DISCRIMINATIVE LEARNING AND SPANNING TREE ALGORITHMS FOR DEPENDENCY PARSING

Ryan McDonald

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2006

Supervisor of Dissertation

Graduate Group Chair

Acknowledgements

First and foremost I would like to thank my advisor Fernando Pereira. His contribution to all the sections of this thesis is immeasurable. While at Penn I was very fortunate to work with Fernando and to be the beneficiary of his vast knowledge and friendly persona.

I am of course grateful to my excellent thesis committee: Eugene Charniak, Aravind Joshi, Mark Liberman, and Mitch Marcus. I am also indebted to Gerald Penn for fostering my interest in natural language processing and giving me my first opportunity to engage in research.

I would like to acknowledge my co-authors on work that has contributed largely to this thesis. In particular, Koby Crammer and I jointly developed material in Chapter 2 and Chapter 3. Kiril Ribarov and Jan Hajič submitted a joint paper with myself and Fernando Pereira that covers Kiril's and my independent formulation of dependency parsing as finding maximum spanning trees, which can be found in Chapter 3. Kevin Lerman helped with the labeled multilingual dependency experiments described in Chapter 5. Thanks also to John Blitzer, Nikhil Dinesh, Kevin Lerman and Nick Montfort for helping with the evaluation in Chapter 9 and to Daniel Marcu for providing data sets.

Discussions with Joakim Nivre, Hal Daumé, Jason Eisner, Charles Sutton, Andrew McCallum, Libin Shen, Julia Hockenmaier and Mike Collins helped form many of the ideas presented here. At Penn I had the fortune of spending many hours at the office (and evenings at the food trucks) waxing poetic with John Blitzer, Nikhil Dinesh, Yuan Ding, and eventually Liang Huang. I am also grateful to Anne, Christie, Ameesh, Kilian, Andrew, Kimia, Norm, Sasha, and most importantly Tania, for making Philadelphia an enjoyable place to spend my graduate studies.

Finally I would like to thank my family. My mom and dad provided an enormous amount of support and convinced me to stick with school as long as I could. My brother Neill has always inspired me to better myself and it was through his example that I decided to pursue my doctoral studies. I dedicate this thesis to them.

ABSTRACT

DISCRIMINATIVE LEARNING AND SPANNING TREE ALGORITHMS FOR DEPENDENCY PARSING

Ryan McDonald

Supervisor: Fernando Pereira

In this thesis we develop a discriminative learning method for dependency parsing using online large-margin training combined with spanning tree inference algorithms. We will show that this method provides state-of-the-art accuracy, is extensible through the feature set and can be implemented efficiently. Furthermore, we display the language independent nature of the method by evaluating it on over a dozen diverse languages as well as show its practical applicability through integration into a sentence compression system.

We start by presenting an online large-margin learning framework that is a generalization of the work of Crammer and Singer [34, 37] to structured outputs, such as sequences and parse trees. This will lead to the heart of this thesis – discriminative dependency parsing. Here we will formulate dependency parsing in a spanning tree framework, yielding efficient parsing algorithms for both projective and non-projective tree structures. We will then extend the parsing algorithm to incorporate features over larger substructures without an increase in computational complexity for the projective case. Unfortunately, the non-projective problem then becomes NP-hard so we provide structurally motivated approximate algorithms. Having defined a set of parsing algorithms, we will also define a rich feature set and train various parsers using the online large-margin learning framework. We then compare our trained dependency parsers to other state-of-the-art parsers on 14 diverse languages: Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, English, German, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish.

Having built an efficient and accurate discriminative dependency parser, this thesis will then turn to improving and applying the parser. First we will show how additional resources can provide useful features to increase parsing accuracy and to adapt parsers to new domains. We will also argue that the robustness of discriminative inference-based learning algorithms lend themselves well to dependency parsing when feature representations or structural constraints do not allow for tractable parsing algorithms. Finally, we integrate our parsing models into a state-of-the-art sentence compression system to show its applicability to a real world problem.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Part of the material in this chapter has been drawn from [91, 95].

1.1 General Parsing Framework

Computational methods for the syntactic parsing of sentences have been at the forefront of natural language processing research since its inception [74, 42, 70, 25, 16]. There are many questions one must address when conducting research on syntactic parsing. Amongst them are, *What formalism will be used? How will a parser be constructed? How will new sentences be parsed to produce their syntactic representations?* In Section 1.3 we describe *dependency graphs*, which is the syntactic representation of sentences we will be concerned with throughout this work.

