Simple Semi-supervised Dependency Parsing

Abstract

We present a simple and effective semi-supervised method for training dependency parsers. We focus on the problem of lexical representation, introducing features that incorporate word clusters derived from a large unannotated corpus. We demonstrate the effectiveness of our approach in a series of dependency parsing experiments on the Penn Treebank, and we show that our cluster-based features yield substantial gains in performance across a wide variety of parsing configurations. For example, in the case of unlabeled second-order parsing, we improve from a baseline accuracy of 92.02% to 93.16%, representing a relative reduction in error of 14%. In addition, we demonstrate that our method also improves performance when small amounts of training data are available, and can roughly halve the amount of supervised data required to reach a desired level of performance.

1 Introduction

In natural language parsing, lexical information is seen as crucial to resolving ambiguous relationships, yet lexicalized statistics are sparse and difficult to estimate directly. It is thus attractive to consider intermediate entities which exist at a coarser level than the words themselves, yet capture the information necessary to resolve the relevant ambiguities.

In this paper, we introduce lexical intermediaries via a simple two-stage semi-supervised approach. First, we use a large unannotated corpus to define word clusters, and then we use that clustering to construct a new cluster-based feature mapping for a discriminative learner. We are thus relying on the ability of discriminative learning methods to identify and exploit informative features while remaining agnostic as to the origin of such features. To demonstrate the effectiveness of our approach, we conduct experiments in dependency parsing, which has been the focus of much recent research (Buchholz and Marsi, 2006; Nivre et al., 2007).

The idea of combining word clusters with discriminative learning has been previously explored by Miller et al. (2004), in the context of named-entity recognition, and their work directly inspired our research. However, our target task of dependency parsing involves more complex structured relationships than named-entity tagging; moreover, it is not at all clear that word clusters should have any relevance to syntactic structure. Nevertheless, our experiments demonstrate that word clusters can be quite effective in dependency parsing applications.

Although the primary goal of this paper is to improve the state of the art in dependency parsing, we are also interested in studying our approach in terms of its behavior as a semi-supervised learning algorithm. In general, semi-supervised learning can be motivated by two concerns: first, given a fixed amount of supervised data, we might wish to leverage additional unlabeled data to facilitate the utilization of the supervised corpus, increasing the performance of the model in absolute terms. Second, given a fixed target performance level, we might wish to use unlabeled data to reduce the amount of annotated data necessary to reach this target.

By conducting experiments on datasets of vary-
Ms. Haag plays Elianti.

Figure 1: An example of a labeled dependency tree. The tree contains a special token “*” which is always at the root of the structure. Each dependency arc is directed from head to modifier and is annotated with a label describing the function of the attachment.

ing sizes, we demonstrate that our cluster-based approach achieves large improvements over the baseline models, regardless of whether the supervised corpus is large or small. Viewed from a different perspective, our results indicate that cluster-based features can reduce the need for supervised data by roughly half, which is a substantial savings in data-annotation costs (see Section 4.2).

Finally, in addition to the new empirical results which we present, a significant contribution of this paper are the observations and insights which we made in the course of creating a successful semi-supervised parsing system. These observations and insights may be helpful in implementing semi-supervised methods for other complex problems in NLP.

The remainder of this paper is divided as follows: Section 2 provides background information on the clustering algorithm and dependency parsing algorithms we used in our experiments. Section 3 describes our cluster-based features, Section 4 presents our empirical results, Section 5 discusses related research, and Section 6 concludes with ideas for future research.

2 Background

2.1 Dependency parsing

Recent work (Buchholz and Marsi, 2006; Nivre et al., 2007) has focused on dependency parsing as an alternative to phrase-structure parsing. Dependency syntax represents syntactic information as a network of head-modifier dependency arcs, typically restricted to be a directed tree (see Figure 1). Dependency parsing depends critically on predicting head-modifier relationships, which can be difficult due to the sparse nature of word-to-word interactions. These bilexical interactions are thus ideal candidates for the application of coarse word proxies such as word clusters.

