TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing

Xavier Carreras **Michael Collins** **Terry Koo**
MIT CSAIL, Cambridge, MA 02139, USA
{carreras,mcollins,maestro}@csail.mit.edu

Abstract

We describe a parsing approach that makes use of the perceptron algorithm, in conjunction with dynamic programming methods, to recover full constituent-based parse trees. The formalism allows a rich set of parse-tree features, including PCFG-based features, bigram and trigram dependency features, and surface features. A severe challenge in applying such an approach to full syntactic parsing is the efficiency of the parsing algorithms involved. We show that efficient training is feasible, using a Tree Adjoining Grammar (TAG) based parsing formalism. A lower-order dependency parsing model is used to restrict the search space of the full model, thereby making it efficient. Experiments on the Penn WSJ treebank show that the model achieves state-of-the-art performance, for both constituent and dependency accuracy.

1 Introduction

In global linear models (GLMs) for structured prediction, (e.g., (Johnson et al., 1999; Lafferty et al., 2001; Collins, 2002; Altun et al., 2003; Taskar et al., 2004)), the optimal label y^* for an input $\mathbf{x}$ is

$$y^* = \arg \max_{y \in \mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) \qquad (1)$$

where $\mathcal{Y}(\mathbf{x})$ is the set of possible labels for the input $\mathbf{x}$; $\mathbf{f}(\mathbf{x}, y) \in \mathbb{R}^d$ is a feature vector that represents the pair $(\mathbf{x}, y)$; and $\mathbf{w}$ is a parameter vector. This paper describes a GLM for natural language parsing, trained using the averaged perceptron. The parser we describe recovers full syntactic representations, similar to those derived by a probabilistic context-free grammar (PCFG). A key motivation for the use of GLMs in parsing is that they allow a great deal of flexibility in the features which can be included in the definition of $\mathbf{f}(\mathbf{x}, y)$.

A critical problem when training a GLM for parsing is the computational complexity of the inference problem. The averaged perceptron requires the training set to be repeatedly decoded under the model; under even a simple PCFG representation, finding the $\arg\max$ in Eq. 1 requires $O(n^3 G)$ time, where n is the length of the sentence, and G is a grammar constant. The average sentence length in the data set we use (the Penn WSJ treebank) is over 23 words; the grammar constant G can easily take a value of 1000 or greater. These factors make exact inference algorithms virtually intractable for training or decoding GLMs for full syntactic parsing.

As a result, in spite of the potential advantages of these methods, there has been very little previous work on applying GLMs for full parsing without the use of fairly severe restrictions or approximations. For example, the model in (Taskar et al., 2004) is trained on only sentences of 15 words or less; reranking models (Collins, 2000; Charniak and Johnson, 2005) restrict $\mathcal{Y}(\mathbf{x})$ to be a small set of parses from a first-pass parser; see section 1.1 for discussion of other related work.

The following ideas are central to our approach:

(1) A TAG-based, splittable grammar. We describe a novel, TAG-based parsing formalism that allows full constituent-based trees to be recovered. A driving motivation for our approach comes from the flexibility of the feature-vector representations $\mathbf{f}(\mathbf{x}, y)$ that can be used in the model. The formalism that we describe allows the incorporation of: (1) basic PCFG-style features; (2) the use of features that are sensitive to *bigram* dependencies between pairs of words; and (3) features that are sensitive to *trigram* dependencies. Any of these feature types can be combined with *surface features* of the sentence $\mathbf{x}$, in a similar way to the use of surface features in conditional random fields (Lafferty et al., 2001). Crucially, in spite of these relatively rich representations, the formalism can be parsed efficiently (in $O(n^4G)$ time) using dynamic-programming algorithms described by Eisner (2000) (unlike many other TAG-related approaches, our formalism is "splittable" in the sense described by Eisner, leading to more efficient parsing algorithms).

(2) Use of a lower-order model for pruning. The $O(n^4G)$ running time of the TAG parser is still too expensive for efficient training with the perceptron. We describe a method that leverages a simple, first-order dependency parser to restrict the search space of the TAG parser in training and testing. The lower-order parser runs in $O(n^3H)$ time where $H \ll G$; experiments show that it is remarkably effective in pruning the search space of the full TAG parser.

Experiments on the Penn WSJ treebank show that the model recovers constituent structures with higher accuracy than the approaches of (Charniak, 2000; Collins, 2000; Petrov and Klein, 2007), and with a similar level of performance to the reranking parser of (Charniak and Johnson, 2005). The model also recovers dependencies with significantly higher accuracy than state-of-the-art dependency parsers such as (Koo et al., 2008; McDonald and Pereira, 2006).

