University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS) | Department of Computer & Information Science

January 2006

Spanning Tree Methods for Discriminative Training of Dependency Parsers

Ryan McDonald
University of Pennsylvania

Koby Crammer
University of Pennsylvania

Fernando C.N. Pereira
University of Pennsylvania, pereira@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
McDonald, Ryan; Crammer, Koby; and Pereira, Fernando C.N., "Spanning Tree Methods for Discriminative Training of Dependency Parsers" (2006). *Technical Reports (CIS)*. Paper 55.
http://repository.upenn.edu/cis_reports/55

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-06-11.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/55
For more information, please contact repository@pobox.upenn.edu.

Spanning Tree Methods for Discriminative Training of Dependency Parsers

Abstract
Untyped dependency parsing can be viewed as the problem of finding maximum spanning trees (MSTs) in directed graphs. Using this representation, the Eisner (1996) parsing algorithm is sufficient for searching the space of projective trees. More importantly, the representation is extended naturally to non-projective parsing using Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) MST algorithm. These efficient parse search methods support large-margin discriminative training methods for learning dependency parsers. We evaluate these methods experimentally on the English and Czech treebanks.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-06-11.

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/55

Spanning Tree Methods for Discriminative Training of Dependency Parsers

Ryan McDonald **Koby Crammer** **Fernando Pereira**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, 19107
{ryantm,crammer,pereira}@cis.upenn.edu

Abstract

Untyped dependency parsing can be viewed as the problem of finding maximum spanning trees (MSTs) in directed graphs. Using this representation, the Eisner (1996) parsing algorithm is sufficient for searching the space of projective trees. More importantly, the representation is extended naturally to non-projective parsing using Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) MST algorithm. These efficient parse search methods support large-margin discriminative training methods for learning dependency parsers. We evaluate these methods experimentally on the English and Czech treebanks.

1 Introduction

Dependency parsing has seen a surge of interest lately. In particular, applications such as relation extraction (Culotta and Sorensen, 2004), machine translation (Ding and Palmer, 2005), synonym generation (Shinyama et al., 2002) and lexical resource augmentation (Snow et al., 2004) have all exploited dependency representations. A primary reason for using dependency trees over more informative lexicalized phrase structures is that they are simpler and thus more efficient to learn and parse while still encoding much of the predicate-argument information needed in such applications.

Dependency trees represent words and their arguments through directed edges in the tree and have a long history (Hudson, 1984). Figure 1 shows a dependency tree for the sentence, *John hit the ball with the bat*. A dependency tree must satisfy the tree constraint: each word must have exactly one incoming edge. We augment each sentence with a dummy root symbol that serves as the root of any dependency analysis of the sentence.

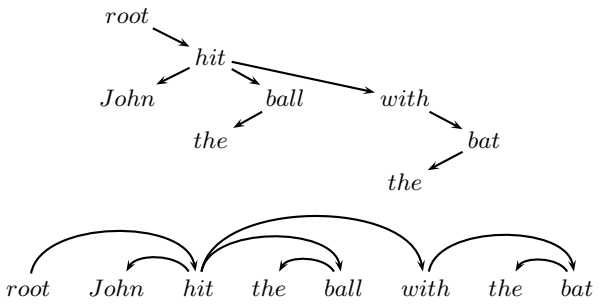


Figure 1: An example dependency tree.

The tree in Figure 1 tree can be seen as a special class of dependencies that only contain projective (also known as nested or non-crossing) edges. Assuming a unique root as the first word in the sentence, a projective tree is one that can be written with all words in a predefined linear order and all edges drawn on the plane with none crossing. Figure 1 shows this construction for the example sentence. Equivalently, we can say a tree is projective if and only if an edge from word w to word u implies that w is an ancestor of all words between w and u.

In English, projective trees are sufficient to analyze most sentence types. In fact, the largest source of English dependency trees is automatically generated from the Penn treebank (Marcus et al., 1993)

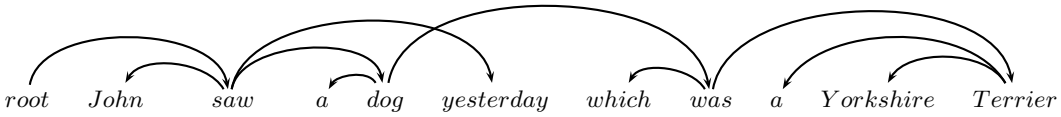


Figure 2: A non-projective dependency tree.

and is by convention exclusively projective. However, there are certain examples in which a non-projective tree is preferable. Consider the sentence, *John saw a dog yesterday which was a Yorkshire Terrier*. Here the relative clause *which was a Yorkshire Terrier* and the object it modifies (the *dog*) are separated by a temporal modifier of the main verb. There is no way to draw the dependency tree for this sentence in the plane with no crossing edges, this is illustrated in Figure 2.

In languages with more flexible word order than English, such as German, Dutch and Czech, non-projective dependencies are more frequent. Rich inflection systems reduce the demands on word order for expressing grammatical relations, leading to non-projective dependencies that we need to represent and parse efficiently.

The trees in Figure 1 and Figure 2 are untyped, that is, edges are not partitioned into types representing additional syntactic information such as grammatical function. However, typed edges are often useful. For instance, the dependency from *hit* to *John* could by typed *NP-SBJ* to indicate that *John* heads a noun phrase that is the subject of the clause headed by *hit*. We study untyped dependency trees mainly, but edge types can be added with simple extensions to the models we present here.

In this work we represent dependency parsing as search for a maximum spanning tree in a directed graph. This representation applies to both the projective and non-projective cases, and the Eisner (1996) and the Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) algorithms allow us to search the space of spanning trees efficiently during both training and testing.

We then present a model for learning dependency structures based on online large-margin discriminative training techniques. Specifically, we look at variants of the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2003) that are extensions to structured outputs such as trees. We show that online learning algorithms allow us to learn complex structures efficiently and with high accuracy. We validate our methods on Czech and English and show that our model achieves state-of-the-art performance for both languages without any language-specific enhancements.

Section 2 describes an edge based factorization of dependency trees and uses it to equate dependency parsing to the problem of finding maximum spanning trees in directed graphs. Section 3 outlines the online large-margin learning framework used to train our dependency parsers. Finally we present results in Section 5 for English and Czech.

1.1 Previous Work

Most recent work on training parsers from annotated data has focused on models and training algorithms for phrase-structure parsing. The best phrase-structure parsing models represent generatively the joint probability $P(\boldsymbol{x}, \boldsymbol{y})$ of sentence $\boldsymbol{x}$ having the structure $\boldsymbol{y}$ (Collins, 1999; Charniak, 2000). These models are easy to train because all of their parameters are simple functions of counts of parsing events in the training set. However, they achieve that simplicity by making drastic conditional independence assumptions, and training does not optimize a criterion directly related to parsing accuracy. Therefore, we might expect better performance from discriminatively trained models, as has been shown for other tasks like document classification (Joachims, 2002) and shallow parsing (Sha and Pereira, 2003). Ratnaparkhi's conditional maximum entropy model (Ratnaparkhi, 1999), trained to maximize conditional likelihood $P(\boldsymbol{y}|\boldsymbol{x})$ of the training data, performed nearly as well as generative models of the same vintage even though it scores individual parsing decisions in isolation and as a result it may suffer from the label bias problem (Lafferty et al., 2001).

