430.

K Lari and S J Young. 1991. Applications of Stochastic Context-Free Grammars Using the Inside–Outside Algorithm. *Computer Speech and Language*, 5:237–257.

Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. *In Proc. of ACL*, pages 75–82.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. *In Proc. of ACL*, pages 433–440.

Slav Petrov. 2010. Products of Random Latent Variable Grammars. *In Proc. of HLT-NAACL*, pages 19–27.

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

Matt Post and Daniel Gildea. 2009. Bayesian Learning of a Tree Substitution Grammar. In *In Proc. of ACL-IJCNLP*, pages 45–48.

Yee Whye Teh. 2006a. A Bayesian Interpretation of Interpolated Kneser-Ney. *NUS School of Computing Technical Report TRA2/06*.

YW Teh. 2006b. A Hierarchical Bayesian Language Model based on Pitman-Yor Processes. *In Proc. of ACL*, 44:985–992.

J Tenenbaum, TJ O'Donnell, and ND Goodman. 2009. Fragment Grammars: Exploring Computation and Reuse in Language. *MIT Computer Science and Artificial Intelligence Laboratory Technical Report Series*.

Mengqiu Wang, Noah A Smith, and Teruko Mitamura. 2007. What is the Jeopardy Model ? A Quasi-Synchronous Grammar for QA. *In Proc. of EMNLP-CoNLL*, pages 22–32.

Elif Yamangil and Stuart M Shieber. 2010. Bayesian Synchronous Tree-Substitution Grammar Induction and Its Application to Sentence Compression. In *In Proc. of ACL*, pages 937–947.

Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. 2009. K-Best Combination of Syntactic Parsers. *In Proc. of EMNLP*, pages 1552–1560.

Willem Zuidema. 2007. Parsimonious Data-Oriented Parsing. *In Proc. of EMNLP-CoNLL*, pages 551–560.