1.1 Related Work

Previous work has made use of various restrictions or approximations that allow efficient training of GLMs for parsing. This section describes the relationship between our work and this previous work.

In *reranking* approaches, a first-pass parser is used to enumerate a small set of candidate parses for an input sentence; the reranking model, which is a GLM, is used to select between these parses (e.g., (Ratnaparkhi et al., 1994; Johnson et al., 1999; Collins, 2000; Charniak and Johnson, 2005)). A crucial advantage of our approach is that it considers a very large set of alternatives in $\mathcal{Y}(\mathbf{x})$, and can thereby avoid search errors that may be made in the first-pass parser.[1]

Another approach that allows efficient training of GLMs is to use *simpler syntactic representations*, in particular dependency structures (McDonald et al., 2005). Dependency parsing can be implemented in $O(n^3)$ time using the algorithms of Eisner (2000). In this case there is no grammar constant, and parsing is therefore efficient. A disadvantage of these approaches is that they do not recover full, constituent-based syntactic structures; the increased linguistic detail in full syntactic structures may be useful in NLP applications, or may improve dependency parsing accuracy, as is the case in our experiments.[2]

There has been some previous work on GLM approaches for full syntactic parsing that make use of dynamic programming. Taskar et al. (2004) describe a max-margin approach; however, in this work training sentences were limited to be of 15 words or less. Clark and Curran (2004) describe a log-linear GLM for CCG parsing, trained on the Penn treebank. This method makes use of parallelization across an 18 node cluster, together with up to 25GB of memory used for storage of dynamic programming structures for training data. Clark and Curran (2007) describe a perceptron-based approach for CCG parsing which is considerably more efficient, and makes use of a supertagging model to prune the search space of the full parsing model. Recent work (Petrov et al., 2007; Finkel et al., 2008) describes log-linear GLMs applied to PCFG representations, but does not make use of dependency features.

2 The TAG-Based Parsing Model

2.1 Derivations

This section describes the idea of *derivations* in our parsing formalism. As in context-free grammars or TAGs, a derivation in our approach is a data structure that specifies the sequence of operations used in combining basic (elementary) structures in a grammar, to form a full parse tree. The parsing formalism we use is related to the tree adjoining grammar (TAG) formalisms described in (Chiang, 2003; Shen and Joshi, 2005). However, an important difference of our work from this previous work is that our formalism is defined to be "splittable", allowing use of the efficient parsing algorithms of Eisner (2000).

A derivation in our model is a pair $\langle E, D \rangle$ where E is a set of *spines*, and D is a set of *dependencies*

[1]Some features used within reranking approaches may be difficult to incorporate within dynamic programming, but it is nevertheless useful to make use of GLMs in the dynamic-programming stage of parsing. Our parser could, of course, be used as the first-stage parser in a reranking approach.

[2]Note however that the lower-order parser that we use to restrict the search space of the TAG-based parser is based on the work of McDonald et al. (2005). See also (Sagae et al., 2007) for a method that uses a dependency parser to restrict the search space of a more complex HPSG parser.

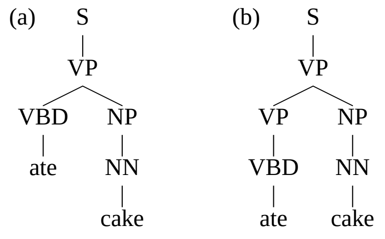
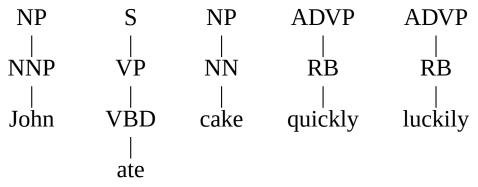


Figure 1: Two example trees.

specifying how the spines are combined to form a parse tree. The spines are similar to elementary trees in TAG. Some examples are as follows:



These structures do not have substitution nodes, as is common in TAGs.[3] Instead, the spines consist of a lexical anchor together with a series of unary projections, which usually correspond to different X-bar levels associated with the anchor.

