Non-projective Dependency Parsing using Spanning Tree Algorithms

Ryan McDonald **Fernando Pereira**
Department of Computer and Information Science
University of Pennsylvania
{ryantm,pereira}@cis.upenn.edu

Kiril Ribarov **Jan Hajič**
Institute of Formal and Applied Linguistics
Charles University
{ribarov,hajic}@ufal.ms.mff.cuni.cz

Abstract

We formalize weighted dependency parsing as searching for maximum spanning trees (MSTs) in directed graphs. Using this representation, the parsing algorithm of Eisner (1996) is sufficient for searching over all projective trees in $O(n^3)$ time. More surprisingly, the representation is extended naturally to non-projective parsing using Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) MST algorithm, yielding an $O(n^2)$ parsing algorithm. We evaluate these methods on the Prague Dependency Treebank using online large-margin learning techniques (Crammer et al., 2003; McDonald et al., 2005) and show that MST parsing increases efficiency and accuracy for languages with non-projective dependencies.

1 Introduction

Dependency parsing has seen a surge of interest lately for 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). The primary reasons for using dependency structures instead of more informative lexicalized phrase structures is that they are more efficient to learn and parse while still encoding much of the predicate-argument information needed in applications.

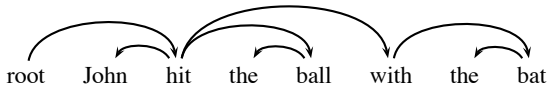


Figure 1: An example dependency tree.

Dependency representations, which link words to their arguments, have a long history (Hudson, 1984). Figure 1 shows a dependency tree for the sentence *John hit the ball with the bat*. We restrict ourselves to *dependency tree* analyses, in which each word depends on exactly one parent, either another word or a dummy root symbol as shown in the figure. The tree in Figure 1 is *projective*, meaning that if we put the words in their linear order, preceded by the root, the edges can be drawn above the words without crossings, or, equivalently, a word and its descendants form a contiguous substring of the sentence.

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) 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 an adverb. There is no way to draw the dependency tree for this sentence in the plane with no crossing edges, as 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 reliance on word order to express

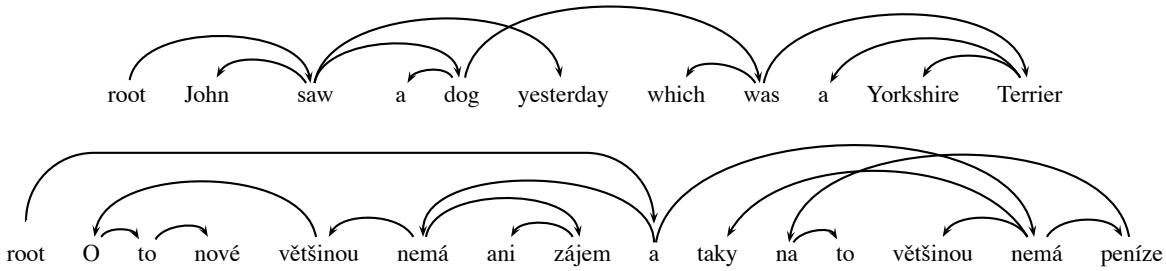


Figure 2: Non-projective dependency trees in English and Czech.

grammatical relations, allowing non-projective dependencies that we need to represent and parse efficiently. A non-projective example from the Czech Prague Dependency Treebank (Hajič et al.,) is also shown in Figure 2.

Most previous dependency parsing models have focused on projective trees, including the work of Eisner (1996), Collins et al. (1999), Yamada and Matsumoto (2003), Nivre and Scholz (2004), and McDonald et al. (2005). These systems have shown that accurate projective dependency parsers can be automatically learned from parsed data. However, non-projective analyses have recently attracted some interest, not only for languages with freer word order but also for English. In particular, Wang and Harper (2004) describe a broad coverage non-projective parser for English based on a hand-constructed constraint dependency grammar rich in lexical and syntactic information. 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 improved accuracy relative to a projective parser. Our approach differs from those earlier efforts in searching optimally and efficiently the full space of non-projective trees.

The main idea of our method is that dependency parsing can be formalized as the search for a maximum spanning tree in a directed graph. This formalization generalizes standard projective parsing models based on the Eisner algorithm (Eisner, 1996) to yield efficient $O(n^2)$ exact parsing methods for non-projective languages like Czech. Using this spanning tree representation, we extend the work of McDonald et al. (2005) on online large-margin discriminative training methods to non-projective dependencies.

