Perceptron Training for a Wide-Coverage Lexicalized-Grammar Parser

Stephen Clark
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK
stephen.clark@comlab.ox.ac.uk

James R. Curran
School of Information Technologies
University of Sydney
NSW 2006, Australia
james@it.usyd.edu.au

Abstract

This paper investigates perceptron training for a wide-coverage CCG parser and compares the perceptron with a log-linear model. The CCG parser uses a phrase-structure parsing model and dynamic programming in the form of the Viterbi algorithm to find the highest scoring derivation. The difficulty in using the perceptron for a phrase-structure parsing model is the need for an efficient decoder. We exploit the lexicalized nature of CCG by using a finite-state supertagger to do much of the parsing work, resulting in a highly efficient decoder. The perceptron performs as well as the log-linear model; it trains in a few hours on a single machine; and it requires only a few hundred MB of RAM for practical training compared to 20 GB for the log-linear model. We also investigate the order in which the training examples are presented to the online perceptron learner, and find that order does not significantly affect the results.

1 Introduction

A recent development in data-driven parsing is the use of discriminative training methods (Riezler et al., 2002; Taskar et al., 2004; Collins and Roark, 2004; Turian and Melamed, 2006). One popular approach is to use a log-linear parsing model and maximise the *conditional* likelihood function (Johnson et al., 1999; Riezler et al., 2002; Clark and Curran, 2004b; Malouf and van Noord, 2004; Miyao and Tsujii, 2005). Maximising the likelihood involves calculating feature expectations, which is computationally expensive. Dynamic programming (DP) in the form of the inside-outside algorithm can be used to calculate the expectations, if the features are sufficiently local (Miyao and Tsujii, 2002); however, the memory requirements can be prohibitive, especially for automatically extracted, wide-coverage grammars. In Clark and Curran (2004b) we use cluster computing resources to solve this problem.

Parsing research has also begun to adopt discriminative methods from the Machine Learning literature, such as the perceptron (Freund and Schapire, 1999; Collins and Roark, 2004) and the large-margin methods underlying Support Vector Machines (Taskar et al., 2004; McDonald, 2006). Parser training involves decoding in an iterative process, updating the model parameters so that the decoder performs better on the training data, according to some training criterion. Hence, for efficient training, these methods require an efficient decoder; in fact, for methods like the perceptron, the update procedure is so trivial that the training algorithm essentially is decoding.

This paper describes a decoder for a lexicalized-grammar parser which is efficient enough for practical discriminative training. We use a lexicalized phrase-structure parser, the CCG parser of Clark and Curran (2004b), together with a DP-based decoder. The key idea is to exploit the properties of lexicalized grammars by using a finite-state supertagger prior to parsing (Bangalore and Joshi, 1999; Clark and Curran, 2004a). The decoder still uses the CKY algorithm, so the worst case complexity of

Proceedings of the 5th Workshop on Important Unresolved Matters, pages 9–16,
Ann Arbor, June 2005.

the parsing is unchanged; however, by allowing the supertagger to do much of the parsing work, the efficiency of the decoder is greatly increased in practice.

We chose the perceptron for the training algorithm because it has shown good performance on other NLP tasks; in particular, Collins (2002) reported good performance for a perceptron tagger compared to a Maximum Entropy tagger. Like Collins (2002), the decoder is the same for both the perceptron and the log-linear parsing models; the only change is the method for setting the weights. The perceptron model performs as well as the log-linear model, but is considerably easier to train.

Another contribution of this paper is to advance wide-coverage CCG parsing. Previous discriminative models for CCG (Clark and Curran, 2004b) required cluster computing resources to train. In this paper we reduce the memory requirements from 20 GB of RAM to only a few hundred MB, but without greatly increasing the training time or reducing parsing accuracy. This provides state-of-the-art CCG parsing with a practical development environment.

More generally, this work provides a practical environment for experimenting with discriminative models for phrase-structure parsing; because the training time for the CCG parser is relatively short (a few hours), experiments such as comparing alternative feature sets can be performed. As an example, we investigate the order in which the training examples are presented to the perceptron learner. Since the perceptron training is an online algorithm — updating the weights one training sentence at a time — the order in which the data is processed affects the resulting model. We consider random ordering; presenting the shortest sentences first; and presenting the longest sentences first; and find that the order does not significantly affect the final results.