The operations used to combine spines are similar to the TAG operations of adjunction and sister adjunction. We will call these operations *regular adjunction* (r-adjunction) and *sister adjunction* (s-adjunction). As one example, the *cake* spine shown above can be s-adjoined into the VP node of the *ate* spine, to form the tree shown in figure 1(a). In contrast, if we use the r-adjunction operation to adjoin the *cake* tree into the VP node, we get a different structure, which has an additional VP level created by the r-adjunction operation: the resulting tree is shown in figure 1(b). The r-adjunction operation is similar to the usual adjunction operation in TAGs, but has some differences that allow our grammars to be splittable; see section 2.3 for more discussion.

We now give formal definitions of the sets E and D. Take $\mathbf{x}$ to be a sentence consisting of $n+1$ words, $x_0 \dots x_n$, where x_0 is a special *root* symbol, which we will denote as $*$. A derivation for the input sentence $\mathbf{x}$ consists of a pair $\langle E, D\rangle$, where:

• E is a set of $(n+1)$ tuples of the form $\langle i, \eta\rangle$, where $i \in \{0 \dots n\}$ is an index of a word in the sentence, and η is the spine associated with the word x_i. The set E specifies one spine for each of the $(n+1)$ words in the sentence. Where it is clear from context, we will use η_i to refer to the spine in E corresponding to the i'th word.

• D is a set of n dependencies. Each dependency is a tuple $\langle h, m, l\rangle$. Here h is the index of the *head-word* of the dependency, corresponding to the spine η_h which contains a node that is being adjoined into. m is the index of the *modifier-word* of the dependency, corresponding to the spine η_m which is being adjoined into η_h. l is a *label*.

The label l is a tuple $\langle \mathtt{POS}, \mathtt{A}, \eta_h, \eta_m, \mathtt{L}\rangle$. η_h and η_m are the head and modifier spines that are being combined. POS specifies which node in η_h is being adjoined into. A is a binary flag specifying whether the combination operation being used is s-adjunction or r-adjunction. L is a binary flag specifying whether or not any "previous" modifier has been r-adjoined into the position POS in η_h. By a previous modifier, we mean a modifier m' that was adjoined from the same direction as m (i.e., such that $h < m' < m$ or $m < m' < h$).

It would be sufficient to define l to be the pair $\langle \mathtt{POS}, \mathtt{A}\rangle$—the inclusion of η_h, η_m and L adds redundant information that can be recovered from the set E, and other dependencies in D—but it will be convenient to include this information in the label. In particular, it is important that given this definition of l, it is possible to define a function $\mathtt{GRM}(l)$ that maps a label l to a triple of nonterminals that represents the grammatical relation between m and h in the dependency structure. For example, in the tree shown in figure 1(a), the grammatical relation between *cake* and *ate* is the triple $\mathtt{GRM}(l) = \langle \mathsf{VP\ VBD\ NP}\rangle$. In the tree shown in figure 1(b), the grammatical relation between *cake* and *ate* is the triple $\mathtt{GRM}(l) = \langle \mathsf{VP\ VP\ NP}\rangle$.

The conditions under which a pair $\langle E, D\rangle$ forms a valid derivation for a sentence $\mathbf{x}$ are similar to those in conventional LTAGs. Each $\langle i, \eta\rangle \in E$ must be such that η is an elementary tree whose anchor is the word x_i. The dependencies D must form a directed, projective tree spanning words $0 \dots n$, with $*$ at the root of this tree, as is also the case in previous work on discriminative approches to dependency parsing (McDonald et al., 2005). We allow any modifier tree η_m to adjoin into any position in any head tree η_h, but the dependencies D must nevertheless be coherent—for example they must be consistent with the spines in E, and they must be nested correctly.[4] We will al-

[3]It would be straightforward to extend the approach to include substitution nodes, and a substitution operation.

[4]For example, closer modifiers to a particular head must adjoin in at the same or a lower spine position than modifiers

low multiple modifier spines to s-adjoin or r-adjoin into the same node in a head spine; see section 2.3 for more details.

2.2 A Global Linear Model

The model used for parsing with this approach is a global linear model. For a given sentence $\mathbf{x}$, we define $\mathcal{Y}(\mathbf{x})$ to be the set of valid derivations for $\mathbf{x}$, where each $y \in \mathcal{Y}(\mathbf{x})$ is a pair $\langle E, D\rangle$ as described in the previous section. A function $\mathbf{f}$ maps $(\mathbf{x}, y)$ pairs to feature-vectors $\mathbf{f}(\mathbf{x}, y) \in \mathbb{R}^d$. The parameter vector $\mathbf{w}$ is also a vector in $\mathbb{R}^d$. Given these definitions, the optimal derivation for an input sentence $\mathbf{x}$ is $y^* = \arg\max_{y\in\mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y)$.