In this paper, we take a part-factored structured classification approach to dependency parsing. For a given sentence $x$, let $\mathcal{Y}(x)$ be the set of possible dependency structures. Each potential dependency structure $y \in \mathcal{Y}(x)$, in turn, is decomposed into a set of “parts” $r \in y$. In the simplest case, these parts are the dependency arcs themselves, yielding a first-order or “edge-factored” dependency parsing model. In higher-order parsing models, the parts can consist of interactions between more than two words. For example, the parser of McDonald et al. (2006) defines parts for sibling interactions, such as the trio “plays”, “Elianti”, and “.” in Figure 1. The Carreras (2007) parser has parts for both sibling interactions and grandparent interactions, such as the trio “*”, “plays”, and “Haag” in Figure 1. These kinds of higher-order part factorizations allow dependency parsers to obtain a limited form of context-sensitivity.

Given a factorization of dependency structures into parts, we rephrase the parsing problem as the following maximization:

$$y^*(x) = \arg\max_{y \in \mathcal{Y}(x)} \sum_{r \in y} Score(x, r)$$

$$= \arg\max_{y \in \mathcal{Y}(x)} \sum_{r \in y} w \cdot f(x, r)$$

In the second line above, we have assumed that the part-scoring function is a linear model with parameters $w$ and feature-mapping $f(\cdot)$. For many different part factorizations and definitions of the structure domains $\mathcal{Y}(x)$, it is possible to solve the above maximization efficiently, and many recent efforts have concentrated on designing new maximization algorithms with increased context-sensitivity (Eisner, 2000; McDonald et al., 2005b; McDonald and Pereira, 2006; Carreras, 2007).

In our experiments, we used the first-order parser of Eisner (2000) and the second-order parser of Carreras (2007). The feature mappings that we used in our experiments are described in Section 3.
Input: Training data $S = \{(x_i, y_i)\}_{i=1}^n$
Number of iterations $I$.
Initialization: Parameters $w = 0$
Summed parameters $v = 0$

for $t = 1 \ldots T$
  for $j = 1 \ldots n$
    $i \leftarrow$ random integer in $[1, n]$
    $y^* = \text{argmax}_{y \in Y(x_i)} \sum_{r \in y} w \cdot f(x_i, y)$
    if $y^* \neq y_i$
      $w = w + f(x_i, y_i) - f(x_i, y^*)$
    endif
    $v = v + w$
  endfor
endfor

Output: Averaged vector $v / (nT)$

Figure 2: Pseudocode for the structured averaged perceptron algorithm.

2.2 Averaged perceptron training

In order to train the parameters $w$ for each of our parsing configurations, we use the structured perceptron with parameter averaging (Freund and Schapire, 1998; Collins, 2002). The averaged perceptron is a simple online learning algorithm that nevertheless obtains good performance in a variety of settings. For completeness, we briefly describe this algorithm below.

The averaged perceptron begins with an initially zeroed parameter vector and proceeds in a series of iterations, which are in turn divided into a series of trials. Each trial involves selecting a random example from the training set,\textsuperscript{1} parsing that example, and checking the parser’s prediction against the gold-standard structure. If the parser made a mistake, then parameters are updated with the difference between the feature vectors of the gold-standard structure and the parser’s prediction. The output of the algorithm is not the final parameter vector, but the average of all parameter vectors across every trial in the training run. Pseudocode for the algorithm is given in Figure 2.

Note that the only computational requirement of the averaged perceptron algorithm is the parser itself (i.e., the argmax operation in Figure 2). In contrast, other parameter optimization frameworks like log-linear models (Lafferty et al., 2001) typically require additional inference mechanisms, such as the computation of marginal probability distributions. For complex structured models like higher-order dependency parsers (McDonald and Pereira, 2006; Carreras, 2007), implementing algorithms that compute marginal probabilities can require effort.