We also use the random orderings to investigate model averaging. We produced 10 different models, by randomly permuting the data, and averaged the weights. Again the averaging was found to have no impact on the results, showing that the perceptron learner — at least for this parsing task — is robust to the order of the training examples.

The contributions of this paper are as follows. First, we compare perceptron and log-linear parsing models for a wide-coverage phrase-structure parser, the first work we are aware of to do so. Second, we provide a practical framework for developing discriminative models for CCG, reducing the memory requirements from over 20 GB to a few hundred MB. And third, given the significantly shorter training time compared to other discriminative parsing models (Taskar et al., 2004), we provide a practical framework for investigating discriminative training methods more generally.

2 The CCG Parser

Clark and Curran (2004b) describes the CCG parser. The grammar used by the parser is extracted from CCGbank, a CCG version of the Penn Treebank (Hockenmaier, 2003). The grammar consists of 425 lexical categories, expressing subcategorisation information, plus a small number of combinatory rules which combine the categories (Steedman, 2000). A Maximum Entropy supertagger first assigns lexical categories to the words in a sentence, which are then combined by the parser using the combinatory rules and the CKY algorithm. A log-linear model scores the alternative parses. We use the normal-form model, which assigns probabilities to single derivations based on the normal-form derivations in CCGbank. The features in the model are defined over local parts of the derivation and include word-word dependencies. A packed chart representation allows efficient decoding, with the Viterbi algorithm finding the most probable derivation.

The supertagger is a key part of the system. It uses a log-linear model to define a distribution over the lexical category set for each word and the previous two categories (Ratnaparkhi, 1996) and the forward backward algorithm efficiently sums over all histories to give a distibution for each word. These distributions are then used to assign a set of lexical categories to each word (Curran et al., 2006).

Supertagging was first defined for LTAG (Bangalore and Joshi, 1999), and was designed to increase parsing speed for lexicalized grammars by allowing a finite-state tagger to do some of the parsing work. Since the elementary syntactic units in a lexicalized grammar — in LTAG's case elementary trees and in CCG's case lexical categories – contain a significant amount of grammatical information, combining them together is easier than the parsing typically performed by phrase-structure parsers. Hence

Bangalore and Joshi (1999) refer to supertagging as *almost parsing*.

Supertagging has been especially successful for CCG: Clark and Curran (2004a) demonstrates the considerable increases in speed that can be obtained through use of a supertagger. The supertagger interacts with the parser in an adaptive fashion. Initially the supertagger assigns a small number of categories, on average, to each word in the sentence, and the parser attempts to create a spanning analysis. If this is not possible, the supertagger assigns more categories, and this process continues until a spanning analysis is found. The number of categories assigned to each word is determined by a parameter β in the supertagger: all categories are assigned whose forward-backward probabilities are within β of the highest probability category (Curran et al., 2006).

Clark and Curran (2004a) also shows how the supertagger can reduce the size of the packed charts to allow discriminative log-linear training. However, even with the use of a supertagger, the packed charts for the complete CCGbank require over 20 GB of RAM. Reading the training instances into memory one at a time and keeping a record of the relevant feature counts would be too slow for practical development, since the log-linear model requires hundreds of iterations to converge. Hence the packed charts need to be stored in memory. In Clark and Curran (2004b) we use a cluster of 45 machines, together with a parallel implementation of the BFGS training algorithm, to solve this problem.

The need for cluster computing resources presents a barrier to the development of further CCG parsing models. Hockenmaier and Steedman (2002) describe a generative model for CCG, which only requires a non-iterative counting process for training, but it is generally acknowledged that discriminative models provide greater flexibility and typically higher performance. In this paper we propose the perceptron algorithm as a solution. The perceptron is an online learning algorithm, and so the parameters are updated one training instance at a time. However, the key difference compared with the log-linear training is that the perceptron converges in many fewer iterations, and so it is practical to read the training instances into memory one at a time.

The difficulty in using the perceptron for training phrase-structure parsing models is the need for an efficient decoder (since perceptron training essentially is decoding). Here we exploit the lexicalized nature of CCG by using the supertagger to restrict the size of the charts over which Viterbi decoding is performed, resulting in an extremely effcient decoder. In fact, the decoding is so fast that we can estimate a state-of-the-art discriminative parsing model in only a few hours on a single machine.

