Transformational Priors Over Grammars

Jason Eisner  <jason@cs.jhu.edu>
Johns Hopkins University, 3400 N. Charles St., NEB 224, Baltimore, MD

Abstract

This paper proposes a novel class of PCFG parameterizations that support linguistically reasonable priors over PCFGs. To estimate the parameters is to discover a notion of relatedness among context-free rules such that related rules tend to have related probabilities. The prior favors grammars in which the relationships are simple to describe and have few major exceptions. A basic version that bases relatedness on weighted edit distance yields superior smoothing of grammars learned from the Penn Treebank (20% reduction of rule perplexity over the best previous method).

1 A Sketch of the Concrete Problem

This paper uses a new kind of statistical model to smooth the probabilities of PCFG rules. It focuses on “flat” or “dependency-style” rules. These resemble subcategorization frames, but include adjuncts as well as arguments. The verb put typically generates 3 dependents—a subject NP at left, and an object NP and goal PP at right:

- $S \rightarrow NP \text{ put } NP PP$: Jim put [the pizza] [in the oven]

But put may also take other dependents, in other rules:

- $S \rightarrow NP \text{ Adv} \text{ put } NP PP$: Jim often put [a pizza] [in the oven]
- $S \rightarrow NP \text{ put } NP PP PP$: Jim put soup [in an oven] [at home]
- $S \rightarrow NP \text{ put } NP$: Jim put [some shares of IBM stock]
- $S \rightarrow NP \text{ put } Prt NP$: Jim put away [the sauce]
- $S \rightarrow TO \text{ put } NP PP$: to put [the pizza] [in the oven]
- $S \rightarrow NP \text{ put } NP PP SBAR$: Jim put it [to me] [that ...]

These other rules arise if put can add, drop, reorder, or retype its dependents. These edit operations on rules are semantically motivated and quite common (Table 1).

We wish to learn contextual probabilities for the edit operations, based on an observed sample of flat rules. In English we should discover, for example, that it is quite common to add or delete PP at the right edge of a rule. These contextual edit probabilities will help us guess the true probabilities of novel or little-observed rules.

However, rules are often idiosyncratic. Our smoothing method should not keep us from noticing (given enough evidence) that put takes a PP more often than most verbs. Hence this paper’s proposal is a Bayesian smoothing method that allows idiosyncrasy in the grammar while presuming regularity to be more likely a priori.

The model will assign a positive probability to each of the infinitely many formally possible rules. The following bizarre rule is not observed in training, and seems very unlikely. But there is no formal reason to rule it out, and it might help us parse an unlikely test sentence. So the model will allow it some tiny probability:

- $S \rightarrow NP \text{ Adv} PP \text{ put } PP PP PP NP \text{ Adv} PP S$

2 Background and Other Approaches

A PCFG is a conditional probability function $p(RHS | LHS)$. For example, $p(V NP PP | VP)$ gives the probability of the rule $VP \rightarrow V NP PP$. With lexicalized non-terminals, it has form $p(V_{put} NP_{pizza} PP_{in} | VP_{put})$.

Usually one makes an independence assumption and defines this as $p(V_{put} NP PP | VP_{put})$ times factors that choose dependent headwords pizza and in according to the selectional preferences of put. This paper is about estimating the first factor, $p(V_{put} NP PP | VP_{put})$.

In supervised learning, it is simplest to use a maximum likelihood estimate (perhaps with backoff from put). Charniak (1997) calls this a “Treebank grammar” and gambles that assigning 0 probability to rules unseen in training data will not hurt parsing accuracy too much.

However, there are four reasons not to use a Treebank grammar. First, ignoring unseen rules necessarily sacrifices some accuracy. Second, we will show that it improves accuracy to flatten the parse trees and use flat, dependency-style rules like $p(NP \text{ put } NP PP | S_{put})$; this avoids overly strong independence assumptions, but it increases the number of unseen rules and so makes Treebank grammars less tenable. Third, backing off from the word is a crude technique that does not distinguish among words. Fourth, one would eventually like to reduce or eliminate supervision, and then generalization is important to constrain the search to reasonable grammars.

To smooth the distribution $p(RHS | LHS)$, one can define it in terms of a set of parameters and then estimate those parameters. Most researchers have used an n-gram model (Eisner, 1996; Charniak, 2000) or more general Markov model (Alshawi, 1996) to model the sequence of nonterminals in the RHS. The sequence $V_{put} NP PP$ in our example is then assumed to be emitted by some Markov model of $VP_{put}$ rules (again with backoff from put). Collins (1997, model 2) uses a more sophisticated model in which all arguments in this sequence are generated jointly, as in a Treebank grammar, and then a Markov process is used to insert adjuncts among the arguments.