2.3 Brown clustering algorithm

In order to provide word clusters for our experiments, we used the Brown clustering algorithm (Brown et al., 1992). We chose to work with the Brown algorithm due to its simplicity and prior success in other NLP applications (Miller et al., 2004; Liang, 2005). However, we expect that our approach can function with other clustering algorithms (as in, e.g., Li and McCallum (2005)). We briefly describe the Brown algorithm below.

The input to the algorithm is a vocabulary of words to be clustered and a corpus of text containing these words. Initially, each word in the vocabulary is considered to be in its own distinct cluster. The algorithm then repeatedly merges the pair of clusters which yields the greatest improvement in the likelihood of the text corpus, according to a bigram language model defined on the word clusters. By tracing the pairwise merge operations, one obtains a hierarchical clustering of the words, which can be represented as a binary tree (see Figure 3).

\textsuperscript{1}A common variation is to enumerate the data in a deterministic sequential order rather than select random examples.
Within this tree, each cluster in the history of merge operations is uniquely identified by its path from the root, and this path can be compactly represented with a bit string, as in Figure 3. In order to obtain a particular clustering of the words, we follow Miller et al. (2004) and select all nodes at a certain depth from the root of the hierarchy. The resulting partition of the hierarchy defines a word clustering; note that the same partitioning can also be obtained by truncating each word’s bit-string to a fixed-length prefix (Miller et al., 2004).

In practice, a straightforward implementation of the Brown algorithm is intractable for realistic vocabulary sizes, due to the need to consider all possible pairwise mergings on each step. We used a modified version of the Brown algorithm which restricts the search space by placing a maximum\(^2\) on the number of available clusters (Liang, 2005). The clusters for our experiments were derived from the text of the BLLIP corpus (Charniak et al., 2000), which contains roughly 43 million words of Wall Street Journal text.

### 3 Feature design

Key to the success of our approach is the design of features which allow the word clusters to assist the parser. Our feature sets are similar to other feature sets in the literature (McDonald et al., 2005a; Carreras, 2007), so we will not attempt to give a complete description of our features in this section. Rather, we provide a high-level description of the features and concentrate on our methodology and motivations. In our experiments, we employed two different feature sets: a baseline feature set which draws upon “normal” information sources such as word forms and parts of speech, and a cluster-based feature set that also uses information derived from the Brown cluster hierarchy.

#### 3.1 Baseline features

Our first-order baseline feature set is similar to the feature set of McDonald et al. (2005a), and includes indicators of various combinations of word identity and part-of-speech tag for the head and modifier of each dependency, as well as certain combinations of contextual information. We augment the McDonald et al. (2005a) feature set with backed-off versions of the “Surrounding Word POS Features” which are indicators of the head-POS, modifier-POS, and one neighboring POS tag. We also add binned distance indicator features, which are active if the number of tokens between the head and modifier is greater than a threshold, for thresholds of 2, 5, 10, 20, 30, and 40 tokens. Our second-order baseline features are the same as those of Carreras (2007) and include indicators for triples of part of speech tags for sibling interactions and grandparent interactions, as well as additional bigram features based on pairs of words involved these higher-order interactions. A partial listing of baseline feature templates is provided in Table 1.

### 3.2 Cluster-based features

The first- and second-order cluster-based feature sets are supersets of the baseline feature sets: they include all of the baseline feature templates, and add an additional layer of features that incorporate word clusters. Following Miller et al. (2004), we use prefixes of the Brown cluster hierarchy to produce clusterings of varying granularity. We found that it was nontrivial to select the proper prefix lengths for the dependency parsing task; in particular, the prefix

---

**Table 1:** Examples of first-order baseline and cluster-based feature templates; each entry represents a class of indicator features based on tuples of information. For example, “ht,mt” represents a class of indicator features, one feature for each possible combination of head POS-tag and modifier POS-tag. Abbreviations: ht = head POS, hw = head word, hc4 = head cluster 4-bit prefix, hc6 = head cluster 6-bit prefix, hc* = full head cluster; mc4, mc6, mc* = likewise for modifier.