3 Perceptron Training

The parsing problem is to find a mapping from a set of sentences $x \in X$ to a set of parses $y \in Y$. We assume that the mapping F is represented through a feature vector $\Phi(x, y) \in \mathcal{R}^d$ and a parameter vector $\overline{\alpha} \in \mathcal{R}^d$ in the following way (Collins, 2002):

$$F(x) = \operatorname*{argmax}_{y \in \text{GEN}(x)} \Phi(x, y) \cdot \overline{\alpha} \tag{1}$$

where $\text{GEN}(x)$ denotes the set of possible parses for sentence x and $\Phi(x, y) \cdot \overline{\alpha} = \sum_i \alpha_i \Phi_i(x, y)$ is the inner product. The learning task is to set the parameter values (the feature weights) using the training set as evidence, where the training set consists of examples (x_i, y_i) for $1 \leq i \leq N$. The decoder is an algorithm which finds the argmax in (1).

In this paper, Y is the set of possible CCG derivations and $\text{GEN}(x)$ enumerates the set of derivations for sentence x. We use the same feature representation $\Phi(x, y)$ as in Clark and Curran (2004b), to allow comparison with the log-linear model. The features are defined in terms of local subtrees in the derivation, consisting of a parent category plus one or two children. Some features are lexicalized, encoding word-word dependencies. Features are integer-valued, counting the number of times some configuration occurs in a derivation.

$\text{GEN}(x)$ is defined by the CCG grammar, plus the supertagger, since the supertagger determines how many lexical categories are assigned to each word in x (through the β parameter). Rather than try to recreate the adaptive supertagging described in Section 2 for training, we simply fix the the value of β so that $\text{GEN}(x)$ is the set of derivations licenced by the grammar for sentence x, given that value. β is now a parameter of the training process which we determine experimentally using development data. The β parameter can be thought of as determining the set of incorrect derivations which the training algorithm uses to "discriminate against", with a smaller value of β resulting in more derivations.

3.1 Feature Forests

The same decoder is used for both training and testing: the Viterbi algorithm. However, the packed representation of GEN(x) in each case is different. When running the parser, a lot of grammatical information is stored in order to produce linguistically meaningful output. For training, all that is required is a packed representation of the features on each derivation in GEN(x) for each sentence in the training data. The *feature forests* described in Miyao and Tsujii (2002) provide such a representation.

Clark and Curran (2004b) describe how a set of CCG derivations can be represented as a feature forest. The feature forests are created by first building packed charts for the training sentences, and then extracting the feature information. Packed charts group together equivalent chart entries. Entries are equivalent when they interact in the same manner with both the generation of subsequent parse structure and the numerical parse selection. In practice, this means that equivalent entries have the same span, and form the same structures and generate the same features in any further parsing of the sentence. Back pointers to the daughters indicate how an individual entry was created, so that any derivation can be recovered from the chart.

A feature forest is essentially a packed chart with only the feature information retained (see Miyao and Tsujii (2002) and Clark and Curran (2004b) for the details). Dynamic programming algorithms can be used with the feature forests for efficient estimation. For the log-linear parsing model in Clark and Curran (2004b), the inside-outside algorithm is used to calculate feature expectations, which are then used by the BFGS algorithm to optimise the likelihood function. For the perceptron, the Viterbi algorithm finds the features corresponding to the highest scoring derivation, which are then used in a simple additive update process.

3.2 The Perceptron Algorithm

The training algorithm initializes the parameter vector as all zeros, and updates the vector by decoding the examples. Each feature forest is decoded with the current parameter vector. If the output is incorrect, the parameter vector is updated by adding the feature vector of the correct derivation and subtracting the feature vector of the decoder output. Training typically involves multiple passes over the data. Figure 1 gives the algorithm, where N is the number of training sentences and T is the number of iterations over the data.

Inputs: training examples (x_i, y_i)
Initialisation: set $\overline{\alpha} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \overline{\alpha}$
    if $z_i \neq y_i$
      $\overline{\alpha} = \overline{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Outputs: $\overline{\alpha}$

Figure 1: The perceptron training algorithm

For all the experiments in this paper, we used the averaged version of the perceptron. Collins (2002) introduced the averaged perceptron, as a way of reducing overfitting, and it has been shown to perform better than the non-averaged version on a number of tasks. The averaged parameters are defined as follows: $\gamma_s = \sum_{t=1...T,i=1...N} \alpha_s^{t,i}/NT$ where $\alpha_s^{t,i}$ is the value of the sth feature weight after the tth sentence has been processed in the ith iteration.