While Treebank models overfit the training data, Markov models underfit. A simple compromise (novel to this paper) is a hybrid Treebank/Markov model, which backs off from a Treebank model to a Markov. Like this paper’s main proposal, it can learn well-observed idiosyncratic rules but generalizes when data are sparse.\footnote{Nonstandardly, this allows infinitely many rules with $p>0$.}

\footnote{One might do better by backing off to word clusters, which Charniak (1997) did find provided a small benefit.}

\footnote{Carroll and Rooth (1998) used a similar hybrid technique for non-PCFGs. See also An et al. (1998) for related work.}

Appeared in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), Philadelphia, July 2002.
These models are beaten by our rather different model, **transformational smoothing**, which learns common rules and common edits to them. The comparison is a direct one, based on the perplexity or cross-entropy of the trained models on a test set of $S \to \cdots$ rules.\(^4\)

A subtlety is that two annotation styles are possible. In the Penn Treebank, `put` is the head of three constituents (\(V_P, VP\), and \(S\), where underlining denotes a head child) and joins with different dependents at different levels:

- \([S [NP Jim] [VP [V_P put] [NP pizza] [PP in the oven]]]\)

In the flattened or dependency version that we prefer, each word joins with all of its dependents at once:

- \([S [NP Jim] put [NP pizza] [PP in the oven]]\)

A PCFG generating the flat structure must estimate \(p(NP \ put \ NP \ PP \mid S_{\text{put}})\). A non-flat PCFG adds the dependents of `put` in 3 independent steps, so in effect it factors the flat rule’s probability into 3 supposedly independent “subrule probabilities,” \(p(NP \ VP_{\text{put}} \mid S_{\text{put}}) \cdot p(VP_{\text{put}} \ NP \ PP \mid VP_{\text{put}}) \cdot p(\text{put} \mid V_P_{\text{put}})\).

Our evaluation judges the estimates of flat-rule probabilities. Is it better to estimate these directly, or as a product of estimated subrule probabilities?\(^5\) Transformational smoothing is best applied to the former, so that the edit operations can freely rearrange all of a word’s dependents. We will see that the Markov and Treebank/Markov models also work much better this way—a useful finding.

### 3 The Abstract Problem: Designing Priors

This section outlines the Bayesian approach to learning probabilistic grammars (for us, estimating a distribution over flat CFG rules). By choosing among the many grammars that could have generated the training data, the learner is choosing how to generalize to novel sentences.

To guide the learner’s choice, one can explicitly specify a prior probability distribution \(p(\theta)\) over possible grammars \(\theta\), which themselves specify probability distributions over strings, rules, or trees. A learner should seek \(\theta\) that maximizes \(p(\theta) \cdot p(D \mid \theta)\), where \(D\) is the set of strings, rules, or trees observed by the learner. The first factor favors regularity (“pick an \textit{a priori} plausible grammar”), while the second favors fitting the idiosyncrasies of the data, especially the commonest data.\(^6\)

---

\(^4\)All the methods evaluated here apply also to full PCFGs, but verb-headed rules \(S \to \cdots\) present the most varied, interesting cases. Many researchers have tried to learn verb subcategorization, though usually not probabilistic subcategorization.

\(^5\)In testing the latter case, we sum over \textit{all} possible internal bracketings of the rule. We do train this case on the true internal bracketing, but it loses even with this unfair advantage.

\(^6\)This approach is called semi-Bayesian or Maximum A Posteriori distribution configurations, which can both unsupervised and supervised learning. (In the semi-supervised experiments here, training data is not raw text but a sparse sample of flat rules.)

Indeed a good deal of syntax induction work has been carried out in just this framework (Stolcke and Omohundro, 1994; Chen, 1996; De Marcken, 1996; Grünwald, 1996; Osborne and Briscoe, 1997). However, all such work to date has adopted rather simple prior distributions. Typically, it has defined \(p(\theta)\) to favor PCFGs whose rules are few, short, nearly equiprobable, and defined over a small set of nonterminals. Such definitions are convenient, especially when specifying an encoding for MDL, but since they treat all rules alike, they may not be good descriptions of linguistic plausibility. For example, they will never penalize the absence of a predictable rule.

A prior distribution can, however, be used to encode various kinds of linguistic notions. After all, a prior is really a soft form of Universal Grammar: it gives the learner enough prior knowledge of grammar to overcome Chomsky’s “poverty of the stimulus” (i.e., sparse data).

- A preference for small or simple grammars, as above.
- Substantive preferences, such as a preference for verbs to take 2 nominal arguments, or to allow PP adjuncts.
- Preferences for systematicity, such as a preference for the rules to be consistently head-initial or head-final.