Only recently has any work been done on discriminatively trained parsing models that score entire structures $\boldsymbol{y}$ for a given sentence $\boldsymbol{x}$ rather than just individual parsing decisions (Riezler et al., 2002; Clark and Curran, 2004; Collins and Roark, 2004; Taskar et al., 2004). The most likely reason for this is that discriminative training requires repeatedly reparsing the training corpus with the current model to determine the parameter updates that will improve the training criterion. This general description applies equally for extensions to parsing of standard discriminative training techniques such as maximum entropy (Berger et al., 1996), the perceptron algorithm (Rosenblatt, 1958), or support vector machines (Boser et al., 1992), which we call here *linear parsing models* because they all score a parse $\boldsymbol{y}$ as a weighted sum of parse features, $\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, \boldsymbol{y})$. The reparsing cost is already quite high for simple context-free models with $O(n^3)$ parsing complexity, but it becomes prohibitive for lexicalized grammars with $O(n^5)$ parsing complexity.

Our approach is most related to those of Collins and Roark (2004) and Taskar et al. (2004) for phrase structure parsing. Collins and Roark (2004) presented a broad coverage linear parsing model trained with the averaged perceptron algorithm. However, in order to use parse features with sufficient history, the parsing algorithm must prune away heuristically most possible parses. Taskar et al. (2004) formulate the parsing problem in the large margin structured classification setting (Taskar et al., 2003), but are limited to parsing sentences of 15 words or less due to computation time. Though these approaches represent good first steps towards discriminatively-trained parsers, they have not yet been able to display the benefits of discriminative training that have been seen in information extraction and shallow parsing.

The following work on dependency parsing is most relevant to our research. Eisner (1996) gave a generative model with a cubic parsing algorithm based on an edge factorization of trees. Yamada and Matsumoto (2003) trained support vector machines (SVM) to make parsing decisions in a shift-reduce dependency parser. As in Ratnaparkhi's parser, the classifiers are trained on individual decisions rather than on the overall quality of the parse. Nivre and Scholz (2004) developed a history-based learning model. Their parser uses a hybrid bottom-up/top-down linear-time heuristic parser and has the ability to label edges with semantic types. The accuracy of their parser is lower than that of Yamada and Matsumoto (2003).

One interesting class of dependency parsers are those that provide types on edges. A well known parser in this class is the link-grammar system of Sleator and Temperley (1993). Nivre and Scholz (2004) provide two systems, one a pure dependency parser and the other a typed model that labels edges with syntactic categories. Wang and Harper (2004) provide a rich dependency model with complex edge types containing an abundant amount of lexical and syntactic information drawn from a treebank. Though we focus primarily on pure (or untyped) dependency trees, simple extensions to the models we describe here allow for the inclusion of types.

Previous attempts at broad coverage dependency parsing have primarily dealt with projective constructions. In particular, the supervised approaches of Yamada and Matsumoto (2003) and Nivre and Scholz (2004) have provided the best results for pure dependency parsers for projective trees. Another source of dependency parsers come from lexicalized phrase-structure parsers with the ability to output dependency information (Collins, 1999; Charniak, 2000; Yamada and Matsumoto, 2003). However, since these systems are based on finding phrase structure through nested chart parsing algorithms, they cannot model projective edges tractably. However, Yamada and Matsumoto (2003) showed that these models are still very powerful since they consider much more information when making decisions then pure dependency parsers.

For non-projective dependency parsing, Nivre and Nilsson (2005) presented a parsing model that allows for the introduction of non-projective edges into dependency trees through learned edge transformations within their memory-based parser. They test this system on Czech and show an improvement over a pure projective parser. Another broad coverage non-projective parser is that of Wang and Harper (2004) for English, which presents very good results using a constraint dependency grammar framework that is rich in lexical and syntactic information.

The present work is closely related to that of

Hirakawa (2001) who, like us, relates the problem of dependency parsing to finding spanning trees for Japanese text. However, that parsing algorithm uses branch and bound techniques and is still in the worst case exponential (though in practice it seems tractable). Furthermore, no justification was provided for the empirical adequacy of equating spanning trees with dependency trees.

As this report was being completed, the work of Ribarov (2004) was brought to our attention. In this work, Ribarov also equates the problem of dependency parsing to finding maximum spanning trees in directed graphs. Furthermore, the learning model employed is the perceptron algorithm (Rosenblatt, 1958), which is also an online learning technique. However, his empirical evaluation on the Prague Dependency Treebank (Hajič, 1998) results in an accuracy well below the state of the art. This is most likely due to a very impoverished feature representation that focuses primarily on aspects of the complex Czech part-of-speech tags and does not consider lexical information.

2 Dependency Parsing and Spanning Trees

2.1 Edge Based Factorization

In what follows, $\boldsymbol{x} = x_1 \cdots x_n$ represents a generic input sentence, and $\boldsymbol{y}$ represents a generic dependency tree for sentence $\boldsymbol{x}$. Seeing $\boldsymbol{y}$ as the set of tree edges, we write $(i,j) \in \boldsymbol{y}$ if there is a dependency in $\boldsymbol{y}$ from word x_i to word x_j.

In this paper we follow the common method of factoring the score of a dependency tree as the sum of the scores of all edges in the tree (Eisner, 1996)[1]. In particular, we define the score of an edge to be the dot product between a high dimensional feature representation of the edge and a weight vector,

$$s(i,j) = \mathbf{w} \cdot \mathbf{f}(i,j)$$

Thus the score of a dependency tree $\boldsymbol{y}$ for sentence $\boldsymbol{x}$ is,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in\boldsymbol{y}} s(i,j) = \sum_{(i,j)\in\boldsymbol{y}} \mathbf{w} \cdot \mathbf{f}(i,j)$$

[1]Eisner (1996) in fact uses probabilities for edge scores and thus defi nes the probability of a tree as the product, not sum, of all the edges in it.

Assuming an appropriate feature representation as well as a weight vector $\mathbf{w}$, dependency parsing is the task of finding the dependency tree $\boldsymbol{y}$ with highest score for a given sentence $\boldsymbol{x}$.

For the rest of this section we assume that the weight vector $\mathbf{w}$ is known and thus we know the score $s(i,j)$ of each possible edge. In Section 3 we present a method for learning the weight vector.

2.2 Maximum Spanning Trees

Consider a directed graph, $G = (V, E)$ in which each edge (i,j) (where $v_i, v_j \in V$) has a score $s(i,j)$. Since G is directed, $s(i,j)$ does not necessarily equal $s(j,i)$. The maximum spanning tree (MST) of G is the tree $\boldsymbol{y}$ that maximizes the value $\sum_{(i,j)\in\boldsymbol{y}} s(i,j)$, such that $(i,j) \in E$ and every vertex in V is used in the construction of $\boldsymbol{y}$. The maximum *projective* spanning, tree of G is constructed similarly except that it can only contain projective edges relative to some linear ordering on the vertices of G. The MST problem for directed graphs is also known as the maximum arborescence problem.

For each sentence $\boldsymbol{x}$ we define the directed graph $G_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ where

$$V_{\boldsymbol{x}} = \{x_0 = \text{root}, x_1, \ldots, x_n\} \text{ and}$$
$$E_{\boldsymbol{x}} = \{(i,j) : x_i \neq x_j, x_i \in V_{\boldsymbol{x}}, x_j \in V_{\boldsymbol{x}} - \text{root}\}$$

That is, $G_{\boldsymbol{x}}$ is a graph where all the words and the dummy root symbol are vertices and there is a directed edge between every pair of words and from the root symbol to every word. It is clear that dependency trees for $\boldsymbol{x}$ and spanning trees for $G_{\boldsymbol{x}}$ coincide, since both kinds of trees are required to reach all the words in the sentence. Hence, finding the (projective) dependency tree of highest score is equivalent to finding the maximum (projective) spanning tree in $G_{\boldsymbol{x}}$ rooted at the artificial root.

2.2.1 Non-projective Trees

To find the highest scoring non-projective tree we simply search the entire space of spanning trees with no restrictions. Well-known algorithms exist for the less general case of finding spanning trees in undirected graphs (Cormen et al., 1990). These algorithms even have k-best extensions (Eppstein, 1990).