The present work is related to that of Hirakawa (2001) who, like us, reduces the problem of dependency parsing to spanning tree search. However, his parsing method uses a branch and bound algorithm that is exponential in the worst case, even though it appears to perform reasonably in limited experiments. Furthermore, his work does not adequately address learning or measure parsing accuracy on held-out data.

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, in Section 4 we present parsing results for Czech. 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. We study untyped dependency trees mainly, but edge types can be added with simple extensions to the methods discussed here.

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 a common method of factoring the score of a dependency tree as the sum of the scores of all edges in the tree. In particular, we define the score of an edge to be the dot product be-

tween 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,

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

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

We represent the generic directed graph $G = (V,E)$ by its vertex set $V = \{v_1,\ldots,v_n\}$ and set $E \subseteq [1:n]\times[1:n]$ of pairs (i,j) of directed edges $v_i \rightarrow v_j$. Each such edge has a score $s(i,j)$. Since G is directed, $s(i,j)$ does not necessarily equal $s(j,i)$. A *maximum spanning tree* (MST) of G is a tree $\boldsymbol{y} \subseteq E$ that maximizes the value $\sum_{(i,j)\in\boldsymbol{y}} s(i,j)$ such that every vertex in V appears in $\boldsymbol{y}$. The maximum *projective* spanning tree of G is constructed similarly except that it can only contain projective edges relative to some total order 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 $G_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ given by

$$\begin{aligned} V_{\boldsymbol{x}} &= \{x_0 = \text{root}, x_1, \ldots, x_n\} \\ E_{\boldsymbol{x}} &= \{(i,j) : i \neq j, (i,j) \in [0:n]\times[1:n]\} \end{aligned}$$

That is, $G_{\boldsymbol{x}}$ is a graph with the sentence words and the dummy root symbol as vertices and a directed edge between every pair of distinct 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 be rooted at the dummy root and reach all the words in the sentence. Hence, finding a (projective) dependency tree with highest score is equivalent to finding a maximum (projective) spanning tree in $G_{\boldsymbol{x}}$.

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.

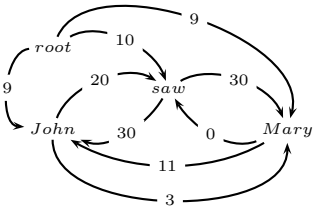
2.2.1 Non-projective Trees

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).

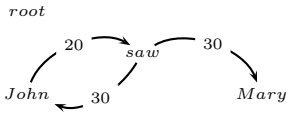
Efficient algorithms for the directed case are less well known, but they exist. We will use here the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967), sketched in Figure 3 following Georgiadis (2003). Informally, the algorithm has each vertex in the graph greedily select the incoming edge with highest weight. If a tree results, it 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 (Georgiadis, 2003). Hence the algorithm can recursively call itself on the new graph. Naively, this algorithm runs in $O(n^3)$ time since each recursive call takes $O(n^2)$ to find the highest incoming edge for each word and to contract the graph. There are at most $O(n)$ recursive calls since we cannot contract the graph more then n times. However,

Tarjan (1977) gives an efficient implementation of the algorithm with $O(n^2)$ time complexity for dense graphs, which is what we need here.

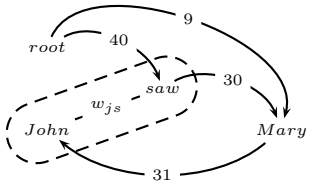
To find the highest scoring non-projective tree for a sentence, $\boldsymbol{x}$, we simply construct the graph $G_{\boldsymbol{x}}$ and run it through the Chu-Liu-Edmonds algorithm. The resulting spanning tree is the best non-projective dependency tree. We illustrate here the application of the Chu-Liu-Edmonds algorithm to dependency parsing on the simple example $\boldsymbol{x} =$ *John saw Mary*, with directed graph representation $G_{\boldsymbol{x}}$,



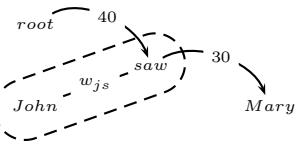
The first step of the algorithm is to find, for each word, the highest scoring incoming edge



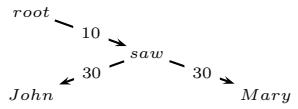
If the result were a tree, it would have to be the maximum spanning tree. However, in this case we have a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 3.