This paper shows how to design a prior that favors a certain kind of systematicity. Lexicalized grammars for natural languages are very large—each word specifies a distribution over all possible dependency rules it could head—but they tend to have internal structure. The new prior prefers grammars \textit{in which a rule’s probability can be well-predicted from the probabilities of other rules}, using linguistic transformations such as edit operations.

For example, \(p(NP \ Adv \ w \ put \ NP \ PP \mid S_w)\) correlates with \(p(NP \ w \ NP \ PP \mid S_w)\). Both numbers are high for \(w = \text{put}\), medium for \(w = \text{fund}\), and low for \(w = \text{sleep}\). The slope of the regression line has to do with the rate of preverbal Adv-insertion in English.

The correlation is not perfect (some verbs are especially prone to adverbial modification), which is why we will only model it with a prior. To just the extent that evidence about \(w\) is sparse, the prior will cause the learner to smooth the two probabilities toward the regression line.

### 4 Patterns Worth Modeling

Before spelling out our approach, let us do a sanity check. A **frame** is a flat rule whose headword is replaced with teriori learning, since it is equivalent to maximizing \(p(\theta \mid D)\). It is also equivalent to Minimum Description Length (MDL) learning, which minimizes the total number of bits \(\ell(\theta) + \ell(D \mid \theta)\) needed to encode grammar and data, because one can choose an encoding scheme where \(\ell(x) = -\log_2 p(x)\), or conversely, define probability distributions by \(p(x) = 2^{-\ell(x)}\).
The variable “—” (corresponding to w above) Table 1 illustrates that in the Penn Treebank, if frequent rules with frame α imply matching rules with frame β, there are usually edit operations (section 1) to easily turn α into β.

How about rare rules, whose probabilities are most in need of smoothing? Are the same edit transformations that we can learn from frequent cases (Table 1) appropriate for predicting the rare cases? The very rarity of these rules makes it impossible to create a table like Table 1.

However, rare rules can be measured in the aggregate, and the result suggests that the same kinds of transformations are indeed useful—perhaps even more useful—in predicting them. Let us consider the set of R of 2,809,545 possible flat rules that stand at edit distance 1 from the set of S → ⋯ rules observed in our English training data. That is, a rule such as Sput → NP put NP is in R if it did not appear in training data itself, but could be derived by a single edit from some rule that did appear.

A bigram Markov model (section 2) was used to identify 2,714,763 rare rules in R—those that were predicted to occur with probability < 0.0001 given their headwords. 79 of these rare rules actually appeared in a development-data set of 1423 rules. The bigram model would have expected only 26.2 appearances, given the lexical headwords in the test data set. The difference is statistically significant (p < 0.001, bootstrap test).

In other words, the bigram model underpredicts the edit-distance “neighbors” of observed rules by a factor of 3.7 One can therefore hope to use the edit transformations to improve the performance on the bigram model. For example, the

Delete Y transformation recognizes that if ⋯ X Y Z ⋯ has been observed, then ⋯ X Z ⋯ is plausible even if the bigram X Z has not previously been observed.

Presumably, edit operations are common because they modify a rule in semantically useful ways, allowing the filler of a semantic role to be expressed (Insert), suppressed (Delete), retyped (Substitute), or heavy-shifted (Swap). Such “valency-affecting operations” have repeatedly been invoked by linguists; they are not confined to English.8 So a learner of an unknown language can reasonably expect a priori that flat rules related by edit operations may have related probabilities.

However, which edit operations varies by language. Each language defines its own weighted, contextual, asymmetric edit distance. So the learner will have to discover how likely particular edits are in particular contexts. For example, it must learn the rates of preverbal Adv-insertion and right-edge P-insertion. Evidence about these rates comes mainly from the frequent rules.

5 A Transformation Model

The form of our new model is shown in Figure 1. The vertices are flat context-free rules, and the arcs between them represent edit transformations. The set of arcs leav-
ing any given vertex has total probability 1. The learner’s job is to discover the probabilities.

Fortunately, the learner does not have to learn a separate probability for each of the (infinitely) many arcs, since many of the arcs represent identical or similar edits. As shown in Figure 1, an arc’s probability is determined from meaningful features of the arc, using a conditional log-linear model of \( p(\text{arc} \mid \text{source vertex}) \). The learner only has to learn the finite vector \( \theta \) of feature weights. Arcs that represent similar transformations have similar features, so they tend to have similar probabilities.