To answer the latter questions, Figure 1.1 graphically displays the framework we will assume. First, a system must define a *learning algorithm* that takes as input the *training data*, which is a parsed set of sentences, and outputs a *parsing model*. This process of a learning algorithm producing a parsing model from a training set is usually called *training* or *learning*. The parsing model (sometimes simply called the *model*) contains the parameter

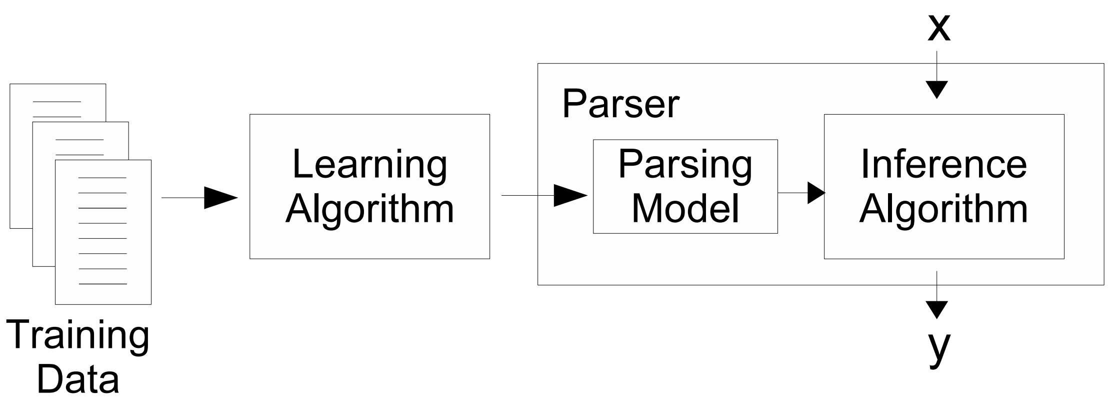


Figure 1.1: Outline of generic syntactic parsing framework.

settings as well as any feature specifications. The learning algorithm is generic and will produce different parsing models when different training data is given as input. In fact, we will show empirically that the learning algorithms presented in this work are language independent. That is, if given training data in English, the learning algorithm will produce an English parsing model. Similarly, if given training data in Spanish, it will produce a Spanish parsing model. The class of learning algorithms used in this work are described in Chapter 2.

The learned parsing model is part of the *parser*. The parser consists of both the model and an *inference algorithm* (or *parsing algorithm*), which specifies how to use the model for parsing. That is, when a new sentence $\boldsymbol{x}$ is given to the parser, the inference algorithm uses the parameter specifications in the model to produce a syntactic representation $\boldsymbol{y}$. In many cases, the parsing model defines the inference algorithm. For example, if the model is a Probabilistic Context Free Grammar, then the inference algorithm will most likely by CKY [152] or Earley's [42]. Both the parsing models and corresponding inference algorithms are described in Chapter 3.

1.2 Discriminative Learning and Parsing

Most recent work on producing parsers from annotated data has focused on models and learning 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 $\boldsymbol{y}$ [16, 25]. 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 statistical independence assumptions, and training does not optimize a criterion directly related to parsing accuracy. Therefore, we might expect better accuracies from discriminatively trained models that set their parameters typically by minimizing the conditional log-loss or error rate of the model on the training data. Furthermore, discriminative models can easily handle millions of rich dependent features necessary to successfully disambiguate many natural language phenomena – a feat that is computationally infeasible in generative models. The advantages of discriminative learning have been exploited before, most notably in information extraction where discriminative models represent the standard for both entity extraction [142] and relation extraction [153]. The obvious question the parsing community has asked is, *can the benefits of discriminative learning be applied to parsing?*

Perhaps the earliest work on discriminative parsing is the local decision maximum entropy model of Ratnaparkhi [111], which is trained to maximize the conditional likelihood of each parsing decision within a shift-reduced parsing algorithm. This system 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 [80]. A similar system was proposed by Henderson [63] that was trained using neural networks.

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 [24, 30, 113, 138]. 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 [5], the perceptron algorithm [116], or support vector machines [10], which we call here *linear parsing models* because they all score a parse $\boldsymbol{y}$ for a sentence $\boldsymbol{x}$ 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 [25] with $O(n^5)$ parsing complexity. The prohibitive cost of training a global discriminative phrase-structure parser results in most systems employing aggressive pruning and other heuristics to make training tractable. Consequently, these systems have failed to convincingly outperform the standard generative parsers of Charniak [16] and Collins [25].

Another line of discriminative parsing research is parse re-ranking, which attempts to alleviate any computational problems by taking the k-best outputs from a generative parsing model and training a post processing ranker to distinguish the correct parse from all others. The advantage of re-ranking is that it reduces parsing to a smaller multi-class classification problem that allows the classifier to condition on rich features spanning the entire structure of each parse. This approach has been applied to both the Collins parser [28] and the Charniak parser [17] and typically results in a 10% relative reduction in error.