| Baseline | Cluster-based |
|----------|---------------|
| ht,mt    | hc4,mc4       |
| hw,mw    | hc6,mc6       |
| hw,ht,mt | hc*,mc*       |
| hw,ht,mw | hc4,mt        |
| ht,mw,mt | ht,mc4        |
| hw,mw,mt | hc6,mt        |
| hw,ht,mw,mt | ht,mc6    |
| ...      | hc4,mw        |
| hw,mc4   |               |

---

\(^2\)In all of our experiments, the maximum number of clusters was set to 1,000.
lengths used in the Miller et al. (2004) work (between 12 and 20 bits) performed poorly in dependency parsing.\(^3\) After experimenting with many different feature configurations, we eventually settled on a simple but effective methodology.

First, we found that it was helpful to employ two different types of word clusters:

1. Short prefixes of cluster labels, usually 4 or 6 bits, which we used as replacements for parts of speech.
2. Full (non-truncated) cluster labels,\(^4\) which we used as substitutes for word forms.

Using these two types of clusters, we generated new features by mimicking the template structure of the original baseline features. For example, the baseline feature set includes indicators for word-to-word and tag-to-tag interactions between the head and modifier of a dependency. In the cluster-based feature set, we correspondingly introduce new indicators for interactions between pairs of short cluster prefixes and pairs of full clusters. Some examples of cluster-based features are given in Table 1.

3.3 Discussion

As we continued to experiment with the cluster-based features, we discovered some other general guidelines that helped us construct a high-quality feature set. For example, our initial attempts focused on creating new features which used word-cluster exclusively. While these cluster-only features provided some benefit, we found that much greater gains were obtained by introducing “hybrid” features involving, e.g., one cluster and one part of speech. One possible explanation for this behavior is that the clusterings generated by the Brown algorithm are not necessarily relevant to syntax, thus features that only include clusters can be misleading or uninformative. It is therefore helpful to “anchor” the word clusters to word forms or part of speech tags.

As another example of a general guideline, we found that a form of vocabulary restriction greatly increased the performance of the cluster-based feature sets. Specifically, for any feature that includes a word form, we only generate this feature if the word in question is one of the top-\(N\) most frequent words in the corpus. If \(N\) is very large (e.g., the entire vocabulary) or very small, then the impact of the cluster-based features is neutral or even negative. When \(N\) is between roughly 100 and 1,000, however, the cluster-based features provide significant improvements over the baseline.\(^5\) We believe that by eliminating the lower-frequency words, the discriminative learner is encouraged to place more weight on the cluster-based features; the clusters thus serve as a kind of backed-off word form. Interestingly, when the same vocabulary restriction is applied to the baseline features, performance always suffers, with increasing severity as more words are discarded.

4 Experiments

We conducted experiments on the English Penn Treebank (Marcus et al., 1993), using a standard set of head-selection rules (Yamada and Matsumoto, 2003) to convert the phrase structure syntax of the Treebank to a dependency tree representation.\(^6\) We split the Treebank into a training set (Sections 2–21, containing 39,832 sentences), a development set (Section 22, containing 1,700 sentences), and a final test set (Section 23, containing 2,416 sentences). The data split and head rules were chosen to match the test conditions of McDonald et al. (2005a; 2006).

The main focus of our experiments is a comparison between the baseline and cluster-based feature sets. We also explored the effect of using or ignoring the dependency labels and varying the model order. Since our feature mappings are quite high-dimensional, we eliminated all features which occur only once in the training data. The resulting models still had very high dimensionality, with labeled parsing models containing as many as a billion features.\(^7\)

We trained our parsers using the averaged percep-

---

\(^3\)One possible explanation is that the kinds of distinctions required in a named-entity recognition task (e.g., “Alice” versus “Intel”) are much finer-grained than the kinds of distinctions relevant to syntax (e.g., “apple” versus “eat”).

\(^4\)Recall that our algorithm produces at most 1000 clusters, so full cluster labels are not equivalent to word forms.

\(^5\)In all of the experiments presented in this paper, we used the fixed value \(N = 800\).