This transformation model is really a PCFG with unusual parameterization. That is, for any value of \( \theta \), it defines a language-specific probability distribution over all possible context-free rules (graph vertices). To sample from this distribution, take a random walk from the special vertex \( \text{START} \) to the special vertex \( \text{HALT} \). The rule at the last vertex reached before \( \text{HALT} \) is the sample.

This sampling procedure models a process where the speaker chooses an initial rule and edits it repeatedly. The random walk might reach \( \text{S}_{\text{fund}} \rightarrow \text{To fund NP} \) in two steps and simply halt there. This happens with probability 0.0011 \( \cdot \frac{\exp \theta_1}{Z_3} \cdot \frac{\exp \theta_0}{Z_2} \). Or, having arrived at \( \text{S}_{\text{fund}} \rightarrow \text{To fund NP} \), it might transform it into \( \text{S}_{\text{fund}} \rightarrow \text{To fund PP NP} \) and then further to \( \text{S}_{\text{fund}} \rightarrow \text{To fund NP PP} \) before halting.

Thus, \( p_\theta(\text{S}_{\text{fund}} \rightarrow \text{To fund NP PP}) \) denotes the probability that the random walk somehow reaches \( \text{S}_{\text{fund}} \rightarrow \text{To fund NP PP} \) and halts there. Conditionalizing this probability gives \( p_\theta(\text{To fund NP PP} \mid \text{S}_{\text{fund}}) \), as needed for the PCFG.\(^9\)

Given \( \theta \), it is nontrivial to solve for the probability distribution over grammar rules \( e \). Let \( I_\theta(e) \) denote the flow to vertex \( e \). This is defined to be the total probability of all paths from \( \text{START} \) to \( e \). Equivalently, it is the expected number of times \( e \) would be visited by a random walk from \( \text{START} \). The following recurrence defines \( p_\theta(e) \):\(^{10}\)

\[
I_\theta(e) = \delta_{e, \text{START}} + \sum_{e'} I_\theta(e') \cdot p(e' \rightarrow e) \quad (1)
\]

\[
p_\theta(e) = I_\theta(e) \cdot p(e \rightarrow \text{HALT}) \quad (2)
\]

Since solving the large linear system (1) would be prohibitively expensive, in practice we use an approximate relaxation algorithm (Eisner, 2001) that propagates flow through the graph until near-convergence. In general this may underestimate the true probabilities somewhat.

Now consider how the parameter vector \( \theta \) affects the distribution over rules, \( p_\theta(e) \), in Figure 1:

- By raising the initial weight \( \theta_1 \), one can increase the flow to \( \text{S}_{\text{fund}} \rightarrow \text{To fund NP} \), \( \text{S}_{\text{merge}} \rightarrow \text{To merge NP} \), and the like. By equation (2), this also increases the probability of these rules. But the effect also feeds through the graph to increase the flow and probability at those rules’ descendants in the graph, such as \( \text{S}_{\text{merge}} \rightarrow \text{To merge NP PP} \).

So a single parameter \( \theta_1 \) controls a whole complex of rule probabilities (roughly speaking, the infinitival transitives). The model thereby captures the fact that, although that change the LHS or headword of a rule, so it is trivial to find the divisor \( p_\theta(S_{\text{fund}}) \). In Figure 1 it is 0.0011. But in general, LHS-changing transformations can be useful (Eisner, 2001).

\(^9\)The experiments of this paper do not allow transformations

\(^{10}\)Where \( \delta_{x,y} = 1 \) if \( x = y \), else \( \delta_{x,y} = 0 \).
rules are mutually exclusive events whose probabilities sum to 1, transformationally related rules have positively correlated probabilities that rise and fall together.

- The exception weight $\theta_9$ appears on all and only the arcs to $S_{\text{merge}} \rightarrow \text{To merge NP PP}$. That rule has even higher probability than predicted by PP-insertion as above (since merge, unlike fund, actually tends to subcategorize for $F_{\text{with}}$). To model its idiosyncratic probability, one can raise $\theta_9$. This “lists” the rule specially in the grammar. Rules derived from it also increase in probability (e.g., $S_{\text{merge}} \rightarrow \text{Adv merge NP PP}$), since again the effect feeds through the graph.

- The generalization weight $\theta_3$ models the strength of the PP-insertion relationship. Equations (1) and (2) imply that $p_\theta(S_{\text{fund}} \rightarrow \text{To fund NP PP})$ is modeled as a linear combination of the probabilities of that rule’s parents in the graph. $\theta_3$ controls the coefficient of $p_\theta(S_{\text{fund}} \rightarrow \text{To fund NP})$ in this linear combination, with the coefficient approaching zero as $\theta_3 \rightarrow -\infty$.

