Conditional Probabilistic Context-Free Grammars

Charles Sutton and Andrew McCallum

March 13, 2004

1 Introduction

In this note we present a discriminative framework for learning distributions over parse trees of context-free languages, which we call *conditional probabilistic context-free grammars (CPCFGs)*. The best-performing approaches to learning statistical parsing models are generative, in that they estimate the joint distribution $p(t, \mathbf{x})$ over parse trees t and strings $\mathbf{x}$. But often we are not interested in generating strings $\mathbf{x}$. Rather, we are given a string $\mathbf{x}$—a sentence from a newspaper, say—and want to find the most likely parse $t^* = \arg\max_t p(t|\mathbf{x})$. To solve this problem, it suffices to estimate the conditional distribution $p(t|\mathbf{x})$, in what is called a discriminative model. Generative models often have difficulty incorporating arbitrary, overlapping features of the input, for example, including capitalization, word suffixes, and semantic features from WordNet. In a discriminative model, incorporating such features is easier, because we do not have to model their distribution. For this reason, discriminatively-trained models have outperformed generatively-trained models in several tasks, including part-of-speech tagging [12, 9] and noun-phrase segmentation [14].

Also, in generative models much effort is spent on smoothing—combining probabilities from distributions that use different evidence—with the best-performing techniques often using complex interpolation or backoff schemes that require hand tuning. In the log-linear model we present, smoothing is accomplished simply by adding features of various specificity, and the weights are determined in the course of the maximum likelihood estimation.

Previous work has aimed to achieve the advantages of discriminative parsing by training a chain of classifiers to make the decisions needed by a parser [13, 7]. However, these have the disadvantage that Lafferty, McCallum, and Pereira [9] call *label bias*: Because each classifier in the chain is trained separately, earlier classifiers cannot adjust their weights to avoid mistakes that become apparent later in the sequence. Random fields avoid this difficulty because their training takes into account a global normalization factor, which implicitly causes weights for earlier decisions to be affected by later decisions.

Another possibility is to use a discriminative algorithm to rerank the top results of a generatively-trained model. Collins [4] reports a large increase in parsing accuracy . If the generative model is inaccurate, however, then the best parse might not be among the top ranked, so that reranking does not help. Using the rich features allowed by a discriminative model could more likely that the correct parse is among the top-ranked.

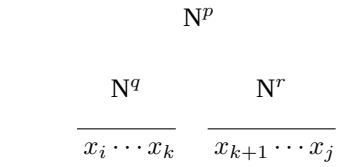

Figure 1: Parse tree that illustrates the arguments to $f_k(N^p, N^q N^r, x, i, j)$.

CPCFGs correspond to Stochastic Unification-Based Grammars [6], where the underlying grammar is restricted to be context-free. In the remainder of the document, we introduce CPCFGs, describing algorithms for parameter estimation and inference, outline some potential advantages of CPCFGs, and give a plan for future work.

2 PCFGs

The most well-known way of defining a distribution over parse trees is a *probabilistic context-free grammars (PCFG)* [10]. CPCFGs can be seen as conditionally-trained PCFGs, and the inference algorithms for CPCFGs closely follows those for PCFGs.

3 CPCFGs

3.1 Definition

In this section, we formally define conditional probabilitstic context-free grammars (CPCFGs). Intuitively, a CPCFG is a context-free grammar annotated with features and weights on each production. Then, given a parse tree t, every node $v \in t$ is assigned a weight, computed both from the features and weights of the production used to expand v, and from the terminal sequence that v spans. The weight of the entire tree t is simply the product of the weight of all its nodes, and the tree's probability $p(t|\mathbf{x})$ is its weight, normalized over all legal parse trees for $\mathbf{x}$.

Now, in a CPCFG, each production $N^p \rightarrow N^s N^t$ is annotated with a set of feature functions $\{f_k(N^p, N^s N^t, \mathbf{x}, i, j)\}$ with associated real-valued weights $\{\lambda_k\}$. In other words, each f_k takes as input a production and a segment of the sentence, and returns some feature of the parse tree that spans $\mathbf{x}_{i:j}$ using the production $N^p \rightarrow N^s N^t$, illustrated in figure 1. The weight λ_k is a real number that indicates how much the feature f_k should contribute to the weight of the production as a whole. High weights mean that a production is more likely. For example, one feature function might return 1 if and only if N^s equals NP and $\mathbf{x}_{i:j}$ contains a capitalized word.