We now come to how the feature-vector $\mathbf{f}(\mathbf{x}, y)$ is defined in our approach. A simple "first-order" model would define

$$\mathbf{f}(\mathbf{x}, y) = \sum_{\langle i,\eta\rangle \in E(y)} \mathbf{e}(\mathbf{x}, \langle i, \eta\rangle) + \sum_{\langle h,m,l\rangle \in D(y)} \mathbf{d}(\mathbf{x}, \langle h, m, l\rangle) \quad (2)$$

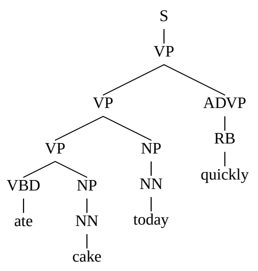
Here we use $E(y)$ and $D(y)$ to respectively refer to the set of spines and dependencies in y. The function $\mathbf{e}$ maps a sentence $\mathbf{x}$ paired with a spine $\langle i, \eta\rangle$ to a feature vector. The function $\mathbf{d}$ maps dependencies within y to feature vectors. This decomposition is similar to the first-order model of McDonald et al. (2005), but with the addition of the $\mathbf{e}$ features.

We will extend our model to include higher-order features, in particular features based on *sibling* dependencies (McDonald and Pereira, 2006), and *grandparent* dependencies, as in (Carreras, 2007). If $y = \langle E, D\rangle$ is a derivation, then:

- $S(y)$ is a set of sibling dependencies. Each sibling dependency is a tuple $\langle h, m, l, s\rangle$. For each $\langle h, m, l, s\rangle \in S$ the tuple $\langle h, m, l\rangle$ is an element of D; there is one member of S for each member of D. The index s is the index of the word that was adjoined to the spine for h immediately before m (or the NULL symbol if no previous adjunction has taken place).

- $G(y)$ is a set of grandparent dependencies of type 1. Each type 1 grandparent dependency is a tuple $\langle h, m, l, g\rangle$. There is one member of G for every member of D. The additional information, the index g, is the index of the word that is the first modifier to the *right* of the spine for m.

that are further from the head.

(a) 

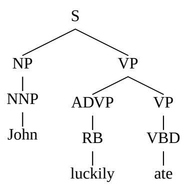
(b) 

Figure 2: Two Example Trees

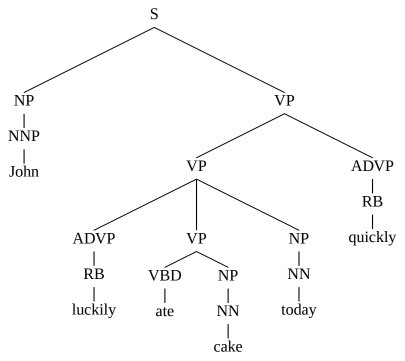


Figure 3: An example tree, formed by a combination of the two structures in figure 2.

- $Q(y)$ is an additional set of grandparent dependencies, of type 2. Each of these dependencies is a tuple $\langle h, m, l, q\rangle$. Again, there is one member of Q for every member of D. The additional information, the index q, is the index of the word that is the first modifier to the *left* of the spine for m.

The feature-vector definition then becomes:

$$\mathbf{f}(\mathbf{x}, y) = \sum_{\langle i,\eta\rangle \in E(y)} \mathbf{e}(\mathbf{x}, \langle i, \eta\rangle) + \sum_{\langle h,m,l\rangle \in D(y)} \mathbf{d}(\mathbf{x}, \langle h, m, l\rangle) + \sum_{\langle h,m,l,s\rangle \in S(y)} \mathbf{s}(\mathbf{x}, \langle h, m, l, s\rangle) + \sum_{\langle h,m,l,g\rangle \in G(y)} \mathbf{g}(\mathbf{x}, \langle h, m, l, g\rangle) + \sum_{\langle h,m,l,q\rangle \in Q(y)} \mathbf{q}(\mathbf{x}, \langle h, m, l, q\rangle) \quad (3)$$

where $\mathbf{s}$, $\mathbf{g}$ and $\mathbf{q}$ are feature vectors corresponding to the new, higher-order elements.[5]

2.3 Recovering Parse Trees from Derivations

As in TAG approaches, there is a mapping from derivations $\langle E, D\rangle$ to parse trees (i.e., the type of trees generated by a context-free grammar). In our case, we map a spine and its dependencies to a constituent structure by first handling the dependen-

[5]We also added constituent-boundary features to the model, which is a simple change that led to small improvements on validation data; for brevity we omit the details.

cies on each side separately and then combining the left and right sides.