- Narrower generalization weights such as $\theta_4$ and $\theta_5$ control where $\text{PP}$ is likely to be inserted. To learn the feature weights is to learn which features of a transformation make it probable or improbable in the language.

Note that the vertex labels, graph topology, and arc parameters are language independent. That is, Figure 1 is supposed to represent Universal Grammar: it tells a learner what kinds of generalizations to look for. The language-specific part is $\theta$, which specifies which generalizations and exceptions help to model the data.

6 The Prior

The model has more parameters than data. Why? Beyond the initial weights and generalization weights, in practice we allow one exception weight (e.g., $\theta_8$, $\theta_9$) for each rule that appeared in training data. (This makes it possible to learn arbitrary exceptions, as in a Treebank grammar.)

Parameter estimation is nonetheless possible, using a prior to help choose among the many values of $\theta$ that do a reasonable job of explaining the training data. The prior constrains the degrees of freedom: while many parameters are available in principle, the prior will ensure that the data are described using as few of them as possible.

The point of reparameterizing a PCFG in terms of $\theta$, as in Figure 1, is precisely that only one parameter is needed per linguistically salient property of the PCFG. Making $\theta_3 > 0$ creates a broadly targeted transformation. Making $\theta_9 \neq 0$ or $\theta_1 \neq 0$ lists an idiosyncratic rule, or class of rules, together with other rules derived from them. But it takes more parameters to encode less systematic properties, such as narrowly targeted edit transformations ($\theta_1$, $\theta_2$) or families of unrelated exceptions.

A natural prior for the parameter vector $\theta \in \mathbb{R}^k$ is therefore specified in terms of a variance $\sigma^2$. We simply say that the weights $\theta_1, \theta_2, \ldots, \theta_k$ are independent samples from the normal distribution with mean 0 and variance $\sigma^2 > 0$ (Chen and Rosenfeld, 1999):

$$\Theta \sim N(0, \sigma^2) \times N(0, \sigma^2) \times \cdots \times N(0, \sigma^2)$$

or equivalently, that $\Theta$ is drawn from a multivariate Gaussian with mean 0 and diagonal covariance matrix $\sigma^2 I$, i.e., $\Theta \sim N(0, \sigma^2 I)$.

This says that a priori, the learner expects most features in Figure 1 to have weights close to zero, i.e., to be irrelevant. Maximizing $p(\theta) \cdot p(D | \theta)$ means finding a relatively small set of features that adequately describe the rules and exceptions of the grammar. Reducing the variance $\sigma^2$ strengthens this bias toward simplicity.

For example, if $S_{\text{fund}} \rightarrow \text{To fund NP PP}$ and $S_{\text{merge}} \rightarrow \text{To fund NP PP}$ are both observed more often than the current $p_\theta$ distribution predicts, then the learner can follow either (or both) of two strategies: raise $\theta_8$ and $\theta_9$, or raise $\theta_3$. The former strategy fits the training data only; the latter affects many disparate arcs and leads to generalization. The latter strategy may harm $p(D | \theta)$ but is preferred by the prior $p(\theta)$ because it uses one parameter instead of two. If more than two words act like merge and fund, the pressure to generalize is stronger.

7 Perturbation Parameters

In experiments, we have found that a slight variation on this model gets slightly better results. Let $\theta_e$ denote the exception weight (if any) that allows one to tune the probability of rule $e$. We eliminate $\theta_e$ and introduce a different parameter $\pi_e$, called a perturbation, which is used in the following replacements for equations (1) and (2):

$$I_\theta(e) = \delta_{e, \text{START}} + \sum_{e'} I_\theta(e') \cdot \exp \pi_e \cdot p(e' \rightarrow e)$$

$$p_\theta(e) = I_\theta(e) \cdot \exp \pi_e \cdot p(e \rightarrow \text{HALT}) / Z$$

where $Z$ is a global normalizing factor chosen so that $\sum_e p_\theta(e) = 1$. The new prior on $\pi_e$ is the same as the old prior on $\theta_e$.

Increasing either $\theta_e$ or $\pi_e$ will raise $p_\theta(e)$; the learner may do this to account for observations of $e$ in training data. The probabilities of other rules consequently decrease so that $\sum_e p_\theta(e) = 1$. When $\pi_e$ is raised, all rules’ probabilities are scaled down slightly and equally (because $Z$ increases). When $\theta_e$ is raised, $e$ steals probability from its siblings, but these are similar to $e$ so tend to appear in test data if $e$ is in training data. Raising $\theta_e$ without disproportionately harming $e$’s siblings requires manipulation of many other parameters, which is discouraged by the prior and may also suffer from search error. We speculate that this is why $\pi_e$ works better.