More specifically, the weight of a node $v \in t$ with children $\mathrm{ch}(v)$ is computed from the features and weights as

$$w(v,t) = \exp\left\{\sum_k \lambda_k f_k(v, \mathrm{ch}(v), \mathrm{span}(v))\right\}, \tag{1}$$

where $\mathrm{span}(v)$ represents the portion of $\mathbf{x}$ spanned by v. Such a model is sometimes called *log-linear*, for $\log w(v,t)$ is a linear function of the features $\{f_k\}$.

Another way of thinking about a CPCFG is to imagine the derivation for a string. Recall that the derivation for a string w is a string $N^1 \Rightarrow \alpha_0 \Rightarrow \cdots \Rightarrow \alpha_m \Rightarrow w$ such that (a) $\alpha_i \in (V \cup \Sigma)^\star$ for all i, and (b) each α_{i+1} is a rewrite of α_i using exactly one production in P. Now, imagine that you are writing a derivation for a CFG, except that each time you use a rewrite rule, you pay a cost, equal to the negative of the logarithm of weight (calculated as above). Then the most likely tree in the CPCFG is simply one that corresponds to the minimum cost derivation.

More formally, the probability assigned to a parse tree t given a string $\mathbf{x}$ in a CPCFG G is given by

$$p(t|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{v \in t} \exp\left\{ \sum_k \lambda_k f_k(v, \mathrm{ch}(v), \mathrm{span}(v)) \right\} \tag{2}$$

where $Z(\mathbf{x})$ is a constant (that is, it depends only on the string, not the parse tree) that ensures that $p(t|\mathbf{x})$ is a valid probability distribution. It is defined as

$$Z(\mathbf{x}) = \sum_t \prod_{v \in t} \exp\left\{ \sum_k \lambda_k f_k(v, \mathrm{ch}(v), \mathrm{span}(v)) \right\}, \tag{3}$$

where the summation t ranges over all parse trees that yield $\mathbf{x}$.

3.2 Features

One of the main potential advantages of CPCFGs is their potential to include arbitrary features of the current context and the input string. Incorporating new features into a generative parser is often more complex, for then one has to take care that the parsing model is a correct generative model.

Here are some potentially useful features for a production $N^p \rightarrow N^s N^t$:

1. Many features of the form $f_k(N^p, N^s N^t, \mathbf{x}, i, j) = 1$ if and only if $x_{i:j}$ contains a specific word w.

2. $f_k(N^p, N^s N^t, \mathbf{x}, i, j) = 1$ if and only if $N^p =$ PP and $x_i =$ with. A large number of features like these provide the benefits of *lexicalization*, namely that the distribution of words in phrase is largely dependent on the phrase's head. Although a plain CPCFG does not attempt to determine heads directly, features that are correlated with headedness can provide some of the benefit.

   More complicated heuristics for determining heads can be used as well, for example, a feature that indicates the last word in a phrase that is not punctuation or a possessive marker ('s) can be useful in finding heads of noun phrases. In fact, one could train a classifier to choose the head of a long phrase given its words, and use the classifier's output as a feature.

3. Features that indicate whether certain word bigrams appear in the span. For example the bigram "to run" is more likely to appear in a verb phrase than a prepositional phrase.

4. Features that indicate membership in various domain-specific lexicons. For example, if a word not seen in training appears in a list of proper names extracted from the Web,

5. Deciding what tags to assign words that weren't seen in training can be incorporate directly into the models, using features that indicate capitalization and word prefixes and suffixes.

6. Length of the phrase $x_{i:j}$. A high weight on this can be used to prefer right-branching structures in English.

7. Semantically-motivated features. For example, including features for hypernyms in WordNet can allow the grammar to learn generalizations over words with similar semantic characteristics.

4 Inference

There are two main algorithmic problems for CPCFGs: finding the most likely parse of a string $\mathbf{x}$, and computing the partition function $Z(\mathbf{x})$. So far, we have described both of these as being computed over all possible parse trees, but both theoretically and in practice, there are far too many possible parse trees for this to be efficient if done naively. In this section, we describe algorithms for these two problems that require polynomial time in the length m of $\mathbf{x}$. It turns out that both of these algorithms are straightforward extensions of the standard algorithms for probabilistic context-free grammars (PCFGs) [10].