Efficient algorithms for the directed case are less well known, but they exist, in particular the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Ed-

Chu-Liu-Edmonds(G, s)
Graph $G = (V, E)$
Edge weight function $s : E \rightarrow \mathbb{R}$
1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $G_C = \text{contract}(G, C, s)$
6. Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$
7. Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$
8. return $\boldsymbol{y} \cup C - \{(x'', x)\}$

contract$(G = (V, E), C, s)$
1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $x \in V - C : \exists_{x' \in C}(x', x) \in E$
   Add edge (c, x) to G_C with
   $s(c, x) = \max_{x' \in C} s(x', x)$
4. For $x \in V - C : \exists_{x' \in C}(x, x') \in E$
   Add edge (x, c) to G_C with
   $s(x, c) = \max_{x' \in C} [s(x, x') - s(a(x'), x') + s(C)]$
   where $a(v)$ is the predecessor of v in C
   and $s(C) = \sum_{v \in C} s(a(v), v)$
5. return G_C

Figure 3: Chu-Liu-Edmonds algorithm for finding maximum spanning trees in directed graphs.

monds, 1967). Informally, the algorithm has each vertex in the graph greedily select the incoming edge with highest weight. If a tree results, then this must be the maximum spanning tree. If not, there must be a cycle. The procedure identifies a cycle and contracts it into a single vertex and recalculates edge weights going into and out of the cycle. It can be shown that a maximum spanning tree on the contracted graph is equivalent to a maximum spanning tree in the original graph (Leonidas, 2003). Hence the algorithm can recursively call itself on the new graph. Pseudo-code for the algorithm is given in Figure 3. This presentation of the algorithm is based on that given by Leonidas (2003). Tarjan (1977) gives an efficient implementation of the algorithm that has a complexity of $O(n^2)$ for dense graphs, which is what we need here.

To find the highest scoring non-projective tree for a sentence, $\boldsymbol{x}$, we simply construct the graph $G_{\boldsymbol{x}}$ and run it through the Chu-Liu-Edmonds algorithm. The resulting spanning tree is the best non-projective dependency tree. This process is illustrated by an example in Appendix A.

The Chu-Liu-Edmonds algorithm has extensions to the k-best case (Hou, 1996), however, for dense graphs these algorithms have a runtime of $O(n^6)$. Hence, for natural language parsing, these k-best extensions are not feasible. This is not a fatal flaw for our purpose, but some of the training algorithms we discuss can benefit from having access to the k-best outcomes.

A possible concern with searching the entire space of spanning trees is that we have not used language-specific syntactic constraints to guide the search. Many languages that allow non-projectivity are still primarily projective. By searching all possible non-projective trees, we run the risk of finding extremely bad trees. We address this concern in Section 5.

2.2.2 Projective Trees

Using a slightly modified version of the CKY (Younger, 1967) chart parsing algorithm, it is possible to generate and represent all projective dependency trees in a forest that is $O(n^5)$ in size and takes $O(n^5)$ time to create. However, Eisner (1996) made the observation that if one keeps the head of each chart item to either the left or right periphery of that item, then it is possible to parse in $O(n^3)$. The idea is to parse the left and right dependents of a word independently, and combine them at a later stage. This removes the need for the additional head indices of the $O(n^5)$ algorithm and requires only two additional binary variables that specify the direction of the item (either gathering left dependents or gathering right dependents) and whether an item is complete (available to gather more dependents). Figure 4 illustrates the algorithm. We use r, s and t for the start and end indices of chart items, and h_1 and h_2 for the indices of the heads of chart items. In the first step, it can be seen that all items are complete. The algorithm then creates an incomplete item from the words h_1 to h_2 with h_1 as the head. This item is eventually completed at a later stage. As with normal CKY parsing, larger elements are created from pairs of smaller elements in a bottom-up fashion.

It is relatively easy to augment the Eisner algorithm so that each chart item also stores the score of the best possible tree that gave rise to the item. This augmentation is identical to those used for the standard CKY algorithms. We must also store back pointers so that it is possible to reconstruct the best tree from the chart item that spans the entire sentence. The Eisner algorithm is discussed in more

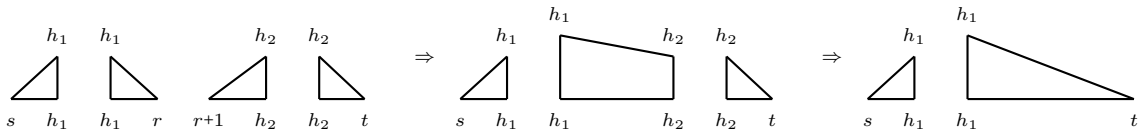


Figure 4: $O(n^3)$ algorithm of Eisner (1996), needs to keep 3 indices at any given stage.

detail in Appendix B.

For the maximum projective spanning tree problem it is easy to show that the Eisner dependency parsing algorithm is an exact solution if we are given a linear ordering of the vertices in the graph. Indeed, every projective dependency tree of sentence $\boldsymbol{x}$ is also a projective spanning tree of the graph $G_{\boldsymbol{x}}$ and vice-versa. Thus, if we can find the maximum projective dependency tree using the Eisner algorithm, then we can also find the maximum spanning tree. For natural language dependency tree parsing, the linear ordering on the graph vertices is explicitly given by the order of the words in the sentence.

In addition to running in $O(n^3)$, the Eisner algorithm has the additional benefit that it is a bottom-up dynamic programming chart parsing algorithm allowing for k-best extensions that increase complexity by just a factor of $O(k \log k)$ (Huang and Chiang, 2005).

2.3 Dependency Trees as MSTs: Summary

In the preceding discussion, we have shown that natural language dependency parsing can be reduced to finding maximum spanning trees in directed graphs. This reduction results from edge-based factorization and can be applied to projective languages with the Eisner parsing algorithm and non-projective languages with the Chu-Liu-Edmonds maximum spanning tree algorithm. The only remaining problem is how to learn the weight vector $\mathbf{w}$. This will be discussed in the next section.

A major advantage of our approach over other dependency parsing models is its uniformity and simplicity. By viewing dependency structures as spanning trees, we have provided a general framework for parsing trees for both projective and non-projective languages. Furthermore, our parsing algorithms are more efficient than lexicalized phrase structure parsing algorithms, allowing us to search the entire space without any pruning. In particular our non-projective parsing algorithm based on the Chu-Liu-Edmonds MST algorithm provides *true* non-projective parsing. This is in contrast to other non-projective methods, such as that of Nivre and Nilsson (2005), who implement non-projectivity in a *pseudo-projective* parser with edge transformations. This formulation also dispels the notion that non-projective parsing is "harder" than projective parsing. In fact, it is actually easier since the best known non-projective parsing complexity is $O(n^2)$ versus the $O(n^3)$ complexity of the Eisner projective parsing algorithm.

It is worth noting that our formalism for dependency parsing is grammarless. This has become common in the parsing community. For instance, the Collins parser (Collins, 1999) does not use a specific context-free grammar, but searches the entire space of phrase structures. This is also true for some state-of-the-art dependency parsers such as Yamada and Matsumoto (2003), who search the same space, but with heavy pruning. This contrasts with grammar-based dependency parsers such as link-grammar parsers (Sleator and Temperley, 1993) and constraint dependency grammars (Wang and Harper, 2004).

3 Online Learning

In this section, we describe an online large-margin method for learning the weight vector $\mathbf{w}$. Large-margin classifiers have been shown to provide good regularization when applied to document classification (Joachims, 2002) and sequential tagging (Collins, 2002; Taskar et al., 2003). As usual for supervised learning, we assume a training set $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$, consisting of pairs of a sentence $\boldsymbol{x}_t$ and its correct dependency tree $\boldsymbol{y}_t$.