---

\footnote{Raising the probability of an arc from $e'$ to $e$ decreases the probabilities of arcs from $e'$ to siblings of $e$, as they sum to 1.}
encountered, though after training, only a few hundred
that was actually explored (including bad arcs that com-
unsupervised learning problem.

rules, it was not annotated with the paths in Figure 1 that
(voxet) perplexity score, the model had to assign reason-

generated those rules, so estimating
reason-

θ
bank (Marcus et al., 1993). Thus, 
abilities of the form
θ
E
θ
p
S

To evaluate the quality of generalization, we used pre-
pared training data D and testing data E (Table 3).
Each dataset consisted of a collection of flat rules such as
S
put → NP put NP PP extracted from the Penn Tree-

Thus, p(D | θ, π) and
D
E
p
θ
π
E

The learner attempted to maximize
p(θ, π) · p(D | θ, π) by gradient ascent. This amounts to learning the
generalizations and exceptions that related the training
rules D. The evaluation measure was then the perplexi-
ity on test data, − log2 p(E | θ, π)/|E|. To get a good
(low) perplexity score, the model had to assign reason-
able probabilities to the many novel rules in E (Table 3).
For many of these rules, even the frame was novel.

Note that although the training data was prepared into
rules, it was not annotated with the paths in Figure 1 that
generated those rules, so estimating θ and π was still an
unsupervised learning problem.

The transformation graph had about 14 features per arc
(Table 2). In the finite part of the transformation graph
that was actually explored (including bad arcs that com-
pe with good ones), about 70000 distinct features were
encountered, though after training, only a few hundred

Table 2: Each Insert arc has 14 features. The features of any
given arc are found by instantiating the tuples above, as shown.
Each instantiated tuple has a weight specified in θ.

Table 3: Properties of the experimental data. “Novel” means
not observed in training. “Frame” was defined in section 4.

8 Evaluation
To evaluate the quality of generalization, we used pre-
pared training data D and testing data E (Table 3).
Each dataset consisted of a collection of flat rules such as
S
put → NP put NP PP extracted from the Penn Tree-

Thus, p(D | θ, π) and
D
E
p
θ
π
E

The learner attempted to maximize
p(θ, π) · p(D | θ, π) by gradient ascent. This amounts to learning the
generalizations and exceptions that related the training
rules D. The evaluation measure was then the perplexi-
ity on test data, − log2 p(E | θ, π)/|E|. To get a good
(low) perplexity score, the model had to assign reason-
able probabilities to the many novel rules in E (Table 3).
For many of these rules, even the frame was novel.

Note that although the training data was prepared into
rules, it was not annotated with the paths in Figure 1 that
generated those rules, so estimating θ and π was still an
unsupervised learning problem.

The transformation graph had about 14 features per arc
(Table 2). In the finite part of the transformation graph
that was actually explored (including bad arcs that com-
pe with good ones), about 70000 distinct features were
encountered, though after training, only a few hundred

Table 3: Properties of the experimental data. “Novel” means
not observed in training. “Frame” was defined in section 4.

8 See (Eisner, 2001) for full details of data preparation,
model structure, parameter initialization, backoff levels for the
comparison models, efficient techniques for computing the ob-
jective and its gradient, and more analysis of the results.

Table 4: Perplexity of the test set under various models. (a) Full
training set. (b) Half training set (sections 0–7 only).

Table: Perplexity of the test set under various models. (a) Full
training set. (b) Half training set (sections 0–7 only).

feature weights were substantial, and only a few thousand
were even far enough from zero to affect performance.
There was also a parameter πe for each observed rule e.

Results are given in Table 4a, which compares the
transformation model to various competing models dis-
cussed in section 2. The best (smallest) perplexities ap-
pear in boldface. The key results:

• The transformation model was the winner, reducing
perplexity by 20% over the best model replicated from
previous literature (a bigram model).

• Much of this improvement could be explained by the
transformation model’s ability to model exceptions.
Adding this ability more directly to the bigram model,
using the new Treebank/Markov approach of section 2,
also reduced perplexity from the bigram model, by 6%
or 14% depending on whether Katz or one-count backoff
was used, versus the transformation model’s 20%.

• Averaging the transformation model with the best com-
peting model (Treebank/bigram) improved it by an addi-
tional 6%. So using transformations yields a total per-
plexity reduction of 12% over Treebank/bigram, and 24%
over the best previous model from the literature (bigram).

• What would be the cost of achieving such a perplexity
improvement by additional annotation? Training the av-
eraged model on only the first half of the training set, with
no further tuning of any options (Table 4b), yielded a test
set perplexity of 118.0. So by using transformations, we
can achieve about the same perplexity as the best model
without transformations (Treebank/bigram, 116.2), using
only half as much training data.