4.1 Parsing

The parsing problem for CPCFGs is, given a sentence $\mathbf{x}$, to find the most likely parse tree $t^* = \arg\max_t p(t|\mathbf{x})$. In this section, we describe how to compute the weight $w(t^*)$ of the most likely parse; computing the actual tree can be done with only a bit more bookkeeping.

Define $\mathcal{T}(p, i, j)$ as the set of all parse trees for the subsequence $\mathbf{x}_{i:j}$ that start with the nonterminal N^p. What we do is to compute, for every nonterminal N^p and every subsequence $\mathbf{x}_{i:j}$, the weight of the most likely tree $N^p \stackrel{\star}{\Rightarrow} \mathbf{x}_{i:j}$. We will call this weight $b_p(i, j)$, thus

$$b_p(i,j) \stackrel{\text{def}}{=} \max_{t \in \mathcal{T}(p,i,j)} \prod_{v \in t} \exp\left\{ \sum_k \lambda_k f_k(v, \text{ch}(v), \text{span}(v)) \right\}. \tag{4}$$

Observe that $b_1(1, m) = w(t^*)$, so that if we can compute b efficiently, then we're done. Consider expanding N^p using some production $N^p \rightarrow N^q N^r$, as in figure 1. Then the weight of the entire tree would be the cost for using that production times the weight of the subtrees. So we can maximize over all possible expansions in turn, first

maximizing over all possible expansions $N^q N^r$, and then maximizing over all possible boundaries ℓ between the two children:

$$b_p(i,j) \stackrel{\text{def}}{=} \max_{t \in \mathcal{T}(p,i,j)} \prod_{v \in t} \exp\left\{\sum_k \lambda_k f_k(v, \text{ch}(v), \text{span}(v))\right\} \tag{5}$$

$$= \max_{qr} \max_{\ell \in [i,j-1]} \max_{t_q \in \mathcal{T}(q,i,\ell)} \max_{t_r \in \mathcal{T}(r,\ell,j)} \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\} w(t_q) w(t_r) \tag{6}$$

$$= \max_{qr} \max_{\ell \in [i,j-1]} \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\} \left(\max_{t_q \in \mathcal{T}(q,i,\ell)} w(t_q)\right) \left(\max_{t_r \in \mathcal{T}(r,\ell,j)} w(t_r)\right) \tag{7}$$

$$= \max_{qr} \max_{\ell \in [i,j-1]} \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\} b(q,i,\ell) b(r,\ell+1,j) \tag{8}$$

But this means we can compute $b(1,1,m)$ recursively, by maintaining a table of all possible bs. This takes time proportional to m^3.

We would like to emphasize that although we assumed that the grammar was in Chomsky normal form, this was just to simplify notation, and that these quantities can be computed for general CFGs using a chart.

4.2 Computing $Z(x)$

If we want to know the actual probability of a tree, which we will need in Section 5, then we need to be able to compute the normalizing factor $Z(x)$. Essentially, this can be computed using all the equations of the previous section, except that every max is replaced with a sum.

First, we define the *inside cost* $\beta_p(i,j)$ for a nonterminal N_p which spans the sequence $\mathbf{x}_{i:j}$. We do this as

$$\beta_p(i,j) \stackrel{\text{def}}{=} \sum_{t \in \mathcal{T}(p,i,j)} \exp\left\{\sum_k \lambda_k f_k(v, \text{ch}(v), \text{span}(v))\right\} \tag{9}$$

Observe that $Z(x) = \beta_1(1,m)$. We can calculate β recursively as in the last section:

$$\beta_p(i,j) = \sum_{qr} \sum_{k=i}^{j-1} \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\} \beta_q(i,k) \beta_r(k+1,j) \tag{10}$$

This gives us one way to calculate $Z(x)$. There is another way, using what we will call the *outside costs*. It is perhaps easier to understand the outside cost in terms of derivations instead of parse trees. Define the set of outside derivations for a nonterminal N^p and subsequence $x_{i:j}$ as

$$\mathcal{O}_p(i,j) \stackrel{\text{def}}{=} \{\text{leftmost derivations } N_1 \stackrel{\star}{\Rightarrow} x_{1:i-1} N_r x_{j+1:m}\} \tag{11}$$