First, it is straightforward to build the constituent structure resulting from multiple adjunctions on the same side of a spine. As one example, the structure in figure 2(a) is formed by first s-adjoining the spine with anchor *cake* into the VP node of the spine for *ate*, then r-adjoining spines anchored by *today* and *quickly* into the same node, where all three modifier words are to the right of the head word. Notice that each r-adjunction operation creates a new VP level in the tree, whereas s-adjunctions do not create a new level. Now consider a tree formed by first r-adjoining a spine for *luckily* into the VP node for *ate*, followed by s-adjoining the spine for *John* into the S node, in both cases where the modifiers are to the left of the head. In this case the structure that would be formed is shown in figure 2(b).

Next, consider combining the left and right structures of a spine. The main issue is how to handle multiple r-adjunctions or s-adjunctions on both sides of a node in a spine, because our derivations do not specify how adjunctions from different sides embed with each other. In our approach, the combination operation preserves the height of the different modifiers from the left and right directions. To illustrate this, figure 3 shows the result of combining the two structures in figure 2. The combination of the left and right modifier structures has led to flat structures, for example the rule VP $\rightarrow$ ADVP VP NP in the above tree.

Note that our r-adjunction operation is different from the usual adjunction operation in TAGs, in that "wrapping" adjunctions are not possible, and r-adjunctions from the left and right directions are independent from each other; because of this our grammars are splittable.

3 Parsing Algorithms

3.1 Use of Eisner's Algorithms

This section describes the algorithm for finding $y^* = \arg\max_{y \in \mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y)$ where $\mathbf{f}(\mathbf{x}, y)$ is defined through either the first-order model (Eq. 2) or the second-order model (Eq. 3).

For the first-order model, the methods described in (Eisner, 2000) can be used for the parsing algorithm. In Eisner's algorithms for dependency parsing each word in the input has left and right finite-state (weighted) automata, which generate the left and right modifiers of the word in question. We make use of this idea of automata, and also make direct use of the method described in section 4.2 of (Eisner, 2000) that allows a set of possible senses for each word in the input string. In our use of the algorithm, each possible sense for a word corresponds to a different possible spine that can be associated with that word. The left and right automata are used to keep track of the last position in the spine that was adjoined into on the left/right of the head respectively. We can make use of separate left and right automata—i.e., the grammar is splittable—because left and right modifiers are adjoined independently of each other in the tree. The extension of Eisner's algorithm to the second-order model is similar to the algorithm described in (Carreras, 2007), but again with explicit use of word senses and left/right automata. The resulting algorithms run in $O(Gn^3)$ and $O(Hn^4)$ time for the first-order and second-order models respectively, where G and H are grammar constants.

3.2 Efficient Parsing

The efficiency of the parsing algorithm is important in applying the parsing model to test sentences, and also when training the model using discriminative methods. The grammar constants G and H introduced in the previous section are polynomial in factors such as the number of possible spines in the model, and the number of possible states in the finite-state automata implicit in the parsing algorithm. These constants are large, making exhaustive parsing very expensive.

To deal with this problem, we use a simple initial model to prune the search space of the more complex model. The first-stage model we use is a first-order dependency model, with labeled dependencies, as described in (McDonald et al., 2005). As described shortly, we will use this model to compute *marginal* scores for dependencies in both training and test sentences. A marginal score $\mu(\mathbf{x}, h, m, l)$ is a value between 0 and 1 that reflects the plausibility of a dependency for sentence $\mathbf{x}$ with head-word x_h, modifier word x_m, and label l. In the first-stage pruning model the labels l are triples of non-terminals representing grammatical relations, as described in section 2.1 of this paper—for example, one possible label would be $\langle$VP VBD NP$\rangle$, and in general any triple of non-terminals is possible.

Given a sentence $\mathbf{x}$, and an index m of a word in that sentence, we define DMAX$(\mathbf{x}, m)$ to be the highest scoring dependency with m as a modifier:

$$\mathtt{DMAX}(\mathbf{x}, m) = \max_{h,l} \mu(\mathbf{x}, h, m, l)$$

For a sentence $\mathbf{x}$, we then define the set of allowable dependencies to be

$$\pi(\mathbf{x}) = \{\langle h, m, l\rangle \ : \ \mu(\mathbf{x}, h, m, l) \geq \alpha \mathtt{DMAX}(\mathbf{x}, m)\}$$

where α is a constant dictating the beam size that is used (in our experiments we used $\alpha = 10^{-6}$).