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 an MST in this graph can be transformed into an MST in the original graph (Georgiadis, 2003). 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 final (and correct) MST,



A possible concern with searching the entire space of spanning trees is that we have not used any 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 4.

2.2.2 Projective Trees

It is well known that projective dependency parsing using edge based factorization can be handled with the Eisner algorithm (Eisner, 1996). This algorithm has a runtime of $O(n^3)$ and has been employed successfully in both generative and discriminative parsing models (Eisner, 1996; McDonald et al., 2005). Furthermore, it is trivial to show that the Eisner algorithm solves the maximum projective spanning tree problem.

The Eisner algorithm differs significantly from the Chu-Liu-Edmonds algorithm. First of all, it is a bottom-up dynamic programming algorithm as opposed to a greedy recursive one. A bottom-up algorithm is necessary for the projective case since it must maintain the nested structural constraint, which is unnecessary for the non-projective case.

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 $\mathbf{w}$.

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, the resulting parsing algorithms are more efficient than lexicalized phrase structure approaches to dependency parsing, allowing us to search the entire space without any pruning. In particular the 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 easier since non-projective parsing does not need to enforce the non-crossing constraint of projective trees. As a result, non-projective parsing complexity is just $O(n^2)$, against the $O(n^3)$ complexity of the Eisner dynamic programming algorithm, which by construction enforces the non-crossing constraint.

3 Online Large Margin Learning

In this section, we review the work of McDonald et al. (2005) for online large-margin dependency parsing. 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 $\boldsymbol{y}_t$. In what follows, $\text{dt}(\boldsymbol{x})$ denotes the set of possible dependency trees for sentence $\boldsymbol{x}$.

The basic idea is to extend the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2003) to learning with structured outputs, in the present case dependency trees. Figure 4 gives pseudo-code for the MIRA algorithm as presented by McDonald et al. (2005). An online learning algorithm considers a single training instance at each update to $\mathbf{w}$. The auxiliary vector $\mathbf{v}$ accumulates the successive values 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 overfitting (Collins, 2002).

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. $\min \left\|\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}\right\|$
s.t. $s(\boldsymbol{x}_t, \boldsymbol{y}_t) - s(\boldsymbol{x}_t, \boldsymbol{y}') \geq L(\boldsymbol{y}_t, \boldsymbol{y}'), \forall \boldsymbol{y}' \in \text{dt}(\boldsymbol{x}_t)$
5. $\mathbf{v} = \mathbf{v} + \mathbf{w}^{(i+1)}$
6. $i = i + 1$
7. $\mathbf{w} = \mathbf{v}/(N * T)$

Figure 4: MIRA learning algorithm.

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. For dependency trees, the loss of a tree is defined to be the number of words with incorrect parents relative to the correct tree. This is closely related to the Hamming loss that is often used for sequences (Taskar et al., 2003).

For arbitrary inputs, there are typically exponentially many possible parses and thus exponentially many margin constraints in line 4 of Figure 4.

3.1 Single-best MIRA

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

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

McDonald et al. (2005) used a similar update with k constraints for the k highest-scoring trees, and showed that small values of k are sufficient to achieve the best accuracy for these methods. However, here we stay with a single best tree because k-best extensions to the Chu-Liu-Edmonds algorithm are too inefficient (Hou, 1996).

This model is related to the averaged perceptron algorithm of Collins (2002). In that algorithm, the single highest scoring tree (or structure) is used to update the weight vector. However, MIRA aggressively updates $\mathbf{w}$ to maximize the margin between

the correct tree and the highest scoring tree, which has been shown to lead to increased accuracy.

3.2 Factored MIRA

It is also possible to exploit the structure of the output space and factor the exponential number of margin constraints into a polynomial number of local constraints (Taskar et al., 2003; Taskar et al., 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. 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.

The factored constraints are in general more restrictive than the original constraints, so they may rule out the optimal solution to the original problem. McDonald et al. (2005) examines briefly factored MIRA for projective English dependency parsing, but for that application, k-best MIRA performs as well or better, and is much faster to train.