We can think of $\mathcal{O}_p(i,j)$ as being the set of all derivations of $\mathbf{x}$ that have N_r span $\mathbf{x}_{i:j}$ without specifying how. The outside cost $\alpha_p(i,j)$ is defined as a sum over all outside derivations:

$$\alpha_p(i,j) \stackrel{\text{def}}{=} \sum_{d \in \mathcal{O}_p(i,j)} w(d), \tag{12}$$

where $w(d)$ is the weight of the derivation d.

We can compute α recursively, essentially because any outside derivation of a nonterminal also contains outside derivations for its ancestors. Thus, while β was computed by recursing down the tree, we compute α by recursing higher up the tree. We will show how to compute $\alpha_r(i.j)$ for a node N^r that spans $x_{i:j}$. We will use the notation φ_i to refer to the production used in step i of a derivation d. Since we are working upward, we do recursion on all rules $N^p \to N^q N^r$, but now there are two cases: either N^q, the sibling of N^r, occurs either before N^r or afterwards. We sum over both cases, being careful not to count twice the case where the sibling of N^r is also labeled N^r:

$$\alpha_r(i,j) \stackrel{\text{def}}{=} \sum_{d \in \mathcal{O}_r(i,j)} \prod_{\varphi_i \in d} \exp\left\{\sum_k \lambda_k f_k(\text{lhs}(\varphi_i), \text{rhs}(\varphi_i), \text{span}(\varphi_i))\right\} \tag{13}$$

$$\begin{aligned}
&= \sum_{p \neq r,q} \sum_{\ell=i}^{j-1} \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\} \\
&\quad \times \left(\sum_{t_q \in \mathcal{T}(q,\ell,i-1)} w(t_q)\right)\left(\sum_{d_p \in \mathcal{O}_p(\ell,j)} w(d_p)\right) \\
&\quad + \sum_{pq} \sum_{\ell=j+1}^{m} \exp\left\{\sum_k \lambda_k f_k(N^p, N^r N^q, \mathbf{x}, i, j)\right\} \\
&\quad \times \left(\sum_{t_q \in \mathcal{T}(q,j+1,\ell)} w(t_q)\right)\left(\sum_{d_p \in \mathcal{O}_p(i,\ell)} w(d_p)\right)
\end{aligned} \tag{14}$$

$$\begin{aligned}
&= \sum_{p \neq r,q} \sum_{\ell=1}^{i-1} \alpha_p(\ell,j)\beta_q(\ell,i-1) \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\} \\
&\quad + \sum_{pq} \sum_{\ell=j+1}^{m} \alpha_p(i,\ell)\beta_q(j+1,\ell) \exp\left\{\sum_k \lambda_k f_k(N^p, N^r N^q, \mathbf{x}, i, j)\right\}
\end{aligned} \tag{15}$$

(Understanding these formulas really requires an illustration. Please imagine some illustration much like the one on page 395 of Manning & Schütze [10].)

There is little reason to use α to compute $Z(x)$, because we can get $Z(x)$ from β, and we would need β to compute α anyway. But α is necessary to do parameter estimation, to which we now turn.

5 Parameter Estimation

The parameters $\Lambda = \{\lambda_k\}$ of a CPCFG can be estimated by maximum likelihood. Suppose that we have a corpus of sentences $\mathcal{X} = \{\mathbf{x}^{(i)}\}$ which have been labeled with their correct parse trees $\mathcal{T} = \{t^{(i)}\}$. The *likelihood* of a parameter vector Λ is given by

$$\begin{aligned}
\mathcal{L}(\Lambda) &= \log p(\mathcal{T}|\mathcal{X}, \Lambda) && (16)\\
&= \log \prod_i p(t^{(i)}|\mathbf{x}^{(i)}, \Lambda) && (17)\\
&= \log \prod_i \frac{1}{Z(\mathbf{x}^{(i)})} \prod_{v \in t^{(i)}} \exp\left\{\sum_k \lambda_k f_k(v, \mathrm{ch}(v), \mathrm{span}(v))\right\} && (18)\\
&= \sum_i \sum_{v \in t^{(i)}} \sum_k \lambda_k f_k(v, \mathrm{ch}(v), \mathrm{span}(v)) - \sum_i \log Z(\mathbf{x}^{(i)}). && (19)
\end{aligned}$$