The online-learning algorithms we consider are instances of the algorithm schema in Figure 5. A single training instance is examined at each iteration, and the weight vector is updated by algorithm-specific rule. The auxiliary vector $\mathbf{v}$ accumulates the

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$
1. $\mathbf{w}_0 = 0;\ \mathbf{v} = 0;\ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. $\mathbf{w}^{(i+1)} =$ update $\mathbf{w}^{(i)}$ according to instance $(\boldsymbol{x}_t, \boldsymbol{y}_t)$
5. $\mathbf{v} = \mathbf{v} + \mathbf{w}^{(i+1)}$
6. $i = i + 1$
7. $\mathbf{w} = \mathbf{v}/(N * T)$

Figure 5: Generic online learning algorithm.

successive values of of $\mathbf{w}$, so that the final weight vector is the *average* of the weight vectors after each iteration. This averaging effect has been shown to help reduce overfitting (Collins, 2002).

In what follows, $\text{dt}(\boldsymbol{x})$ denotes the set of possible dependency trees for sentence $\boldsymbol{x}$, and $\text{best}_k(\boldsymbol{x}; \mathbf{w}) \subseteq \text{dt}(\boldsymbol{x})$ denotes the set of k highest scoring trees relative to the weight vector $\mathbf{w}$.

3.1 MIRA

(Crammer and Singer, 2001) present a natural approach to large-margin multi-class classification, which was later extended by (Taskar et al., 2003) to structured classification:

$$\begin{aligned}
&\min \|\mathbf{w}\| \\
&\text{s.t.} \quad s(\boldsymbol{x}, \boldsymbol{y}) - s(\boldsymbol{x}, \boldsymbol{y}') \geq L(\boldsymbol{y}, \boldsymbol{y}') \\
&\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T},\ \boldsymbol{y}' \in \text{dt}(\boldsymbol{x})
\end{aligned}$$

where $L(\boldsymbol{y}, \boldsymbol{y}')$ is a real-valued loss for the output $\boldsymbol{y}'$ relative to the correct output $\boldsymbol{y}$.

Informally, this minimizes the norm of the weight vector subject to *margin constraints* that keep the score of the correct output above the score of each incorrect output by an amount given by the loss of the incorrect output.

The Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2003) employs this optimization directly within the online framework. On each update, MIRA attempts to keep the new weight vector as close as possible to the old weight vector, subject to correctly classifying the instance under consideration with a margin given by the loss of the incorrect classifications. This can be formalized by substituting the following update into line 4 of the

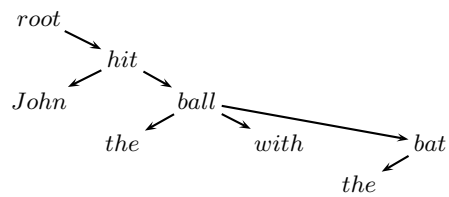


Figure 6: An example incorrect dependency tree relative to that in Figure 1. The loss of this dependency tree is 2 since *with* and *bat* are incorrectly identified as dependents of *ball*.

generic online algorithm,

$$\begin{aligned}
&\min \left\|\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}\right\| \\
&\text{s.t. } s(\boldsymbol{x}_t, \boldsymbol{y}_t) - s(\boldsymbol{x}_t, \boldsymbol{y}') \geq L(\boldsymbol{y}_t, \boldsymbol{y}') \\
&\forall \boldsymbol{y}' \in \text{dt}(\boldsymbol{x}_t)
\end{aligned}$$

This quadratic programming problem can be solved using Hildreth's algorithm (Censor and Zenios, 1997). Crammer and Singer (2003) and Crammer et al. (2003) provide an analysis of both the online generalization error and convergence properties of MIRA.

For the dependency/spanning tree problem, we defined the loss of a tree to be the number of words with incorrect incoming edges relative to the correct tree. This is closely related to the Hamming loss that is often used for sequences (Taskar et al., 2003). For instance, consider the correct tree in Figure 1 and an incorrect tree in Figure 6. The loss of the incorrect tree relative to the correct one is 2 since *with* and *bat* are both incorrectly labeled as dependents of *ball*. This is just one possible definition of the loss for an incorrect tree. Other possibilities are the 0-1 loss (Taskar, 2004) or another more linguistically motivated loss that penalizes some errors (say conjunction and preposition dependencies) over others. Being able to learn a weight setting relative to some loss function is quite advantageous. For many applications, there are certain parts of a dependency tree that are relevant and others that are not. One merely needs to change the loss function to focus on reducing specific errors in the trees.

To use these algorithms for parsing, we follow the common identification of structure prediction with multi-class classification, in which each structure is a possible class for a sentence. The primary problem with this view is that for arbitrary inputs there are

typically exponentially many possible classes and thus exponentially many margin constraints. This is the case for dependency parsing.

3.1.1 k-best MIRA

One solution for the exponential blow-up in number of classes is to relax the optimization by using only the margin constraints for the k outputs $\boldsymbol{y}$ with the highest scores $s(\boldsymbol{x}, \boldsymbol{y})$. The resulting online update (to be inserted in Figure 5, line 4) would then be:

$$\begin{aligned} &\min \ \left\|\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}\right\| \\ &\text{s.t.} \quad s(\boldsymbol{x}_t, \boldsymbol{y}_t) - s(\boldsymbol{x}_t, \boldsymbol{y}') \geq L(\boldsymbol{y}_t, \boldsymbol{y}') \\ &\forall \boldsymbol{y}' \in \text{best}_k(\boldsymbol{x}_t; \mathbf{w}^{(i)}) \end{aligned}$$

In Section 5 we show that this relaxation works well and even small values of k yield near optimal performance. We call this algorithm *k-best MIRA*.

With 1-best MIRA, we can use the Eisner or the Chu-Liu-Edmonds algorithms to find the best dependency trees during training and make the appropriate update to the weight vector. As mentioned earlier, the Eisner algorithm can be extended to the k-best case. However, the Chu-Liu-Edmonds algorithm cannot be extended in practice and we are force to set $k = 1$ when using it.

3.1.2 Factored MIRA

Another solution to the blow-up in possible parses is to exploit the structure of the output to factor the exponential number of margin constraints into a polynomial number of local constraints (Taskar et al., 2003; Taskar et al., 2004; Taskar, 2004).

For the directed maximum spanning tree problem, we can factor the output by edges to obtain the following constraints:

$$\begin{aligned} &\min \left\|\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}\right\| \\ &\text{s.t. } s(l, j) - s(k, j) \geq 1 \\ &\forall (l, j) \in \boldsymbol{y}_t, (k, j) \notin \boldsymbol{y}_t \end{aligned}$$

This states that the weight of the correct incoming edge to the word x_j and the weight of all other incoming edges must be separated by a margin of 1. It is easy to show that when all these constraints are satisfied, the correct spanning tree and all incorrect spanning trees are separated by a score at least as large as the number of incorrect incoming edges. This is because the scores for all the correct arcs cancel out, leaving only the scores for the errors causing the difference in overall score. Since each single error results in a score increase of at least 1, the entire score difference must be at least the number of errors.

Though this condition is sufficient, it is not necessary. For sequences, this form of factorization has been called local lattice preference (Crammer et al., 2004). Let n be the number of nodes in graph $G_{\boldsymbol{x}}$. Then the number of constraints is $O(n^2)$, since for each node we must maintain $n - 1$ constraints.