\(^6\)We used Joakim Nivre’s “Penn2Malt” conversion tool (http://w3.msi.vxu.se/ nivre/research/Penn2Malt.html). Dependency labels were obtained via the “Malt” hard-coded setting.

\(^7\)After training, however, only a very small fraction of those features are active.
tron (Collins, 2002), which represents a balance between strong performance and fast training times. In order to select the number of iterations of perceptron training, we performed up to 30 iterations and selected the iteration which obtained the best accuracy on the development set. The part of speech tags for the development and final test data were automatically assigned by MXPOST (Ratnaparkhi, 1996), where the tagger was trained on the entire training corpus. In order to generate part of speech tags for the training data, we split the training set into 10 segments and tagged each segment using MXPOST trained on the other 9 segments.

### 4.1 Main results

In our final experiments, we tested eight different parsing configurations, representing all possible choices between baseline and cluster-based feature sets, first-order (Eisner, 2000) and second-order (Carreras, 2007) factorizations, and labeled and unlabeled parsing. Despite the use of labeled parsing models, in our evaluations we consider only the unlabeled dependency accuracy and ignore the labels discovered by the parser. In addition, we ignore the parent-predictions of punctuation tokens, which is the standard for English (Yamada and Matsumoto, 2003; McDonald et al., 2005a).

Table 2 compiles our final test results and also includes two results from previous work by McDonald et al. (2005a; 2006), for the purposes of comparison. It should be noted, however, that the comparison between their work and ours is not exact. For example, McDonald et al. used MIRA (Crammer and Singer, 2003; Crammer et al., 2003) to train their parsers, whereas we use the averaged perceptron. In addition, the “MD2” second-order model uses only sibling interactions, whereas our second-order models “dep2” and “dep2c” include both sibling and grandparent interactions. Finally, McDonald et al. trained unlabeled models only, whereas we trained both labeled and unlabeled models.

We can observe several trends in the results of Table 2. First, performance increases as the “order” of the parser increases: edge-factored models (dep1 and MD1) have the lowest performance, adding sibling relationships (MD2) increases performance, and adding grandparent relationships (dep2) yields even better accuracies. Similar observations have also made by Carreras (2007).

Second, the use of labels in the parsing model increases performance for all tested models. Presumably, this is because a labeled model can bin the dependency statistics according to label and achieve a more consistent and accurate distribution within each dependency label.

Finally, note that the parsers using cluster-based feature sets consistently outperform the models using the baseline features. Some of these improvements can be quite large; for example, a first-order model using cluster-based features performs as well as a second-order model using baseline features. Moreover, the benefits of cluster-based feature sets combine additively with the gains of increasing model order. For example, consider the unlabeled parsers in Table 2: increasing the model order from dep1 to dep2 results in a relative reduction in error of roughly 13%, while introducing cluster-based features from dep2 to dep2c yields an additional relative error reduction of roughly 14%. Interestingly, when cluster-based features are in use, the benefit of introducing labels into the parsing model is reduced.

### 4.2 Results for smaller training sets

We also performed experiments to evaluate the effect of cluster-based features as the amount of training data is varied. Figure 4 displays the accuracy of first- and second-order models when trained on smaller portions of the Treebank. Note that the cluster-based features obtain consistent gains regardless of the size of the training set.
It is interesting to consider the amount by which cluster-based features reduce the need for supervised data, given a desired level of accuracy. Based on Figure 4, we can extrapolate that cluster-based features reduce the need for supervised data by roughly a factor of 2. For example, the performance of the dep1c model trained on 1k sentences roughly coincides with the position of the dep1 curve at the 2k sentence mark. Similarly, the dep2c curve at the 8k sentence mark exceeds the performance of the dep2 curve at the 16k mark. This data-halving effect appears throughout the learning curves in Figure 4.

When combining the effects of model order and cluster-based features, the reductions in the amount of supervised data required are even larger. For example, the dep2c model trained on 1k sentences is only slightly below the dep1 model trained on 4k sentences, and the dep2c model trained on 4k sentences is close to the dep1 model trained on the entire training set (roughly 40k sentences).