A naive implementation of the averaged perceptron updates the accumulated weight for each feature after each example. However, the number of features whose values change for each example is a small proportion of the total. Hence we use the algorithm described in Daume III (2006) which avoids unnecessary calculations by only updating the accumulated weight for a feature f_s when α_s changes.

4 Experiments

The feature forests were created as follows. First, the value of the β parameter for the supertagger was fixed (for the first set of experiments at 0.004). The supertagger was then run over the sentences in Sections 2-21 of CCGbank. We made sure that every word was assigned the correct lexical category among its set (we did not do this for testing). Then the parser was run on the supertagged sentences, using the CKY algorithm and the CCG combinatory rules. We applied the same normal-form restrictions used in Clark and Curran (2004b): categories can only combine if they have been seen to combine in Sections 2-21 of CCGbank, and only if they do not violate the Eisner (1996a) normal-form constraints. This part of the process requires a few hundred MB of RAM to run the parser, and takes a few hours for Sections 2-21 of CCGbank. Any further training times or memory requirements reported do not include the resources needed to create the forests.

The feature forests are extracted from the packed chart representation used in the parser. We only use a feature forest for training if it contains the correct derivation (according to CCGbank). Some forests do not have the correct derivation, even though we ensure the correct lexical categories are present, because the grammar used by the parser is missing some low-frequency rules in CCGbank. The total number of forests used for the experiments was 35,370 (89% of Sections 2-21) . Only features which occur at least twice in the training data were used, of which there are 477,848. The complete set of forests used to obtain the final perceptron results in Section 4.1 require 21 GB of disk space.

The perceptron is an online algorithm, updating the weights after each forest is processed. Each forest is read into memory one at a time, decoding is performed, and the weight values are updated. Each forest is discarded from memory after it has been used. Constantly reading forests off disk is expensive, but since the perceptron converges in so few iterations the training times are reasonable.

In contrast, log-linear training takes hundreds of iterations to converge, and so it would be impractical to keep reading the forests off disk. Also, since log-linear training uses a batch algorithm, it is more convenient to keep the forests in memory at all times. In Clark and Curran (2004b) we use a cluster of 45 machines, together with a parallel implementation of BFGS, to solve this problem, but need up to 20 GB of RAM.

The feature forest representation, and our implementation of it, is so compact that the perceptron training requires only 20 MB of RAM. Since the supertagger has already removed much of the practical parsing complexity, decoding one of the forests is extremely quick, and much of the training time is taken with continually reading the forests off disk. However, the training time for the perceptron is still only around 5 hours for 10 iterations.

model	RAM	iterations	time (mins)
perceptron	20 MB	10	312
log-linear	19 GB	475	91

Table 1: Training requirements for the perceptron and log-linear models

Table 1 compares the training for the perceptron and log-linear models. The perceptron was run for 10 iterations and the log-linear training was run to convergence. The training time for 10 iterations of the perceptron is longer than the log-linear training, although the results in Section 4.1 show that the perceptron typically converges in around 4 iterations. The striking result in the table is the significantly smaller memory requirement for the perceptron.

4.1 Results

Table 2 gives the first set of results for the averaged perceptron model. These were obtained using Section 00 of CCGbank as development data. Gold-standard POS tags from CCGbank were used for all the experiments. The parser provides an analysis for 99.37% of the sentences in Section 00. The F-scores are based only on the sentences for which there is an analysis. Following Clark and Curran (2004b), accuracy is measured using F-score over the gold-standard predicate-argument dependencies in CCGbank. The table shows that the accuracy increases initially with the number of iterations, but converges quickly after only 4 iterations. The accuracy after only one iteration is also surprisingly high.

