Online Large-Margin Training of Dependency Parsers

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

Abstract

We present an effective training algorithm for linearly-scored dependency parsers that implements online large-margin multi-class training (Crammer and Singer, 2003; Crammer et al., 2003) on top of efficient parsing techniques for dependency trees (Eisner, 1996). The trained parsers achieve a competitive dependency accuracy for both English and Czech with no language specific enhancements.

1 Introduction

Research on training parsers from annotated data has for the most part 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). Generative parsing models are very convenient because training consists of computing probability estimates from counts of parsing events in the training set. However, generative models make complicated and poorly justified independence assumptions and estimations, so 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 parsing decisions in isolation and thus may suffer from the label bias problem (Lafferty et al., 2001).

Discriminatively trained parsers that score entire trees for a given sentence have only recently been investigated (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. 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.

Dependency trees are an alternative syntactic representation with a long history (Hudson, 1984). Dependency trees capture important aspects of functional relationships between words and have been shown to be useful in many applications including relation extraction (Culotta and Sorensen, 2004), paraphrase acquisition (Shinyama et al., 2002) and machine translation (Ding and Palmer, 2005). Yet, they can be parsed in $O(n^3)$ time (Eisner, 1996). Therefore, dependency parsing is a potential "sweet spot" that deserves investigation. We focus here on *projective* dependency trees in which a word is the parent of all of its arguments, and dependencies are non-crossing with respect to word order (see Figure 1). However, there are cases where crossing dependencies may occur, as is the case for Czech (Hajič, 1998). Edges in a dependency tree may be typed (for instance to indicate grammatical function). Though we focus on the simpler non-typed

Proceedings of the 43rd Annual Meeting of the ACL, pages 91–98,
Ann Arbor, June 2005.

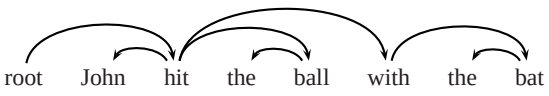

Figure 1: An example dependency tree.

case, all algorithms are easily extendible to typed structures.

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 the ability to label edges with semantic types. The accuracy of their parser is lower than that of Yamada and Matsumoto (2003).

We present a new approach to training dependency parsers, based on the online large-margin learning algorithms of Crammer and Singer (2003) and Crammer et al. (2003). Unlike the SVM parser of Yamada and Matsumoto (2003) and Ratnaparkhi's parser, our parsers are trained to maximize the accuracy of the overall tree.

Our approach is related to those of Collins and Roark (2004) and Taskar et al. (2004) for phrase structure parsing. Collins and Roark (2004) presented a linear parsing model trained with an averaged perceptron algorithm. However, to use parse features with sufficient history, their parsing algorithm must prune heuristically most of the 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 named-entity extraction and shallow parsing.

Besides simplicity, our method is efficient and accurate, as we demonstrate experimentally on English and Czech treebank data.

2 System Description

2.1 Definitions and Background

In what follows, the generic sentence is denoted by $\boldsymbol{x}$ (possibly subscripted); the ith word of $\boldsymbol{x}$ is denoted by x_i. The generic dependency tree is denoted by $\boldsymbol{y}$. If $\boldsymbol{y}$ is a dependency tree for sentence $\boldsymbol{x}$, we write $(i,j) \in \boldsymbol{y}$ to indicate that there is a directed edge from word x_i to word x_j in the tree, that is, x_i is the parent of x_j. $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$ denotes the training data.

We follow the edge based factorization method of Eisner (1996) and define the score of a dependency tree as the sum of the score of all edges in the tree,

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

where $\mathbf{f}(i,j)$ is a high-dimensional binary feature representation of the edge from x_i to x_j. For example, in the dependency tree of Figure 1, the following feature would have a value of 1:

$$f(i,j) = \begin{cases} 1 \text{ if } x_i\text{='hit' and } x_j\text{='ball'} \\ 0 \text{ otherwise.} \end{cases}$$

In general, any real-valued feature may be used, but we use binary features for simplicity. The feature weights in the weight vector $\mathbf{w}$ are the parameters that will be learned during training. Our training algorithms are iterative. We denote by $\mathbf{w}^{(i)}$ the weight vector after the i^{th} training iteration.

Finally we define $\text{dt}(\boldsymbol{x})$ as the set of possible dependency trees for the input sentence $\boldsymbol{x}$ and $\text{best}_k(\boldsymbol{x}; \mathbf{w})$ as the set of k dependency trees in $\text{dt}(\boldsymbol{x})$ that are given the highest scores by weight vector $\mathbf{w}$, with ties resolved by an arbitrary but fixed rule.