12 Back off from Treebank grammar with Katz vs. one-count
backoff (Chen and Goodman, 1996) (Note: One-count was al-
ways used for backoff within the n-gram and Collins models.)

See section 2 for discussion

Collins (1997, model 2)

Average of transformation model with best other model

Table 4: Perplexity of the test set under various models. (a) Full
training set. (b) Half training set (sections 0–7 only).
This is an encouraging result for the use of the method in less supervised contexts (although results on a noisy dataset would be more convincing in this regard).

- The competing models from the literature are best used to predict flat rules directly, rather than by summing over their possible non-flat internal structures, as has been done in the past. This result is significant in itself. Extending Johnson (1998), it shows the inappropriateness of the traditional independence assumptions that build up a frame by several rule expansions (section 2).

Figure 2 shows that averaging the transformation model with the Treebank/bigram model improves the latter not merely on balance, but across the board. In other words, there is no evident class of phenomena for which incorporating transformations would be a bad idea.

- Transformations particularly helped raise the estimates of the low-probability novel rules in test data, as hoped.
- Transformations also helped on test rules that had been observed once in training with relatively infrequent words. (In other words, the transformation model does not discount singletons too much.)
- Transformations hurt slightly on balance for rules observed more than once in training, but the effect was tiny.

All these differences are slightly exaggerated if one compares the transformation model directly with the Treebank/bigram model, without averaging.

The transformation model was designed to use edit operations in order to generalize appropriately from a word’s observed frames to new frames that are likely to appear with that word in test data. To directly test the model’s success at such generalization, we compared it to the bigram model on a pseudo-disambiguation task.

Each instance of the task consisted of a pair of rules from test data, expressed as (word, frame) pairs \((w_1, f_1)\) and \((w_2, f_2)\), such that \(f_1\) and \(f_2\) are “novel” frames that did not appear in training data (with any headword).

Each model was then asked: Does \(f_1\) go with \(w_1\) and \(f_2\) with \(w_2\), or vice-versa? In other words, which is bigger, \(p(f_1 \mid w_1) \cdot p(f_2 \mid w_2)\) or \(p(f_2 \mid w_1) \cdot p(f_1 \mid w_2)\)?

Since the frames were novel, the model had to make the choice according to whether \(f_1\) or \(f_2\) looked more like the frames that had actually been observed with \(w_1\) in the past, and likewise \(w_2\). What this means depends on the model. The bigram model takes two frames to look alike if they contain many bigrams in common. The transformation model takes two frames to look alike if they are connected by a path of probable transformations.

The test data contained 62 distinct rules \((w, f)\) in which \(f\) was a novel frame. This yielded \(\frac{62 \times 61}{2} = 1891\) pairs of rules, leading to 1811 task instances after obvious ties were discarded.\(^1\)

Baseline performance on this difficult task is 50% (random guess). The transformation model chose correctly in 1595 of the 1811 instances (88.1%). Parameters for “memorizing” specific frames do not help on this task, which involves only novel frames, so the Treebank/bigram model had the same performance. By contrast, the transformation model got 1669 of 1811 correct (92.2%), for a more-than-34% reduction in error rate. (The development set showed similar results.) However, since the 1811 task instances were derived non-independently from just 62 novel rules, this result is based on a rather small sample.

9 Discussion

This paper has presented a nontrivial way to reparameterize a PCFG in terms of “deep” parameters representing transformations and exceptions. A linguistically sensible prior was natural to define over these deep parameters.

Famous examples of “deep reparameterization” are the Fourier transform in speech recognition and the SVD transform for Latent Semantic Analysis in IR. Like our technique, they are intended to reveal significant structure through the leading parameters while relegating noise and exceptions to minor parameters. Such representations

\(^{13}\)An obvious tie is an instance where \(f_1 = f_2\), or where both \(w_1\) and \(w_2\) were novel headwords. (The 62 rules included 11 with novel headwords.) In such cases, neither the bigram nor the transformation model has any basis for making its decision: the probabilities being compared will necessarily be equal.
make it easier to model the similarity or probability of the objects at hand (waveforms, documents, or grammars).

Beyond the fact that it shows at least a good perplexity improvement (it has not yet been applied to a real task), an exciting “big idea” aspect of this work is its flexibility in defining linguistically sensible priors over grammars. Our reparameterization is made with reference to a user-designed transformation graph (Figure 1). The graph need not be confined to edit distance transformations, or to the simple features of Table 2 (used here for comparability with the Markov models), which condition a transformation’s probability on local context.