Table 3 compares the accuracy of the perceptron and log-linear models on the development data. LP is labelled precision, LR is labelled recall, and CAT is the lexical category accuracy. The same feature forests were used for training the perceptron and log-linear models, and the same parser and decoding algorithm were used for testing, so the results for the two models are directly comparable. The only difference in each case was the weights file used.[1]

The table also gives the accuracy for the perceptron model (after 6 iterations) when a smaller value of the supertagger β parameter is used during the

[1]Both of these models have parameters which have been optimised on the development data, in the log-linear case the Gaussian smoothing parameter and in the perceptron case the number of training iterations.

iteration	1	2	3	4	5	6	7	8	9	10
F-score	85.87	86.28	86.33	86.49	86.46	86.51	86.47	86.52	86.53	86.54

Table 2: Accuracy on the development data for the averaged perceptron ($\beta = 0.004$)

model	LP	LR	F	CAT
log-linear$_{\beta=0.004}$	87.02	86.07	86.54	93.99
perceptron$_{\beta=0.004}$	87.11	85.98	86.54	94.03
perceptron$_{\beta=0.002}$	87.25	86.20	86.72	94.08

Table 3: Comparison of the perceptron and log-linear models on the development data

forest creation (with the number of training iterations again optimised on the development data). A smaller β value results in larger forests, giving more incorrect derivations for the training algorithm to "discriminate against". Increasing the size of the forests is no problem for the perceptron, since the memory requirements are so modest, but this would cause problems for the log-linear training which is already highly memory intensive. The table shows that increasing the number of incorrect derivations gives a small improvement in performance for the perceptron.

Table 4 gives the accuracies for the two models on the test data, Section 23 of CCGbank. Here the coverage of the parser is 99.63%, and again the accuracies are computed only for the sentences with an analysis. The figures for the averaged perceptron were obtained using 6 iterations, with $\beta = 0.002$. The perceptron slightly outperforms the log-linear model (although we have not carried out significance tests). We justify the use of different β values for the two models by arguing that the perceptron is much more flexible in terms of the size of the training forests it can handle.

Note that the important result here is that the perceptron model performs *at least as well as* the log-linear model. Since the perceptron is considerably easier to train, this is a useful finding. Also, since the log-linear parsing model is a Conditional Random Field (CRF), the results suggest that the perceptron should be compared with a CRF for other tasks for which the CRF is considered to give state-of-the-art results.

model	LP	LR	F	CAT
log-linear$_{\beta=0.004}$	87.39	86.51	86.95	94.07
perceptron$_{\beta=0.002}$	87.50	86.62	87.06	94.08

Table 4: Comparison of the perceptron and log-linear models on the test data

5 Order of Training Examples

As an example of the flexibility of our discriminative training framework, we investigated the order in which the training examples are presented to the online perceptron learner. These experiments were particularly easy to carry out in our framework, since the 21 GB file containing the complete set of training forests can be sampled from directly. We stored the position on disk of each of the forests, and selected the forests one by one, according to some order.

The first set of experiments investigated ordering the training examples by sentence length. Buttery (2006) found that a psychologically motivated Categorial Grammar learning system learned faster when the simplest linguistic examples were presented first. Table 5 shows the results both when the shortest sentences are presented first and when the longest sentences are presented first. Training on the longest sentences first provides the best performance, but is no better than the standard ordering.

For the random ordering experiments, forests were randomly sampled from the complete 21 GB training file on disk, without replacement. The new forests file was then used for the averaged-perceptron training, and this process was repeated 9 times.

The number of iterations for each training run was optimised in terms of the accuracy of the resulting model on the development data. There was little variation among the models, with the best model scoring 86.84% F-score on the development data and the worst scoring 86.63%. Table 6 shows that the performance of this best model on the test data is only slightly better than the model trained using the CCGbank ordering.

iteration	1	2	3	4	5	6
Standard order	86.14	86.30	86.53	86.61	86.69	86.72
Shortest first	85.98	86.41	86.57	86.56	86.54	86.53
Longest first	86.25	86.48	86.66	86.72	86.74	86.75

Table 5: F-score of the averaged perceptron on the development data for different data orderings ($\beta = 0.002$)

perceptron model	LP	LR	F	CAT
standard order	87.50	86.62	87.06	94.08
best random order	87.52	86.72	87.12	94.12
averaged	87.53	86.67	87.10	94.09

Table 6: Comparison of various perceptron models on the test data

Finally, we used the 10 models (including the model from the original training set) to investigate model averaging. Corston-Oliver et al. (2006) motivate model averaging for the perceptron in terms of Bayes Point Machines. The averaged perceptron weights resulting from each permutation of the training data were simply averaged to produce a new model. Table 6 shows that the averaged model again performs only marginally better than the original model, and not as well as the best-performing "random" model, which is perhaps not surprising given the small variation among the performances of the component models.