Three basic questions must be answered for models of this form: how to find the dependency tree $\boldsymbol{y}$ with highest score for sentence $\boldsymbol{x}$; how to learn an appropriate weight vector $\mathbf{w}$ from the training data; and finally, what feature representation $\mathbf{f}(i,j)$ should be used. The following sections address each of these questions.

2.2 Parsing Algorithm

Given a feature representation for edges and a weight vector $\mathbf{w}$, we seek the dependency tree or

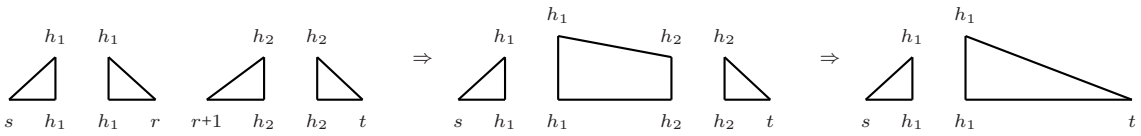

Figure 2: $O(n^3)$ algorithm of Eisner (1996), needs to keep 3 indices at any given stage.

trees that maximize the score function, $s(\boldsymbol{x}, \boldsymbol{y})$. The primary difficulty is that for a given sentence of length n there are exponentially many possible dependency trees. Using a slightly modified version of a lexicalized CKY chart parsing algorithm, it is possible to generate and represent these sentences in a forest that is $O(n^5)$ in size and takes $O(n^5)$ time to create.

Eisner (1996) made the observation that if the head of each chart item is on the left or right periphery, 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 2 shows the algorithm schematically. As with normal CKY parsing, larger elements are created bottom-up from pairs of smaller elements.

Eisner showed that his algorithm is sufficient for both searching the space of dependency parses and, with slight modification, finding the highest scoring tree $\boldsymbol{y}$ for a given sentence $\boldsymbol{x}$ under the edge factorization assumption. Eisner and Satta (1999) give a cubic algorithm for lexicalized phrase structures. However, it only works for a limited class of languages in which tree spines are regular. Furthermore, there is a large grammar constant, which is typically in the thousands for treebank parsers.

2.3 Online Learning

Figure 3 gives pseudo-code for the generic online learning setting. A single training instance is considered on each iteration, and parameters updated by applying an algorithm-specific update rule to the instance under consideration. The algorithm in Figure 3 returns an *averaged* weight vector: an auxiliary weight vector $\mathbf{v}$ is maintained that accumulates

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 3: Generic online learning algorithm.

the values of $\mathbf{w}$ after each iteration, and the returned weight vector is the average of all the weight vectors throughout training. Averaging has been shown to help reduce overfitting (Collins, 2002).

2.3.1 MIRA

Crammer and Singer (2001) developed a natural method for 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 tree $\boldsymbol{y}'$ relative to the correct tree $\boldsymbol{y}$. We define the loss of a dependency tree as the number of words that have the incorrect parent. Thus, the largest loss a dependency tree can have is the length of the sentence.

Informally, this update looks to create a margin between the correct dependency tree and each incorrect dependency tree at least as large as the loss of the incorrect tree. The more errors a tree has, the farther away its score will be from the score of the correct tree. In order to avoid a blow-up in the norm of the weight vector we minimize it subject to constraints that enforce the desired margin between the correct and incorrect trees[1].

[1]The constraints may be unsatisfiable, in which case we can relax them with slack variables as in SVM training.

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 norm of the change to the parameter vector as small as possible, subject to correctly classifying the instance under consideration with a margin at least as large as the loss of the incorrect classifications. This can be formalized by substituting the following update into line 4 of the generic online algorithm,

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

This is a standard quadratic programming problem that can be easily 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. In equation (1), $s(\boldsymbol{x}, \boldsymbol{y})$ is calculated with respect to the weight vector after optimization, $\mathbf{w}^{(i+1)}$.

To apply MIRA to dependency parsing, we can simply see parsing as a multi-class classification problem in which each dependency tree is one of many possible classes for a sentence. However, that interpretation fails computationally because a general sentence has exponentially many possible dependency trees and thus exponentially many margin constraints.