The set $\pi(\mathbf{x})$ is used to restrict the set of possible parses under the full TAG-based model. In section 2.1 we described how the TAG model has dependency labels of the form $\langle \mathtt{POS}, \mathtt{A}, \eta_h, \eta_m, \mathtt{L}\rangle$, and that there is a function GRM that maps labels of this form to triples of non-terminals. The basic idea of the pruned search is to only allow dependencies of the form $\langle h, m, \langle \mathtt{POS}, \mathtt{A}, \eta_h, \eta_m, \mathtt{L}\rangle\rangle$ if the tuple $\langle h, m, \mathtt{GRM}(\langle \mathtt{POS}, \mathtt{A}, \eta_h, \eta_m, \mathtt{L}\rangle)\rangle$ is a member of $\pi(\mathbf{x})$, thus reducing the search space for the parser.

We now turn to how the marginals $\mu(\mathbf{x}, h, m, l)$ are defined and computed. A simple approach would be to use a conditional log-linear model (Lafferty et al., 2001), with features as defined by McDonald et al. (2005), to define a distribution $P(y|\mathbf{x})$ where the parse structures y are dependency structures with labels that are triples of non-terminals. In this case we could define

$$\mu(\mathbf{x}, h, m, l) = \sum_{y:(h,m,l)\in y} P(y|\mathbf{x})$$

which can be computed with inside-outside style algorithms, applied to the data structures from (Eisner, 2000). The complexity of training and applying such a model is again $O(Gn^3)$, where G is the number of possible labels, and the number of possible labels (triples of non-terminals) is around $G = 1000$ in the case of treebank parsing; this value for G is still too large for the method to be efficient. Instead, we train three separate models μ_1, μ_2, and μ_3 for the three different positions in the non-terminal triples. We then take $\mu(\mathbf{x}, h, m, l)$ to be a product of these three models, for example we would calculate

$$\begin{aligned}\mu(\mathbf{x}, h, m, \langle \mathsf{VP\ VBD\ NP}\rangle) \;\; &= \\ \mu_1(\mathbf{x}, h, m, \langle \mathsf{VP}\rangle) &\times \mu_2(\mathbf{x}, h, m, \langle \mathsf{VBD}\rangle) \\ &\times \mu_3(\mathbf{x}, h, m, \langle \mathsf{NP}\rangle)\end{aligned}$$

Training the three models, and calculating the marginals, now has a grammar constant equal to the number of non-terminals in the grammar, which is far more manageable. We use the algorithm described in (Globerson et al., 2007) to train the conditional log-linear model; this method was found to converge to a good model after 10 iterations over the training data.

4 Implementation Details

4.1 Features

Section 2.2 described the use of feature vectors associated with spines used in a derivation, together with first-order, sibling, and grandparent dependencies. The dependency features used in our experiments are closely related to the features described in (Carreras, 2007), which are an extension of the McDonald and Pereira (2006) features to cover grandparent dependencies in addition to first-order and sibling dependencies. The features take into account the identity of the labels l used in the derivations. The features could potentially look at any information in the labels, which are of the form $\langle \mathtt{POS}, \mathtt{A}, \eta_h, \eta_m, \mathtt{L}\rangle$, but in our experiments, we map labels to a pair $(\mathtt{GRM}(\langle \mathtt{POS}, \mathtt{A}, \eta_h, \eta_m, \mathtt{L}\rangle), \mathtt{A})$. Thus the label features are sensitive only to the triple of non-terminals corresponding to the grammatical relation involved in an adjunction, and a binary flag specifiying whether the operation is s-adjunction or r-adjunction.

For the spine features $\mathbf{e}(\mathbf{x}, \langle i, \eta\rangle)$, we use feature templates that are sensitive to the identity of the spine η, together with contextual features of the string $\mathbf{x}$. These features consider the identity of the words and part-of-speech tags in a window that is centered on x_i and spans the range $x_{(i-2)} \ldots x_{(i+2)}$.