4 Experiments

We performed experiments on the Czech Prague Dependency Treebank (PDT) (Hajič, 1998; Hajič et al.,). We used the predefined training, development and testing split of this data set. Furthermore, we used the automatically generated POS tags that are provided with the data. Czech POS tags are very complex, consisting of a series of slots that may or may not be filled with some value. These slots represent lexical and grammatical 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 $13,450,672$, using the feature set outlined by McDonald et al. (2005).

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 edges 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. We compared the following systems:

1. **COLL1999:** The projective lexicalized phrase-structure parser of Collins et al. (1999).
2. **N&N2005:** The pseudo-projective parser of Nivre and Nilsson (2005).
3. **McD2005:** The projective parser of McDonald et al. (2005) that uses the Eisner algorithm for both training and testing. This system uses k-best MIRA with k=5.
4. **Single-best MIRA:** In this system we use the Chu-Liu-Edmonds algorithm to find the best dependency tree for Single-best MIRA training and testing.
5. **Factored MIRA:** Uses the quadratic set of constraints based on edge factorization as described in Section 3.2. We use the Chu-Liu-Edmonds algorithm to find the best tree for the test data.

4.1 Results

Results are shown in Table 1. There are two main metrics. The first and most widely recognized is *Accuracy*, which measures the number of words that correctly identified their parent in the tree. *Complete* measures the number of sentences in which the resulting tree was completely correct.

Clearly, there is an advantage in using the Chu-Liu-Edmonds algorithm for Czech dependency pars-

	Czech-A		Czech-B	
	Accuracy	**Complete**	**Accuracy**	**Complete**
COLL1999	82.8	-	-	-
N&N2005	80.0	31.8	-	-
McD2005	83.3	31.3	74.8	0.0
Single-best MIRA	84.1	32.2	81.0	**14.9**
Factored MIRA	**84.4**	**32.3**	**81.5**	14.3

Table 1: Dependency parsing results for Czech. Czech-B is the subset of Czech-A containing only sentences with at least one non-projective dependency.

ing. 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 dependency, 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, which in practice leads to improvements in parsing time. The results also show that in terms of *Accuracy*, factored MIRA performs better than single-best MIRA. However, for the factored model, we do have $O(n^2)$ margin constraints, which results in a significant increase in training time over single-best MIRA. Furthermore, we can also see that the MST parsers perform favorably compared to the more powerful lexicalized phrase-structure parsers, such as those presented by Collins et al. (1999) and Zeman (2004) that use expensive $O(n^5)$ parsing algorithms. We should note that the results in Collins et al. (1999) are different then reported here due to different training and testing data sets.

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 problem. This is because the model sets its weights with respect to the parsing algorithm and will disfavor features over unlikely non-projective edges.

Since the space of projective trees is a subset of the space of non-projective trees, it is natural to wonder how the Chu-Liu-Edmonds parsing algorithm performs on projective data since it is asymptotically better than the Eisner algorithm. Table 2 shows the results for English projective dependency trees extracted from the Penn Treebank (Marcus et al., 1993) using the rules of Yamada and Matsumoto (2003).

	English	
	Accuracy	**Complete**
McD2005	**90.9**	**37.5**
Single-best MIRA	90.2	33.2
Factored MIRA	90.2	32.3

Table 2: Dependency parsing results for English using spanning tree algorithms.

This shows that for projective data sets, training and testing with the Chu-Liu-Edmonds algorithm is worse than using the Eisner algorithm. This is not surprising since the Eisner algorithm uses the a priori knowledge that all trees are projective.

5 Discussion

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 online large-margin learning (McDonald et al., 2005) that empirically provides state-of-the-art performance for 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 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.

Our method requires a tree score that decomposes according to the edges of the dependency tree. One might hope that the method would generalize to

include features of larger substructures. Unfortunately, that would make the search for the best tree intractable (Höffgen, 1993).

Acknowledgments

We thank Lillian Lee for bringing an important missed connection to our attention, and Koby Crammer for his help with learning algorithms. This work has been supported by NSF ITR grants 0205448 and 0428193.

References

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

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

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. 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*.

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

J. Hajič, E. Hajicova, P. Pajas, J. Panevova, P. Sgall, and B. Vidova Hladka.

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*.

Klaus-U. Höffgen. 1993. Learning and robust learning of product distributions. In *Proceedings of COLT'93*, pages 77–83.

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

R. Hudson. 1984. *Word Grammar*. Blackwell.

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

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

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

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

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

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*.

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