4 Feature Space

In Section 2.1 we defined the score of an edge as

$$s(i, j) = \mathbf{w} \cdot \mathbf{f}(i, j)$$

This assumes that we have a high-dimensional feature representation for each edge (i, j). MIRA, like perceptron (Rosenblatt, 1958) and support-vector machines (Boser et al., 1992), has also direct dual formulation that supports the use of kernels between instances instead of an explicit feature representation. Kernels would allow us to express relationships between instances that would be impossible or impractical to express in terms of feature vectors, but the dual formulation would require computing kernel values between all instances (or instance parts in the factored case), which is impractical for the large training sets we consider here. Therefore, all of our experiments use the primal formulation in terms of feature vectors.

The basic set of features we use are shown in Table 1a and b. All features are conjoined with the direction of attachment as well as the distance between the two words creating the dependency. These features provide back-off from very specific features over words and part-of-speech (POS) tags to less sparse features over just POS tags. These features are added for both the entire words as well as the 5-gram prefix if the word is longer than 5 characters.

Using just features over edges (or word pairs) in the tree was not enough for high accuracy since all attachment decisions were made outside of the context in which the words occurred. To solve this problem, we added two more types of features, which can be seen in Table 1c. The first feature class recognizes word types that occur between the parent and

child words in an attachment decision. These features take the form of POS trigrams: the POS of the parent, that of the child, and that of a word in between, for all the words between the parent and the child. This feature was particularly helpful for nouns to select their parents correctly, since it helps reduce the score for attaching a noun to another noun with a verb in between, which is a relatively infrequent configuration. The second class of additional features represents the local context of the attachment, that is, the words before and after the parent-child pair. These features take the form of POS 4-grams: The POS of the parent, child, word before/after parent and word before/after child. We also include back-off features to trigrams where one of the local context POS tags was removed.

These new features can be efficiently added since they are given as part of the input and do not rely on knowledge of dependency decisions outside the current edge under consideration. Adding these features resulted in a large improvement in performance and brought the system to state-of-the-art accuracy. For illustrative purposes Appendix C shows the feature representation for our example sentence over the edge (*hit,with*).

All the features described here are binary.

4.1 Language Generality

The feature set we propose is generalizable to any language that can be tokenized and assigned POS tags similar to English. In fact, our feature templates were created by trial and error on our English development set, but used for both our English and Czech parsers. The only difference between the two parsers is that they are trained on language specific data sets.

5 Experiments

We performed experiments on two sets of data, the English Penn Treebank (Marcus et al., 1993) and the Czech Prague Dependency Treebank (PDT) (Hajič, 1998). For the English data we extracted dependency trees using the rules of Yamada and Matsumoto (2003), which are similar, but not identical, to those used by Collins (1999). Because the dependency trees are extracted from the phrase structures in the Penn Treebank, they are by convention exclusively projective. We used sections 02-21 of the Treebank for training data, section 22 for development and section 23 for testing. All experiments were run using every single sentence in each set of data regardless of length. For the English data only, we followed the standards of Yamada and Matsumoto (2003) and did not include punctuation in the calculation of accuracies. For the test set, the number of words without punctuation is 49,892, which aligns with the experiments of Yamada and Matsumoto. Since our system assumes part-of-speech information as input, we used the maximum entropy part-of-speech tagger of Ratnaparkhi (1996) to provide tags for the development and testing data. The number of features extracted from the Penn Treebank was $6,998,447$.

For the Czech data, we did not have to automatically extract dependency structures since manually annotated dependency trees are precisely what the PDT contains. We used the predefined training, development and testing split of this data. Furthermore, we used the automatically generated POS tags that were provided with the data. Czech POS tags are extremely complex and consist of a series of slots that may or may not be filled with some value. These slots represent lexical properties such as standard POS, case, gender, and tense. The result is that Czech POS tags are rich in information, but quite sparse when viewed as a whole. To reduce sparseness, our features rely only on the reduced POS tag set from Collins et al. (1999). The number of features extracted from the PDT training set was $13,450,672$.

Czech has more flexible word order than English and as a result the PDT contains non-projective dependencies. On average, 23% of the sentences in the training, development and test sets have at least one non-projective dependency. However, less than 2% of total edges are actually non-projective. Therefore, handling non-projective arcs correctly has a relatively small effect on overall accuracy. To show the effect more clearly, we created two Czech data sets. The first, Czech-A, consists of the entire PDT. The second, Czech-B, includes only the 23% of sentences with at least one non-projective dependency. This second set will allow us to analyze the effectiveness of the algorithms on non-projective material.

The following sections describe three main sys-

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Bi-gram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

Table 1: Features used by system. p-word: word of parent in dependency edge. c-word: word of child. p-pos: POS of parent. c-pos: POS of child. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child.

tems we evaluated.

5.1 k-best MIRA Eisner

This system uses the projective dependency parsing algorithm of Eisner to obtain the k best dependency trees for k-best MIRA training. Even though this system cannot extract non-projective dependencies, it still does very well since less than 2% of all dependency edges in our datasets are non-projective (0% for English). We also use the Eisner algorithm for testing since we found it performed the best on this model. We set $k = 5$ for this system.

5.2 k-best MIRA CLE

In this system we use the Chu-Liu-Edmonds algorithm to find the best dependency tree for 1-best MIRA training and testing. The asymptotic decoding complexity is actually better than the projective case: $O(n^2)$ (Tarjan, 1977) versus $O(n^3)$ (Eisner, 1996). We use the Chu-Liu-Edmonds algorithm to find the best tree for the test data.

5.3 Factored MIRA

This system uses the quadratic set of constraints based on edge factorization as described in Section 3.1.2. We use the Chu-Liu-Edmonds algorithm to find the best tree for the test data.

5.4 Results

Results are shown in Table 2 (Czech) and Table 3 (English). Each table has three metrics. The first and most widely recognized is *Accuracy*, which measures the number of words that correctly identified their parent in the tree. The second is *Root*, which measures the number of sentences that correctly identified the root word of the sentence (the real root, not the artificially inserted root). For Czech this is actually the harmonic mean between the precision and recall since the PDT allows sentences to have more than one root. The final metric is *Complete*, which measures the number of sentences in which the resulting tree was completely correct.

For Czech, we compare our results to Nivre and Nilsson (2005). This system also allows for the introduction of non-projective arcs through the use of pseudo-projective parsing with edge transformations. Nivre and Nilsson (2005) do not report root accuracies. We report results for all three models.

Clearly, there is an advantage in using the Chu-Liu-Edmonds algorithm for Czech dependency parsing. Even though less than 2% of all dependencies are non-projective, we still see an absolute improvement of up to 1.1% in overall accuracy over the projective model. Furthermore, when we focus on the subset of data that only contains sentences with at least one non-projective dependent, the effect is amplified. Another major improvement here is that the Chu-Liu-Edmonds non-projective MST algorithm has a parsing complexity of $O(n^2)$, versus the $O(n^3)$ complexity of the projective Eisner algorithm. This is substantial considering that $n = 20$ on average. However, for the factored model, we do have $O(n^2)$ margin constraints versus k for the k-best models, which does result in a significant increase in training time.

One concern raised in Section 2.2.1 is that searching the entire space of non-projective trees could cause problems for languages that are primarily projective. However, as we can see, this is not a prob-

	Czech-A			Czech-B		
	Accuracy	**Root**	**Complete**	**Accuracy**	**Root**	**Complete**
N&N2005	80.1	-	22.2	-	-	-
k-best MIRA Eisner	83.3	88.6	31.3	74.8	82.0	0.0
1-best MIRA CLE	84.1	89.0	32.2	81.0	86.5	14.9
factored MIRA	84.4	88.3	32.3	81.5	85.8	14.3

Table 2: Dependency parsing results for Czech. Czech-B is the subset of Czech-A containing only sentences with at least one non-projective dependency.