To circumvent this problem we make the assumption that the constraints that matter for large margin optimization are those involving the incorrect trees $\boldsymbol{y}'$ with the highest scores $s(\boldsymbol{x}, \boldsymbol{y}')$. The resulting optimization made by MIRA (see Figure 3, line 4) would then be:

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

reducing the number of constraints to the constant k. We tested various values of k on a development data set and found that small values of k are sufficient to achieve close to best performance, justifying our assumption. In fact, as k grew we began to observe a slight degradation of performance, indicating some overfitting to the training data. All the experiments presented here use $k = 5$. The Eisner (1996) algorithm can be modified to find the k-best trees while only adding an additional $O(k \log k)$ factor to the runtime (Huang and Chiang, 2005).

A more common approach is to factor the structure of the output space to yield a polynomial set of local constraints (Taskar et al., 2003; Taskar et al., 2004). One such factorization for dependency trees is

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

It is trivial to show that if these $O(n^2)$ constraints are satisfied, then so are those in (1). We implemented this model, but found that the required training time was much larger than the k-best formulation and typically did not improve performance. Furthermore, the k-best formulation is more flexible with respect to the loss function since it does not assume the loss function can be factored into a sum of terms for each dependency.

2.4 Feature Set

Finally, we need a suitable feature representation $\mathbf{f}(i, j)$ for each dependency. The basic features in our model are outlined in Table 1a and b. All features are conjoined with the direction of attachment as well as the distance between the two words being attached. These features represent a system of backoff from very specific features over words and part-of-speech tags to less sparse features over just part-of-speech 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 the parent-child node pairs in the tree was not enough for high accuracy, because all attachment decisions were made outside of the context in which the words occurred. To solve this problem, we added two other types of features, which can be seen in Table 1c. Features of the first type look at words that occur between a child and its parent. These features take the form of a POS trigram: the POS of the parent, of the child, and of a word in between, for all words linearly between the parent and the child. This feature was particularly helpful for nouns identifying their parent, since

a)

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

b)

Basic Big-ram 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 node in dependency tree. c-word: word of child node. p-pos: POS of parent node. c-pos: POS of child node. 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 nodes.

it would typically rule out situations when a noun attached to another noun with a verb in between, which is a very uncommon phenomenon.

The second type of feature provides the local context of the attachment, that is, the words before and after the parent-child pair. This feature took the form of a POS 4-gram: The POS of the parent, child, word before/after parent and word before/after child. The system also used back-off features to various trigrams where one of the local context POS tags was removed. Adding these two features resulted in a large improvement in performance and brought the system to state-of-the-art accuracy.

2.5 System Summary

Besides performance (see Section 3), the approach to dependency parsing we described has several other advantages. The system is very general and contains no language specific enhancements. In fact, the results we report for English and Czech use identical features, though are obviously trained on different data. The online learning algorithms themselves are intuitive and easy to implement.

The efficient $O(n^3)$ parsing algorithm of Eisner allows the system to search the entire space of dependency trees while parsing thousands of sentences in a few minutes, which is crucial for discriminative training. We compare the speed of our model to a standard lexicalized phrase structure parser in Section 3.1 and show a significant improvement in parsing times on the testing data.

The major limiting factor of the system is its restriction to features over single dependency attachments. Often, when determining the next dependent for a word, it would be useful to know previous attachment decisions and incorporate these into the features. It is fairly straightforward to modify the parsing algorithm to store previous attachments. However, any modification would result in an asymptotic increase in parsing complexity.

3 Experiments

We tested our methods experimentally on the English Penn Treebank (Marcus et al., 1993) and on the Czech Prague Dependency Treebank (Hajič, 1998). All experiments were run on a dual 64-bit AMD Opteron 2.4GHz processor.

To create dependency structures from the Penn Treebank, we used the extraction rules of Yamada and Matsumoto (2003), which are an approximation to the lexicalization rules of Collins (1999). We split the data into three parts: sections 02-21 for training, section 22 for development and section 23 for evaluation. Currently the system has $6,998,447$ features. Each instance only uses a tiny fraction of these features making sparse vector calculations possible. Our system assumes POS tags as input and uses the tagger of Ratnaparkhi (1996) to provide tags for the development and evaluation sets.