In principle, the approach could be used to capture a great many linguistic phenomena. Figure 1 could be extended with more ambitious transformations, such as gapping, gap-threading, and passivization. The flat rules could be annotated with internal structure (as in TAG) and thematic roles. Finally, the arcs could bear further features. For example, the probability of an unaccusative movement (someone sank the boat → the boat sank) should depend on whether the headword is a change-of-state verb.

Indeed, Figure 1 can be converted to any lexicalized theory of grammar, such as categorial grammar, TAG, LFG, HPSG, or Minimalism. The vertices represent lexical entries and the arcs represent probabilistic lexical redundancy rules or metarules (see footnote 8). The transformation model approach is therefore a full stochastic treatment of lexicalized syntax—apparently the first to treat lexical redundancy rules, although (Briscoe and Copestake, 1999) give an ad hoc approach. See (Eisner, 2001; Eisner, 2002a) for more discussion.

It is worthwhile to compare the statistical approach here with some other approaches:

- Transformation models are similar to graphical models: they allow similar patterns of deductive and abductive inference from observations. However, the vertices of a transformation graph do not represent different random variables, but rather mutually exclusive values of the same random variable, whose probabilities sum to 1.

- Transformation models incorporate conditional log-linear (maximum entropy) models. As an alternative, one could directly build a conditional log-linear model of p(RHS | LHS). However, such a model would learn probabilities, not relationships. A feature weight would not really model the strength of the relationship between two frames e, e′ that share that feature. It would only influence both frames’ probabilities. If the probability of e were altered by some unrelated factor (e.g., an exception weight), then the probability of e′ would not respond.

- A transformation model can be regarded as a probabilistic FSA that consists mostly of ε-transitions. (Rules are only emitted on the arcs to HALT.) This perspective allows use of generic methods for finite-state parameter estimation (Eisner, 2002b). We are strongly interested in improving the speed of such methods and their ability to avoid local maxima, which are currently the major difficulty with our system, as they are for many unsupervised learning techniques. We expect to further pursue transformation models (and simpler variants that are easier to estimate) within this flexible finite-state framework.

The interested reader is encouraged to look at (Eisner, 2001) for a much more careful and wide-ranging discussion of transformation models, their algorithms, and their relation to linguistic theory, statistics, and parsing. Chapter 1 provides a good overview. For a brief article highlighting the connection to linguistics, see (Eisner, 2002a).

References

H. Alshawi. 1996. Head automata for speech translation. In Proceedings of ICSLP, Philadelphia, PA.
T. Briscoe and A. Copestake. 1999. Lexical rules in constraint-based grammar. Computational Linguistics, 25(4):487–526.
Bob Carpenter. 1991. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. Computational Linguistics, 17(3):301–313.
Glenn Carroll and Mats Rooth. 1998. Valence induction with a head-lexicalized PCFG. In Proceedings of EMNLP.
Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In Proc. of AAAI, 598–603.
Eugene Charniak. 2000. A maximum-entropy inspired parser. In Proceedings of NAACL.
Stanley Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques. In Proceedings of ACL.
Stanley F. Chen and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Carnegie Mellon University, February.
Stanley Chen. 1996. Building Probabilistic Models for Natural Language. Ph.D. thesis, Harvard University.
Michael J. Collins. 1997. Three generative, lexicalised models for statistical parsing. In Proceedings of ACL/EACL, 16–23.
Carl De Marcken. 1996. Unsupervised Language Acquisition. Ph.D. thesis, MIT.
Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. Proc. of COLING, 340–345.
Jason Eisner. 2001. Smoothing a Probabilistic Lexicon via Syntactic Transformations. Ph.D. thesis, Univ. of Pennsylvania.
Jason Eisner. 2002a. Discovering syntactic deep structure via Bayesian statistics. Cognitive Science, 26(3), May.
Jason Eisner. 2002b. Parameter estimation for probabilistic finite-state transducers. In Proceedings of the 40th ACL.
P. Gräw. 1996. A minimum description length approach to grammar inference. In S. Wermer et al., eds., Symbolic, Connectionist and Statistical Approaches to Learning for NLP, no. 1040 in Lecture Notes in AI, pages 203–216.
Mark Johnson. 1998. PCFG models of linguistic tree representations. Computational Linguistics, 24(4):613–632.
Beth Levin. 1993. English Verb Classes and Alternations: A Preliminary Investigation. University of Chicago Press.
M. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19(2):313–330.
Miles Osborne and Ted Briscoe. 1997. Learning stochastic category grammars. In Proceedings of CoNLL, 80–87. ACL.
A. Stolcke and S.M. Omohundro. 1994. Inducing probabilistic grammars by Bayesian model merging. In Proc. of ICGI.