	English		
	Accuracy	**Root**	**Complete**
Y&M2003	90.3	91.6	38.4
N&S2004	87.3	84.3	30.4
k-best MIRA Eisner	90.9	94.2	37.5

Table 3: Dependency parsing results for English compared with other state-of-the-art models.

lem. The main reason is probably that the learning algorithm sets low weights for non-projective arcs between words that are distant from each other. Furthermore, in the k-best learning scheme, our parameters are set relative to our parsing algorithm. Hence, the parser should learn to produce either projective or mildly non-projective trees.

One option with the k-best projective system is to rearrange the words in the training set so that the data is projectivized (Collins et al., 1999). This at least allows the training algorithm to obtain reasonably low error on the training set. We found that this did improve performance slightly to 83.6% accuracy. However, this model still uses the more cumbersome $O(n^3)$ Eisner algorithm. Furthermore this method is somewhat unprincipled in that it scrambles contextual features somewhat.

For English, we compared the model to the systems of Yamada and Matsumoto (2003) and Nivre and Scholz (2004). The only model we present results is *k-best MIRA Eisner* (Section 5.1) since we found all other models to perform worse (see Appendix D for more on this). This is of course because they do not take into account the a-priori knowledge that the English data is exclusively projective.

It can be seen that our online large-margin models using the projective algorithm of Eisner provides state-of-the-art performance when compared to current systems. In particular, the models presented here do much better than all previous models for finding the correct root in the dependency tree. This is most likely due to the fact that our models do not prune the search space and thus are not subject to error propagation due to poor decisions near leaves in the tree.

The results just discussed are the highlights from a complete experiment matrix. We have two degrees of freedom, training and test. For training, we can choose k-best MIRA with projective Eisner parsing, 1-best MIRA with Chu-Liu-Edmonds non-projective parsing, or factored MIRA. At test time, we can choose to use either projective or non-projective parsing. Appendix D presents the results for the full set of training-test combinations for both English and Czech.

5.4.1 Lexicalized Phrase Structure Parsers

It is well known that dependency trees extracted from lexicalized phrase structure parsers (Collins, 1999; Charniak, 2000) typically are more accurate than those produced by pure dependency parsers (Yamada and Matsumoto, 2003). We compared our system to the Bikel re-implementation of the Collins parser (Bikel, 2004; Collins, 1999) trained with the same head rules as our system. There are two ways to extract dependencies from lexicalized phrase structure. The first is to use the automatically generated dependencies that are explicit in the lexicalization of the trees. We call this system *Collins-auto*. The second is to take just the phrase structure output of the parser and run the automatic head rules over it to extract the dependencies. We call this system *Collins-rules*. Table 4 shows the results comparing our system, *MIRA-normal*, to the Collins parser for English. All systems are implemented in Java and run on the same machine.

Interestingly, the dependencies that are automatically produced by the Collins parser are worse than

	English				
	Accuracy	**Root**	**Complete**	**Complexity**	**Time**
Collins-auto	88.2	92.3	36.1	$O(n^5)$	98m 21s
Collins-rules	91.4	95.1	42.6	$O(n^5)$	98m 21s
MIRA-normal	90.9	94.2	37.5	$O(n^3)$	5m 52s
MIRA-Collins	92.2	95.8	42.9	$O(n^5)$	105m 08s

Table 4: Results comparing our system to those based on the Collins parser. *Complexity* represents the computational complexity of each parser and *Time* the CPU time to parse sec. 23 of the Penn Treebank.

those extracted statically using the head rules. Arguably, this displays the artificialness of English dependency parsing using dependencies automatically extracted from treebank phrase-structure trees. Our system falls in-between, better than the automatically generated dependency trees and worse than the head-rule extracted trees.

Since the dependencies returned from our system are better than those actually learnt by the Collins parser, one could argue that our model is learning to parse dependencies more accurately. However, phrase structure parsers are built to maximize the accuracy of the phrase structure and use lexicalization as just an additional source of information. Thus it is not too surprising that the dependencies output by the Collins parser are not as accurate as our system, which is trained and built to maximize accuracy on dependency trees. In complexity and runtime, our system is a huge improvement over the Collins parser.

The final system in Table 4 takes the output of *Collins-rules* and adds a feature to *MIRA-normal* that indicates for given edge, whether the Collins parser believed this dependency actually exists. We call this system *MIRA-Collins*. This is a well known discriminative training trick — using the suggestions of a generative system to influence decisions. This system can essentially be considered a corrector of the Collins parser and represents a significant improvement over it. However, there is an added complexity with such a model as it requires the output of the $O(n^5)$ Collins parser.

5.4.2 k-best MIRA Approximation

We need to determine how justifiable is the k-best MIRA approximation. Table 5 indicates the test accuracy and training time on English for the k-best MIRA Eisner system with $k = 1, 2, 5, 10, 20$. Even though the k-best parsing multiplies asymptotic parsing complexity by a $k \log k$ factor, empirically the training times seem to scale linearly with k. Peak performance is achieved for low k with a slight degradation around $k = 20$. The most likely reason for this phenomenon is that the model is overfitting by ensuring that even unlikely trees are separated from the correct tree in proportion to their loss.

	k=1	k=2	k=5	k=10	k=20
Accuracy	90.73	90.82	90.88	90.92	90.91
Train time	183m	235m	627m	1372m	2491m

Table 5: Evaluation of k-best MIRA approximation.

5.5 Training Time and Model Size

A major concern when training discriminative learners on large training sets for computationally heavy tasks such as dependency parsing is training time. Currently, on a 64-bit Linux machine running Java 1.5, the largest model (Czech on the full data with the Eisner algorithm) takes just under a day to train. However, most models take under 10 hours. These times increase by a factor of 2-2.5 when the model is run on a 32-bit machine. Currently, the Czech model can only be run on the 64-bit machine, because of the very large feature set. However, we could easily remedy this by only including features that occur more than once in the training set. This reduces the feature space substantially with little harm to performance. Feature count cut-offs are common, but typically used as a form of regularization.

5.6 Experimental Summary

To summarize, in this section we presented results for online large-margin algorithms trained to parse dependency trees either as projective structures (using the Eisner algorithm) or non-projective structures (using the Chu-Liu-Edmonds algorithm). For English, we showed that the model outper-

forms other pure dependency parsers assuming projective structures. For Czech, we compared both a projective and non-projective model and showed that we achieve superior performance with the non-projective system. This is interesting for several reasons. First, it shows that viewing dependency trees as maximum spanning trees is a viable, efficient and general method for untyped dependency parsing. Second, it allows a natural method for parsing non-projective trees through the use of the Chu-Liu-Edmonds algorithm, which is asymptotically more efficient then the projective algorithm of Eisner.

In addition, we showed that our models compare favorably to more powerful lexicalized phrase structure models in terms of performance and parsing time. In particular the models we describe here can outperform lexicalized phrase structure parsers when they are treated as correctors of the output of these models. We also showed empirically that our k-best learning approximation is reasonable.

We should note the work of Zeman (2004), who presents good results for Czech based on an extension to the work of Collins et al. (1999). In his system, he transforms Czech dependency structures to phrase structure and uses both the Collins parser and Charniak parser for learning and extracting lexicalized phrase-structure. These structures are then deterministically transformed back into dependencies. The results presented by Zeman are comparable to those shown here. However, Zeman only presents results for the development data. Furthermore, since these models are based on lexicalized phrase structure parsing they have a parsing complexity of $O(n^5)$.

6 Conclusions and Future Work

In this paper we presented a general framework for parsing dependency trees based on an equivalence to maximum spanning trees in directed graphs. This framework provides natural and efficient mechanisms for parsing both projective and non-projective languages through the use of the Eisner and Chu-Liu-Edmonds algorithms. To learn these structures we used a simple supervised online large-margin learning technique (MIRA) that empirically provides state-of-the-art performance for both English and Czech.