4.2 Extracting Derivations from Parse Trees

In the experiments in this paper, the following three-step process was used: (1) derivations were extracted from a training set drawn from the Penn WSJ treebank, and then used to train a parsing model; (2) the test data was parsed using the resulting model, giving a derivation for each test data sentence; (3) the resulting test-data derivations were mapped back to Penn-treebank style trees, using the method described in section 2.1. To achieve step (1), we first apply a set of head-finding rules which are similar to those described in (Collins, 1997). Once the head-finding rules have been applied, it is straightforward to extract

	precision	recall	F_1
PPK07	–	–	88.3
FKM08	88.2	87.8	88.0
CH2000	89.5	89.6	89.6
CO2000	89.9	89.6	89.8
PK07	90.2	89.9	90.1
this paper	**91.4**	**90.7**	**91.1**
CJ05	–	–	91.4
H08	–	–	91.7
CO2000(s24)	89.6	88.6	89.1
this paper (s24)	**91.1**	**89.9**	**90.5**

Table 1: Results for different methods. PPK07, FKM08, CH2000, CO2000, PK07, CJ05 and H08 are results on section 23 of the Penn WSJ treebank, for the models of Petrov et al. (2007), Finkel et al. (2008), Charniak (2000), Collins (2000), Petrov and Klein (2007), Charniak and Johnson (2005), and Huang (2008). (CJ05 is the performance of an updated model at http://www.cog.brown.edu/mj/software.htm.) "s24" denotes results on section 24 of the treebank.

	s23	s24
KCC08 unlabeled	92.0	91.0
KCC08 labeled	92.5	91.7
this paper	**93.5**	**92.5**

Table 2: Table showing unlabeled dependency accuracy for sections 23 and 24 of the treebank, using the method of (Yamada and Matsumoto, 2003) to extract dependencies from parse trees from our model. KCC08 unlabeled is from (Koo et al., 2008), a model that has previously been shown to have higher accuracy than (McDonald and Pereira, 2006). KCC08 is the labeled dependency parser from (Koo et al., 2008); here we only evaluate the unlabeled accuracy.

derivations from the Penn treebank trees.

Note that the mapping from parse trees to derivations is many-to-one: for example, the example trees in section 2.1 have structures that are as "flat" (have as few levels) as is possible, given the set D that is involved. Other similar trees, but with more VP levels, will give the same set D. However, this issue appears to be benign in the Penn WSJ treebank. For example, on section 22 of the treebank, if derivations are first extracted using the method described in this section, then mapped back to parse trees using the method described in section 2.1, the resulting parse trees score 100% precision and 99.81% recall in labeled constituent accuracy, indicating that very little information is lost in this process.

4.3 Part-of-Speech Tags, and Spines

Sentences in training, test, and development data are assumed to have part-of-speech (POS) tags. POS tags are used for two purposes: (1) in the features described above; and (2) to limit the set of allowable spines for each word during parsing. Specifically, for each POS tag we create a separate

	1st stage		2nd stage		
α	active	coverage	oracle F_1	speed	F_1
10^{-4}	0.07	97.7	97.0	5:15	91.1
10^{-5}	0.16	98.5	97.9	11:45	91.6
10^{-6}	0.34	99.0	98.5	21:50	92.0

Table 3: Effect of the beam size, controlled by α, on the performance of the parser on the development set (1,699 sentences). In each case α refers to the beam size used in both training and testing the model. "active": percentage of dependencies that remain in the beam out of the total number of labeled dependencies (1,000 triple labels times 1,138,167 unlabeled dependencies); "coverage": percentage of correct dependencies in the beam out of the total number of correct dependencies. "oracle F_1": maximum achievable score of constituents, given the beam. "speed": parsing time in *min:sec* for the TAG-based model (this figure does not include the time taken to calculate the marginals using the lower-order model); "F_1": score of predicted constituents.

dictionary listing the spines that have been seen with this POS tag in training data; during parsing we only allow spines that are compatible with this dictionary. (For test or development data, we used the part-of-speech tags generated by the parser of (Collins, 1997). Future work should consider incorporating the tagging step within the model; it is not challenging to extend the model in this way.)

5 Experiments

Sections 2-21 of the Penn Wall Street Journal treebank were used as training data in our experiments, and section 22 was used as a development set. Sections 23 and 24 were used as test sets. The model was trained for 20 epochs with the averaged perceptron algorithm, with the development data performance being used to choose the best epoch. Table 1 shows the results for the method.