Before we conclude this section, we should reiterate that the data-reduction factors discussed above are extrapolations based on relatively few data points. Nevertheless, the comparisons are encouraging.

5 Related Work

As mentioned earlier, our research was inspired by the success of Miller et al. (2004), who demonstrated the effectiveness of using word clusters as features in a discriminative learning approach. Our research, however, applies this approach to dependency parsing rather than named-entity recognition.

In this paper, we have focused on developing new representations for lexical information. Previous research in this area includes several models which incorporate hidden variables (Matsuzaki et al., 2005; Koo and Collins, 2005; Petrov et al., 2006; Titov and Henderson, 2007). These approaches have the advantage that the model is able to learn different usages for the hidden variables, depending on the target problem at hand. However, performing inference in the presence of hidden variables can be computationally demanding.

Semi-supervised parsing approaches have been attempted previously, most notably by McClosky et al. (2006), who leveraged a discriminative reranker on a large unsupervised corpus in order to obtain additional training data for the parser. Their method was shown to be quite effective in practice.

6 Conclusions

In this paper, we have presented a simple but effective semi-supervised learning approach and demonstrated that it achieves substantial improvement over the baseline in a broad-coverage dependency parsing task. Despite this success, there are several ways in which our approach might be improved.

To begin, recall that the Brown clustering algorithm uses the likelihood of a bigram language
model as the criterion for merging clusters. Intuitively, such clusters should be more appropriate for sequence-labeling problems than dependency parsing. A natural avenue for further research would be the development of clustering algorithms that are better suited to dependency parsing. For example, one might devise a Brown-style clustering algorithm that attempts to maximize the likelihood of the Treebank according to a probabilistic dependency parser. Alternatively, one could design clustering algorithms that cluster entire head-modifier arcs rather than individual words.

Another idea would be to integrate the clustering algorithm into the training algorithm in a limited fashion. For example, after training an initial parser, one could parse a large amount of unlabeled text and use those parses to improve the quality of the clusters. These improved clusters can then be used to retrain an improved parser, resulting in a training algorithm similar to that of McClosky et al. (2006).

Setting aside the development of new clustering algorithms, a final area for future work is the extension of our method to new domains, such as conversational text or other languages, and new NLP problems, such as machine translation. In conclusion, we look forward to continuing to develop our cluster-based approach in the future.

Acknowledgments

References

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. CoNLL-X*.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. EMNLP-CoNLL*.

E. Charniak, D. Blaheta, N. Ge, K. Hall, and M. Johnson. 2000. *BLLIP 1987-89 WSJ Corpus Release 1*. Linguistic Data Consortium.

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. Koby Crammer, Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2003. Online passive-aggressive algorithms. In *Proc. NIPS*.

J. Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In H. C. Bunt and A. Nijholt, editors, *New Developments in Natural Language Parsing*, pages 29–62. Kluwer Academic Publishers.

Y. Freund and R. Schapire. 1998. Large margin classification using the perceptron algorithm. In *Computational Learning Theory*, pages 209–217.

T. Koo and M. Collins. 2005. Hidden-variable models for discriminative reranking. In *Proc. EMNLP*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*.

W. Li and A. McCallum. 2005. Semi-supervised sequence modeling with syntactic topic models.

P. Liang. 2005. Semi-supervised learning for natural language. Master’s thesis, Massachusetts Institute of Technology.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank.

Takuya Matsuzaki, Yusuke Miyao, and Jun’ichi Tsujii. 2005. Probabilistic cfg with latent annotations. In *Proc. ACL*.

D. McClosky, E. Charniak, and M. Johnson. 2006. Effective self-training for parsing. In *Proc. NAACL*.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. EACL*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proc. ACL*.

R. McDonald, F. Pereira, K. Ribaric, and J. Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. HLT-EMNLP*.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 337–342, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Proc. EMNLP-CoNLL*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proc. COLING/ACL*.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proc. EMNLP*, pages 133–142.

Ivan Titov and James Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *Proc. ACL*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. IWPT*.