Then we choose the parameter vector Λ_{ML} that maximizes the likelihood $\mathcal{L}(\Lambda)$. This maximization can be done using standard algorithms as long as we can calculate the partial derivative $\partial\mathcal{L}/\partial\lambda_k$ of $\mathcal{L}$ with respect to each parameter λ_k. This derivative is given by

$$\frac{\partial \mathcal{L}}{\partial \lambda_k} = \sum_i \sum_{v \in t^{(i)}} f_k(v, \mathrm{ch}(v), \mathrm{span}(v)) - \sum_i \sum_t p(t|x^{(i)}) \sum_{v \in t} f_k(v, \mathrm{ch}(v), \mathrm{span}(v)). \tag{20}$$

As written, this requires summing over all parse trees t, which is infeasible. But we can use the algorithms from the last section to compute this in polynomial time. To simplify the discussion, define

$$Ef_k \stackrel{\text{def}}{=} \sum_t p(t|x^{(i)}) \sum_{v \in t} f_k(v, \mathrm{ch}(v), \mathrm{span}(v)). \tag{21}$$

First, we rewrite the sum over all parse trees to enumerate the parse trees differently. Let $R = N^p \rightarrow N^q N^r$ be the production associated with the parameter λ_k, and $\mathcal{T}(R, i, j)$ be the set of all trees where N^p spans $\mathbf{x}_{i:j}$, and N^p is expanded using rule R. Now we can rewrite Ef_k as:

$$\begin{aligned}
Ef_k &= \sum_t p(t|x^{(i)}) \sum_{v \in t} f_k(v, \mathrm{ch}(v), \mathrm{span}(v)) && (22)\\
&= \sum_{ij} \sum_{t \in \mathcal{T}(R,i,j)} p(t|\mathbf{x}^{(i)}) f_k(\mathrm{lhs}(R), \mathrm{rhs}(R), \mathbf{x}, i, j) && (23)\\
&= \sum_{ij} f_k(\mathrm{lhs}(R), \mathrm{rhs}(R), \mathbf{x}, i, j) \sum_{t \in \mathcal{T}(R,i,j)} p(t|\mathbf{x}^{(i)}). && (24)
\end{aligned}$$

Now, consider a derivation for t, with the steps arranged so that $N^1 \stackrel{\star}{\Rightarrow} x_{1:i-1} N^p x_{j+1:m} \stackrel{\star}{\Rightarrow} \mathbf{x}$. Note that such a derivation is always possible. Writing the derivation in this way shows that it first contains an outside derivation for N^p, followed by the expansion

$N^p \to N^q N^r$, and finally inside derivations for N^q and N^r. This means that the sum $\sum_t p(t|\mathbf{x}^{(i)})$ can be factorized as

$$\sum_t p(t|\mathbf{x}^{(i)}) = \frac{1}{Z(\mathbf{x})} \alpha_p(i,j)\beta_q(i,k)\beta_r(k,j) \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\}. \tag{25}$$

Since this requires three summations over the length of the string, i.e., the summations over i, j, and k, computing this probability requires cubic time in the length of the string.

Using equation 25, we have

$$Ef_k = \sum_{ij} f_k(\mathrm{lhs}(R), \mathrm{rhs}(R), \mathbf{x}, i, j) \sum_{\ell=i}^{j-1} \alpha_p(i,j)\beta_q(i,\ell)\beta_r(\ell,j) \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\}, \tag{26}$$

and the gradient is calculated as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \lambda_k} &= \sum_i \sum_{v \in t^{(i)}} f_k(v, \mathrm{ch}(v), \mathrm{span}(v)) \\ &\quad - \sum_i \sum_{ij} f_k(\mathrm{lhs}(R), \mathrm{rhs}(R), \mathbf{x}, i, j) \sum_{\ell=i}^{j-1} \alpha_p(i,j)\beta_q(i,\ell)\beta_r(\ell,j) \exp\left\{\sum_k \lambda_k f_k(N^p, N^q N^r, \mathbf{x}, i, j)\right\}.\end{aligned} \tag{27}$$

6 Efficient Training

CPCFGs can be evaluated on the Penn Treebank [11], a large collection of sentences hand-annotated with parse trees. The productions can be learned by reading them off the human-constructed parse trees [2], and then learning weights as in section 5. The difficulty here is in efficient training: Reading productions of the parse trees results in thousands of productions, which makes it expensive to parse 36 000 training sentences.