Our experiments show an improvement in performance over the results in (Collins, 2000; Charniak, 2000). We would argue that the Collins (2000) method is considerably more complex than ours, requiring a first-stage generative model, together with a reranking approach. The Charniak (2000) model is also arguably more complex, again using a carefully constructed generative model. The accuracy of our approach also shows some improvement over results in (Petrov and Klein, 2007). This work makes use of a PCFG with latent variables that is trained using a split/merge procedure together with the EM algorithm. This work is in many ways complementary to ours—for example, it does not make use of GLMs, dependency features, or of representations that go beyond PCFG productions—and some combination of the two methods may give further gains.

Charniak and Johnson (2005), and Huang (2008), describe approaches that make use of non-local features in conjunction with the Charniak (2000) model; future work may consider extending our approach to include non-local features. Finally, other recent work (Petrov et al., 2007; Finkel et al., 2008) has had a similar goal of scaling GLMs to full syntactic parsing. These models make use of PCFG representations, but do not explicitly model bigram or trigram dependencies. The results in this work (88.3%/88.0% F1) are lower than our F1 score of 91.1%; this is evidence of the benefits of the richer representations enabled by our approach.

Table 2 shows the accuracy of the model in recovering unlabeled dependencies. The method shows improvements over the method described in (Koo et al., 2008), which is a state-of-the-art second-order dependency parser similar to that of (McDonald and Pereira, 2006), suggesting that the incorporation of constituent structure can improve dependency accuracy.

Table 3 shows the effect of the beam-size on the accuracy and speed of the parser on the development set. With the beam setting used in our experiments ($\alpha = 10^{-6}$), only 0.34% of possible dependencies are considered by the TAG-based model, but 99% of all correct dependencies are included. At this beam size the best possible F_1 constituent score is 98.5. Tighter beams lead to faster parsing times, with slight drops in accuracy.

6 Conclusions

We have described an efficient and accurate parser for constituent parsing. A key to the approach has been to use a splittable grammar that allows efficient dynamic programming algorithms, in combination with pruning using a lower-order model. The method allows relatively easy incorporation of features; future work should leverage this in producing more accurate parsers, and in applying the parser to different languages or domains.

Acknowledgments X. Carreras was supported by the Catalan Ministry of Innovation, Universities and Enterprise, by the GALE program of DARPA, Contract No. HR0011-06-C-0022, and by a grant from NTT, Agmt. Dtd. 6/21/1998. T. Koo was funded by NSF grant IIS-0415030. M. Collins was funded by NSF grant IIS-0347631 and DARPA contract No. HR0011-06-C-0022. Thanks to Jenny Rose Finkel for suggesting that we evaluate dependency parsing accuracies.

References

Altun, Y., I. Tsochantaridis, and T. Hofmann. 2003. Hidden markov support vector machines. In *ICML*.

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

Charniak, E. and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. ACL*.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proc. NAACL*.

Chiang, D. 2003. Statistical parsing with an automatically extracted tree adjoining grammar. In Bod, R., R. Scha, and K. Sima'an, editors, *Data Oriented Parsing*, pages 299–316. CSLI Publications.

Clark, S. and J. R. Curran. 2004. Parsing the wsj using ccg and log-linear models. In *Proc. ACL*.

Clark, Stephen and James R. Curran. 2007. Perceptron training for a wide-coverage lexicalized-grammar parser. In *Proc. ACL Workshop on Deep Linguistic Processing*.

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. ACL*.

Collins, M. 2000. Discriminative reranking for natural language parsing. In *Proc. ICML*.

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

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

Finkel, J. R., A. Kleeman, and C. D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proc. ACL/HLT*.

Globerson, A., T. Koo, X. Carreras, and M. Collins. 2007. Exponentiated gradient algorithms for log-linear structured prediction. In *Proc. ICML*.

Huang, L. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. ACL/HLT*.

Johnson, M., S. Geman, S. Canon, Z. Chi, and S. Riezler. 1999. Estimators for stochastic unification-based grammars. In *Proc. ACL*.

Koo, Terry, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proc. ACL/HLT*.

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

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

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

Petrov, S. and D. Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. of HLT-NAACL*.

Petrov, S., A. Pauls, and D. Klein. 2007. Discriminative log-linear grammars with latent variables. In *Proc. NIPS*.

Ratnaparkhi, A., S. Roukos, and R. Ward. 1994. A maximum entropy model for parsing. In *Proc. ICSLP*.

Sagae, Kenji, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Hpsg parsing with shallow dependency constraints. In *Proc. ACL*, pages 624–631.

Shen, L. and A.K. Joshi. 2005. Incremental ltag parsing. In *Proc HLT-EMNLP*.

Taskar, B., D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proceedings of the EMNLP-2004*.

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