In summary, the perceptron learner appears highly robust to the order of the training examples, at least for this parsing task.

6 Comparison with Other Work

Taskar et al. (2004) investigate discriminative training methods for a phrase-structure parser, and also use dynamic programming for the decoder. The key difference between our work and theirs is that they are only able to train on sentences of 15 words or less, because of the expense of the decoding.

There is work on discriminative models for dependency parsing (McDonald, 2006); since there are efficient decoding algorithms available (Eisner, 1996b), complete resources such as the Penn Treebank can used for estimation, leading to accurate parsers. There is also work on discriminative models for parse reranking (Collins and Koo, 2005). The main drawback with this approach is that the correct parse may get lost in the first phase.

The existing work most similar to ours is Collins and Roark (2004). They use a beam-search decoder as part of a phrase-structure parser to allow practical estimation. The main difference is that we are able to store the complete forests for training, and can guarantee that the forest contains the correct derivation (assuming the grammar is able to generate it given the correct lexical categories). The downside of our approach is the restriction on the locality of the features, to allow dynamic programming. One possible direction for future work is to compare the search-based approach of Collins and Roark with our DP-based approach.

In the tagging domain, Collins (2002) compared log-linear and perceptron training for HMM-style tagging based on dynamic programming. Our work could be seen as extending that of Collins since we compare log-linear and perceptron training for a DP-based wide-coverage parser.

7 Conclusion

Investigation of discriminative training methods is one of the most promising avenues for breaking the current bottleneck in parsing performance. The drawback of these methods is the need for an efficient decoder. In this paper we have demonstrated how the lexicalized nature of CCG can be used to develop a very efficient decoder, which leads to a practical development environment for discriminative training.

We have also provided the first comparison of a perceptron and log-linear model for a wide-coverage phrase-structure parser. An advantage of the perceptron over the log-linear model is that it is considerably easier to train, requiring 1/1000th of the memory requirements and converging in only 4 iterations.

Given that the global log-linear model used here (CRF) is thought to provide state-of-the-art performance for many NLP tasks, it is perhaps surprising that the perceptron performs as well. The evaluation in this paper was based solely on CCGbank, but we have shown in Clark and Curran (2007) that the CCG parser gives state-of-the-art performance, outperforming the RASP parser (Briscoe et al., 2006) by over 5% on DepBank. This suggests the need for more comparisons of CRFs and discriminative methods such as the perceptron for other NLP tasks.

Acknowledgements

James Curran was funded under ARC Discovery grants DP0453131 and DP0665973.

References

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the Interactive Demo Session of COLING/ACL-06*, Sydney, Austrailia.

Paula Buttery. 2006. Computational models for first language acquisition. Technical Report UCAM-CL-TR-675, University of Cambridge Computer Laboratory.

Stephen Clark and James R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING-04*, pages 282–288, Geneva, Switzerland.

Stephen Clark and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the ACL*, pages 104–111, Barcelona, Spain.

Stephen Clark and James R. Curran. 2007. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the ACL*, Prague, Czech Republic.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the ACL*, pages 111–118, Barcelona, Spain.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 40th Meeting of the ACL*, Philadelphia, PA.

S. Corston-Oliver, A. Aue, K. Duh, and E. Ringger. 2006. Multilingual dependency parsing using bayes point machines. In *Proceedings of HLT/NAACL-06*, New York.

James R. Curran, Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of COLING/ACL-06*, pages 697–704, Sydney, Austrailia.

Jason Eisner. 1996a. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Meeting of the ACL*, pages 79–86, Santa Cruz, CA.

Jason Eisner. 1996b. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th COLING Conference*, pages 340–345, Copenhagen, Denmark.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Meeting of the ACL*, pages 535–541, University of Maryland, MD.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop: Beyond shallow analyses - Formalisms and statistical modeling for deep analyses*, Hainan Island, China.

Ryan McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*, San Diego, CA.

Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd meeting of the ACL*, pages 83–90, University of Michigan, Ann Arbor.

Adwait Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the EMNLP Conference*, pages 133–142, Philadelphia, PA.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the ACL*, pages 271–278, Philadelphia, PA.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proceedings of the EMNLP conference*, pages 1–8, Barcelona, Spain.

Joseph Turian and I. Dan Melamed. 2006. Advances in discriminative parsing. In *Proceedings of COLING/ACL-06*, pages 873–880, Sydney, Australia.