Exact training for CPCFGs (Section 5) requires parsing every training sentence.[1] This can be expensive with thousands of productions and tens of thousands of training sentences. For this reason, approximate methods for training are desirable.

1. Training on random subsets of training data at each iteration.

2. The number of iterations used by gradient descent can be reduced by good initialization of parameters. For example, the weights λ_k could be initialized the feature's frequency in the training set, or to its probability if the model were trained generatively.

3. Pseudolikelihood training [1] can be used to train based on local constraints rather than global constraints. However, it can potentially suffer from label bias.

[1] A naive implementation would also require reparsing every training sentence at each iteration of the gradient descent algorithm, but the charts can be saved between iterations.

4. Pruning infrequent rules when the grammar is being constructed (e.g., from a treebank), or pruning low-weight edges in the chart while the inside and outside values are being computed.

5. Perhaps a perceptron-based training algorithm (e.g., [5]) could train more efficiently than the approach in Section 5, because much more pruning of the chart is possible.

6. Probabilistic parsing can be viewed as inference in a suitably-constructed graphical model, i.e., with $O(m^2)$ nodes. It is possible that approximate inference methods from graphical models could be applied.

7 Conclusions

I am currently implementing CPCFGs so that I can compare their performance to other statistical parsers on the Penn Treebank. They may also have application to tasks like information extraction, semantic role labeling, relation extraction, and citation matching, with a suitably task-specific grammar. Within information extraction, grammars of this sort are likely to be useful in many domains with natural hierarchical structure, such as citations in research papers and address blocks in email and the Web. For example, the hierarchical HMMs of Skounakis et al. [15] are essentially conditionally-trained fixed-depth grammars for information extraction from biomedical abstracts.

Previous statistical parsers, e.g., [3], have achieved high accuracy in two ways: by conditioning statistics on more history in the derivation, and by *lexicalization*, which is conditioning statistics for a phrase on its head word. We believe that the rich feature sets possible with CPCFGs, as described in Section 3.2, can incorporate some of the benefits of lexicalization with lower computational cost: full lexicalized parsing takes $O(m^5)$ time, while parsing with a plain PCFG or a CPCFG takes $O(m^3)$ time.

Including more of the derivation history, however, may be worthwhile. Johnson [8] finds that great improvement in the performance of plain PCFGs is possible if one conditions statistics on the grandparent node as well as the parent node. In a CPCFG, this would correspond to having f_k depend also on the parent of N^p in Figure 1. It is highly likely that this change would lead to better CPCFG performance as well.

References

[1] Julian Besag. Efficiency of pseudolikelihood estimation for simple gaussian fields. *Biometrika*, 64(3):616–618, 1977.

[2] Eugene Charniak. Tree-bank grammars. In *proceedings of the Thirteenth National Conference on Artifical Intelligence (AAAI '96)*, pages 1031–1036, 1996.

[3] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[4] Michael Collins. Discriminative reranking for natural language parsing. In *Proc. 17th International Conf. on Machine Learning*, pages 175–182. Morgan Kaufmann, San Francisco, CA, 2000.

[5] Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

[6] Stuart Geman and Mark Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars'. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

[7] Ulf Hermjakob and Raymond Mooney. Learning parse and translation decisions from examples with rich context. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, pages 482–489, 1997.

[8] Mark Johnson. The effect of alternative tree representations on tree bank grammars. In *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning (NeMLaP3/CoNLL98)*, pages 39–48, 1998.

[9] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning*, 2001.

[10] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.

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

[12] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proc. of the 1996 Conference on Empirical Methods in Natural Language Proceeding (EMNLP 1996)*, 1996.

[13] Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *In Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, 1997.

[14] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003*. Association for Computational Linguistics, 2003.

[15] Marios Skounakis, Mark Craven, and Soumya Ray. Hierarchical hidden Markov models for information extraction. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.