Table 2 shows the performance of the systems that were compared. Y&M2003 is the SVM-shift-reduce parsing model of Yamada and Matsumoto (2003), N&S2004 is the memory-based learner of Nivre and Scholz (2004) and MIRA is the the system we have described. We also implemented an averaged perceptron system (Collins, 2002) (another online learning algorithm) for comparison. This table compares only *pure* dependency parsers that do

	English			Czech		
	Accuracy	**Root**	**Complete**	**Accuracy**	**Root**	**Complete**
Y&M2003	90.3	91.6	38.4	-	-	-
N&S2004	87.3	84.3	30.4	-	-	-
Avg. Perceptron	90.6	94.0	36.5	82.9	88.0	30.3
MIRA	90.9	94.2	37.5	83.3	88.6	31.3

Table 2: Dependency parsing results for English and Czech. *Accuracy* is the number of words that correctly identified their parent in the tree. *Root* is the number of trees in which the root word was correctly identified. For Czech this is f-measure since a sentence may have multiple roots. *Complete* is the number of sentences for which the entire dependency tree was correct.

not exploit phrase structure. We ensured that the gold standard dependencies of all systems compared were identical.

Table 2 shows that the model described here performs as well or better than previous comparable systems, including that of Yamada and Matsumoto (2003). Their method has the potential advantage that SVM batch training takes into account all of the constraints from all training instances in the optimization, whereas online training only considers constraints from one instance at a time. However, they are fundamentally limited by their approximate search algorithm. In contrast, our system searches the entire space of dependency trees and most likely benefits greatly from this. This difference is amplified when looking at the percentage of trees that correctly identify the root word. The models that search the entire space will not suffer from bad approximations made early in the search and thus are more likely to identify the correct root, whereas the approximate algorithms are prone to error propagation, which culminates with attachment decisions at the top of the tree. When comparing the two online learning models, it can be seen that MIRA outperforms the averaged perceptron method. This difference is statistically significant, $p < 0.005$ (McNemar test on head selection accuracy).

In our Czech experiments, we used the dependency trees annotated in the Prague Treebank, and the predefined training, development and evaluation sections of this data. The number of sentences in this data set is nearly twice that of the English treebank, leading to a very large number of features — $13,450,672$. But again, each instance uses just a handful of these features. For POS tags we used the automatically generated tags in the data set. Though we made no language specific model changes, we did need to make some data specific changes. In particular, we used the method of Collins et al. (1999) to simplify part-of-speech tags since the rich tags used by Czech would have led to a large but rarely seen set of POS features.

The model based on MIRA also performs well on Czech, again slightly outperforming averaged perceptron. Unfortunately, we do not know of any other parsing systems tested on the same data set. The Czech parser of Collins et al. (1999) was run on a different data set and most other dependency parsers are evaluated using English. Learning a model from the Czech training data is somewhat problematic since it contains some crossing dependencies which cannot be parsed by the Eisner algorithm. One trick is to rearrange the words in the training set so that all trees are nested. 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.

3.1 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 of 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 sys-

	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 3: 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.

tem *Collins-rules*. Table 3 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 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 actually 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 run-time, our system is a huge improvement over the Collins parser.

The final system in Table 3 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.

	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 4: Evaluation of k-best MIRA approximation.

3.2 k-best MIRA Approximation

One question that can be asked is how justifiable is the k-best MIRA approximation. Table 4 indicates the accuracy on testing and the time it took to train models with $k = 1, 2, 5, 10, 20$ for the English data set. Even though the parsing algorithm is proportional to $O(k \log k)$, empirically, the training times scale linearly with k. Peak performance is achieved very early 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 proportional to their loss.

4 Summary

We described a successful new method for training dependency parsers. We use simple linear parsing models trained with margin-sensitive online training algorithms, achieving state-of-the-art performance with relatively modest training times and no need for pruning heuristics. We evaluated the system on both English and Czech data to display state-of-the-art performance without any language specific enhancements. Furthermore, the model can be augmented to include features over lexicalized phrase structure parsing decisions to increase dependency accuracy over those parsers.

We plan on extending our parser in two ways. First, we would add labels to dependencies to represent grammatical roles. Those labels are very important for using parser output in tasks like information extraction or machine translation. Second,

we are looking at model extensions to allow non-projective dependencies, which occur in languages such as Czech, German and Dutch.

Acknowledgments: We thank Jan Hajič for answering queries on the Prague treebank, and Joakim Nivre for providing the Yamada and Matsumoto (2003) head rules for English that allowed for a direct comparison with our systems. This work was supported by NSF ITR grants 0205456, 0205448, and 0428193.

References

D.M. Bikel. 2004. Intricacies of Collins parsing model. *Computational Linguistics*.

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*.

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*.

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*.

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. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proc. ACL*.

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

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

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*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*.

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*.

A. Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*.

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. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. HLT-NAACL*.

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

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*.

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