A major advantage of our models is the ability to naturally model non-projective parses. Non-projective parsing is commonly considered more difficult than projective parsing. However, under our framework, we show that the opposite is actually true and that non-projective parsing has a lower asymptotic complexity. Using this framework, we presented results showing that the non-projective model outperforms the projective model on the Prague Dependency Treebank, which contains a small number of non-projective edges.

The systems we present are also language general in that none of the features are tailored specifically to any one language. In fact, it is reasonable to assume that any language that may be tokenized and assigned part-of-speech tags like English and Czech can be modeled by our framework. To test this hypothesis we plan on applying our parser on other languages such as Dutch and Turkish, both of which have available treebanks.

We plan to extend the present work to typed dependency structures. In particular, we wish to identify what kinds of edge types are useful for parsing and information extraction applications and how such types can be modeled in a spanning tree framework.

Another direction for future work is to investigate different methods of factorizing the output for the factored MIRA learning method. We used a rather naive and strict factorization. A softer factorization would be similar to the M^3 factorizations described by Taskar (2004). It would be interesting to see which factorizations work best for projective and non-projective languages and what this tells us about the structure of these languages.

Acknowledgments

We thank Jan Hajič for answering queries on the Prague Treebank; Joakim Nivre for providing the Yamada and Matsumoto (2003) head rules for English, which allowed for a direct comparison of our systems, and a preprint of of his pseudo-projective parsing paper; Nikhil Dinesh for the non-projective English example; and Lillian Lee for pointing out the work of Ribarov (2004). This work combines research in machine learning for structured data and in information extraction supported by NSF under the ITR grants 0205456, 0205448, and 0428193.

References

A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1).

D.M. Bikel. 2004. Intricacies of collins parsing model. *Computational Linguistics (to appear)*.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Computational Learning Theory*, pages 144–152.

Y. Censor and S.A. Zenios. 1997. *Parallel optimization : theory, algorithms, and applications*. Oxford University Press.

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

Y.J. Chu and T.H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

S. Clark and J.R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proc. ACL*.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. ACL*.

M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann. 1999. A statistical parser for Czech. In *Proc. ACL*.

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

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

T.H. Cormen, C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill.

K. Crammer and Y. Singer. 2001. On the algorithmic implementation of multiclass kernel based vector machines. *JMLR*.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*.

K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2003. Online passive aggressive algorithms. In *Proc. NIPS*.

K. Crammer, R. McDonald, and F. Pereira. 2004. New large margin algorithms for structured prediction. In *Learning with Structured Outputs Workshop (NIPS)*.

A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proc. ACL*.

Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proc. ACL*.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. COLING*.

D. Eppstein. 1990. Finding the k smallest spanning trees. In *2nd Scand. Workshop on Algorithm Theory*.

J. Hajič. 1998. Building a syntactically annotated corpus: The Prague dependency treebank. *Issues of Valency and Meaning*, pages 106–132.

H. Hirakawa. 2001. Semantic dependency analysis method for japanese based on optimum tree search algorithm. In *Proc. of PACLING*.

W. Hou. 1996. Algorithm for finding the first k shortest arborescences of a digraph. *Mathematica Applicata*, 9(1):1–4.

L. Huang and D. Chiang. 2005. Better k-best parsing. Technical Report MS-CIS-05-08, University of Pennsylvania.

Richard Hudson. 1984. *Word Grammar*. Blackwell.

T. Joachims. 2002. *Learning to Classify Text using Support Vector Machines*. Kluwer.

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

G. Leonidas. 2003. Arborescence optimization problems solvable by Edmonds' algorithm. *Theoretical Computer Science*, 301:427 – 437.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. Proc. ACL.

J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of english text. In *Proc. COLING*.

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

A. Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

K. Ribarov. 2004. *Automatic building of a dependency tree*. Ph.D. thesis, Charles University.

S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proc. ACL*.

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 68:386–407.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fi elds. In *Proc. HLT-NAACL*, pages 213–220.

Y. Shinyama, S. Sekine, K. Sudo, and R. Grishman. 2002. Automatic paraphrase acquisition from news articles. In *Proc. HLT*.

D. Sleator and D. Temperley. 1993. Parsing english with a link grammar. In *Proceedings of IWPT*.

R. Snow, D. Jurafsky, and A. Y. Ng. 2004. Learning syntactic patterns for automatic hypernym discovery. In *NIPS 2004*.

R.E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7:25–35.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *Proc. NIPS*.

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

B. Taskar. 2004. *Learning Structured Prediction Models: A Large Margin Approach*. Ph.D. thesis, Stanford.

W. Wang and M. P. Harper. 2004. A statistical constraint dependency grammar (cdg) parser. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*.

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

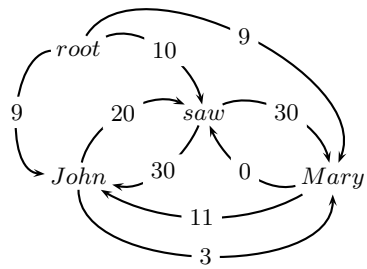
D.H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 12(4):361–379.

Daniel Zeman. 2004. *Parsing with a Statistical Dependency Model*. Ph.D. thesis, Univerzita Karlova, Praha.

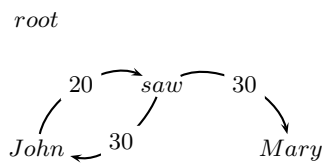
APPENDIX

A Chu-Liu-Edmonds Example

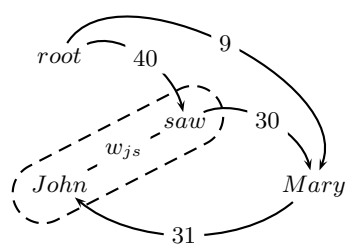
We illustrate here the application of the Chu-Liu-Edmonds algorithm to dependency parsing on the simple example$\boldsymbol{x}$ = *John saw Mary*. We assume that we know $\mathbf{w}$. The directed graph representation $G_{\boldsymbol{x}}$ of sentence $\boldsymbol{x}$ is



The first step of the algorithm is to find, for each word, the highest scoring incoming edge

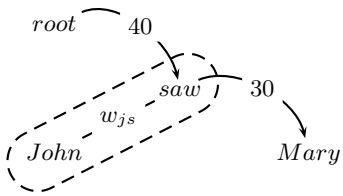


If the result were a tree, it would have to be the maximum spanning tree. However, in this case we have a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 3.

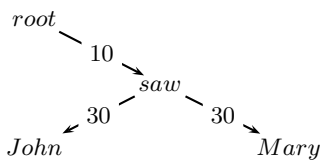


The new vertex w_{js} represents the contraction of vertices *John* and *saw*. The edge from w_{js} to *Mary* is 30 since that is the highest scoring edge from any vertex in w_{js}. The edge from *root* into w_{js} is set to 40 since this represents the score of the best spanning tree originating from *root* and including only the vertices in w_{js}. The same leads to the edge from *Mary* to w_{js}. The fundamental property of the Chu-Liu-Edmonds algorithm is that a MST in this graph can be transformed into an MST in the original graph. Thus, we recursively call the algorithm

on this graph. Note that we need to keep track of the real endpoints of the edges into and out of w_{js} for reconstruction later. Running the algorithm, we must find the best incoming edge to all words,



This is a tree and thus the MST of this graph. We now need to go up a level and reconstruct the graph. The edge from w_{js} to *Mary* originally was from the word *saw*, so we include that edge. Furthermore, the edge from *root* to w_{js} represented a tree from *root* to *saw* to *John*, so we include all those edges to get the MST,

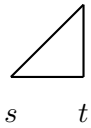


This is obviously the MST for this graph.

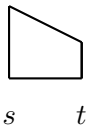
B Eisner Algorithm

Here we give a brief description of the Eisner algorithm (Eisner, 1996) for parsing projective dependency/spanning trees.

Let $C[s][t][d][c]$ be a dynamic programming table that stores the score of the best subtree from position s to position t, $s \leq t$, with direction d and complete value c. $d \in \{\leftarrow, \rightarrow\}$ and indicates the direction of the subtree (gathering left or right dependents). If $d = \leftarrow$ then t must be the head of the subtree and if $d = \rightarrow$ then s is the head. $c \in \{0, 1\}$ indicates if a subtree is complete ($c = 1$, no more dependents) or incomplete ($c = 0$, needs to be completed). For instance, $C[s][t][\leftarrow][1]$ would be the score of the best subtree represented by the item,

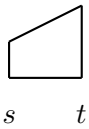


and $C[s][t][\rightarrow][0]$ for the following item,

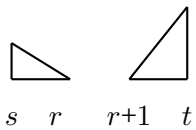


The Eisner algorithm fills in the dynamic programming table bottom-up just like the CKY parsing algorithm (Younger, 1967) by finding optimal subtrees for substrings of increasing increasing length. Pseudo code for filling up the dynamic programming table is in Figure 7.

Consider the line in Figure 7 indicated by (*). This says that to find the best score for an incomplete left subtree



we need to find the index $s \leq r < t$ that leads to the best possible score through joining two complete subtrees,



The score of joining these two complete subtrees is the score of these subtrees plus the score of creating an edge from word x_t to word x_s. This is guaranteed to be the score of the best subtree provided the table correctly stores the scores of all smaller subtrees. This is because by enumerating over all values of r, we are considering all possible combinations.

By forcing a unique root at the left-hand side of the sentence, the score of the best tree for the entire sentence is $C[1][n][\rightarrow][1]$. The only remaining problem is how to extract the best dependency tree after running the Eisner algorithm. In order to do this we simply need to maintain back pointers to the subtrees that gave rise to the each item in the dynamic programming table. This is identical to maintaining back pointers for the Viterbi algorithm for sequences and the CKY algorithm for parsing.

A quick look at the pseudo-code shows that the run-time of the Eisner algorithm is $O(n^3)$. Note, that unlike CFG parsing, there is no grammar constant. This is significant because large scale CFG parsing can sometimes have a grammar constant in the thousands.

Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$
for $k : 1..n$
  for $s : 1..n$
    $t = s + k$
    if $t > n$ then break

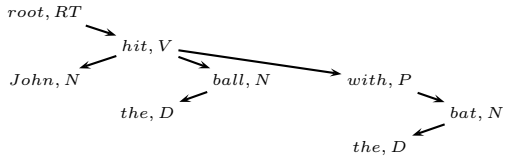
$$C[s][t][\leftarrow][0] = max_{s \leq r < t}(C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s)) \quad (*)$$
$$C[s][t][\rightarrow][0] = max_{s \leq r < t}(C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$$
$$C[s][t][\leftarrow][1] = max_{s \leq r < t}(C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$$
$$C[s][t][\rightarrow][1] = max_{s < r \leq t}(C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$$

  end for
end for

Figure 7: Pseudo-code for bottom-up Eisner cubic parsing algorithm.

C Feature Example

Here we show a concrete example of the feature representation of an edge in a dependency tree. The tree is given below and the edge of interest is the dependency between the main verb *hit* and its argument headed preposition *with*. We use simplified part-of-speech tags for illustrative purposes only.



f(hit,with)

Basic Features

p-word="hit", p-pos="V", c-word="with", c-pos="P"
p-pos="V", c-word="with", c-pos="P"
p-word="hit", c-word="with", c-pos="P"
p-word="hit", p-pos="V", c-pos="P"
p-word="hit", p-pos="V", c-word="with"
p-word="hit", c-word="with"
p-pos="V", c-pos="P"
p-word="hit", p-pos="V"
c-word="with", c-pos="P"
p-word="hit"
p-pos="V"
c-word="with"
c-pos="P"

Extended Features

p-pos="V", b-pos="D", c-pos="P"
p-pos="V", b-pos="N", c-pos="P"
p-pos="V", p-pos+1="D", c-pos-1="N", c-pos="P"
p-pos="V", c-pos-1="N", c-pos="P"
p-pos="V", p-pos+1="D", c-pos="P"
p-pos-1="N", p-pos="V", c-pos-1="N", c-pos="P"
p-pos="V", c-pos-1="N", c-pos="P"
p-pos-1="N", p-pos="V", c-pos="P"
p-pos="V", p-pos+1="D", c-pos="P", c-pos+1="D"
p-pos="V", c-pos="P", c-pos+1="D"
p-pos="V", p-pos+1="D", c-pos="P"
p-pos-1="N", p-pos="V", c-pos="P", c-pos+1="D"
p-pos="V", c-pos="P", c-pos+1="D"
p-pos-1="N", p-pos="V", c-pos="P"

Note that since *hit* and *with* are not longer than 5 characters we do not have any additional 5-gram back-off features. If, however, the verb was *smashed*, we could have the feature,

p-word:5="smash", c-word="with"

along with other 5-gram back-off features.

All features are also conjoined with the direction of attachment and the distance between the words. So, in addition to the feature,

p-word="hit", c-word="with"

the system would also have the feature,

p-word="hit", c-word="with", dir=R, dist=2

to indicate that the child *with* is to the right of the parent *hit* and that they are separated by 2 words. Distances were calculated into buckets with thresholds of 1, 2, 3, 4, 5 and 10.

	ENGLISH		
	k-best MIRA: Eisner	1-best MIRA: CLE	Factored MIRA
Eisner	90.9 / 94.2 / 37.5	90.6 / 94.2 / 35.3	90.6 / 93.1 / 34.3
CLE	83.7 / 81.8 / 17.5	90.2 / 94.3 / 33.2	90.2 / 92.9 / 32.3

Table 6: Training *vs.* testing methods for English. Columns represent different training methods and rows different testing methods. Each cell contains the standard dependency parsing metrics, *Accuracy/Root/Complete* as defined earlier.

	CZECH-A		
	k-best MIRA: Eisner	1-best MIRA: CLE	Factored MIRA
Eisner	83.3 / 88.6 / 31.3	83.6 / 88.2 / 29.9	83.9 / 87.5 / 30.3
CLE	75.0 / 86.3 / 17.6	84.1 / 89.0 / 32.2	84.4 / 88.3 / 32.3

Table 7: Training *vs.* testing methods for Czech.

D Additional Experimental Results

The systems we described in this paper have two degrees of freedom: learning and testing. The learning phase can take on any one of three values: k-best MIRA with projective Eisner parsing (we set $k = 5$), 1-best MIRA with non-projective Chu-Liu-Edmonds MST parsing (CLE) or factored MIRA. The test phase can take on one of two values: projective Eisner parsing or non-projective CLE parsing. Table 6 and Table 7 present the results for each system with the complete cross product of all the degrees of freedom for both languages.

The first interesting aspect of these results is that training and parsing using projective algorithms provides the best performance for English. This is not surprising at all since we know a priori that all trees in the English data set are projective. However, if we train using the non-projective parsing algorithm then at test time we can obtain reasonably good results using the non-projective parsing algorithm. This is worth noting since the result is a dependency parser for English that parses sentences in $O(n^2)$.

Furthermore, we can see that every system that is trained using projective parsing algorithms performs extremely bad when the test data is parsed using a non-projective algorithm. This also makes sense. During training, the parameters are set while never considering unlikely non-projective parses. So, if at testing we introduce these parses into the search space, the model will have seen no negative evidence to disregard them.