1.3 Dependency Parsing

In this work, we wish to explore central questions in discriminative parsing. To do this we turn to dependency representations of natural language syntactic structure. A primary reason for using dependency representations 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 applications. As a result, dependency parsing has seen a surge of interest lately in applications such as relation extraction [38], machine translation [40], synonym generation [122] and lexical resource augmentation [130]. Thus, dependency parsing represents a syntactic formalism whose computational complexity will allow us to explore discriminative training to its fullest, while at the same time providing a usable representation of language for many natural language processing tasks.

Another advantage of dependency parsers is the existence of numerous large annotated resources. The Prague Dependency Treebank [56, 57] contains tens of thousands of human annotated dependency representations for Czech. The Nordic Treebank Network [100] is a group of European researchers that have developed many tools for dependency parsing including treebanks for Danish [79] and Swedish [44]. There are also Turkish [107] and Arabic [58] dependency treebanks available. Recently, the organizers of the shared-task at CoNLL 2006 [13] standardized data sets for 13 languages: Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish [58, 124, 123, 19, 9, 79, 148, 11, 73, 2, 41, 23, 102, 107, 4]. Furthermore, most phrase-structure treebanks typically have common tools for converting them into dependency treebanks including both the English and Chinese Penn treebanks [84, 150].

1.3.1 Formal Definition

Dependency graphs represent words and their relationship to syntactic modifiers using directed edges. Figure 1.2 shows a dependency graph for the sentence, *John hit the ball with the bat*. This example belongs to the special class of dependency graphs that only contain projective (also known as nested or non-crossing) edges. Assuming a unique root as the left most word in the sentence, a projective graph is one that can be written with all words

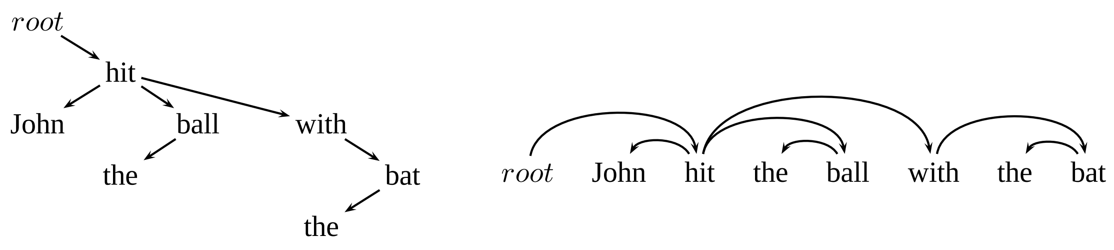


Figure 1.2: An example dependency graph.

in a predefined linear order and all edges drawn on the plane above the sentence, with no edge crossing another. Figure 1.2 shows this construction for the example sentence. Equivalently, we can say a dependency graph is projective if and only if an edge from word w to word u implies that there exists a directed path in the graph from w to every word between w and u in the sentence.

Due to English's rigid word order, projective graphs are sufficient to analyze most English sentences. In fact, the largest source of English dependencies is automatically generated from the Penn Treebank [84] and is by construction exclusively projective [151]. However, there are certain examples in which a non-projective graph 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 noun it modifies (the *dog*) are separated by a temporal modifier of the main verb. There is no way to draw the dependency graph for this sentence in the plane with no crossing edges, as illustrated in Figure 1.3. In languages with flexible word, such as Czech, Dutch and German, 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.

Formally, a dependency structure for a given sentence is a directed graph originating out of a unique and artificially inserted $root$ node, which we always insert as the left most word.

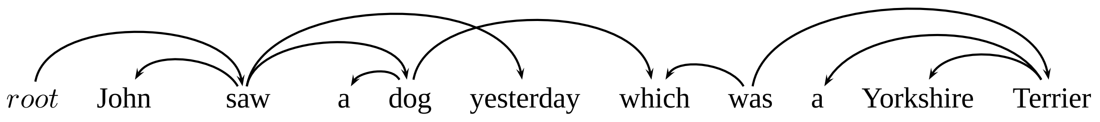


Figure 1.3: A non-projective dependency graph.

In the most common case, every valid dependency graph has the following properties,

1. It is weakly connected (in the directed sense).
2. Each word has exactly one incoming edge in the graph (except the root, which has no incoming edge).
3. There are no cycles.
4. If there are n words in the sentence (including $root$), then the graph has exactly $n-1$ edges.

It is easy to show that 1 and 2 imply 3, and that 2 implies 4. In particular, a dependency graph that satisfies these constraints must be a tree. Thus we will say that dependency graphs satisfying these properties satisfy the *tree constraint*, and call such graphs *dependency trees*. For most of this work we will be addressing the problem of parsing dependency graphs that are trees, which is a common constraint [103]. In Section 8.2, we address the problem of more general dependency graphs.

Directed edges in a dependency graph represent words and their modifiers, e.g., a verb and its subject, a noun and a modifying adjective, etc. The word constitutes the *head* of the edge and the argument the *modifier*. This relationship is often called the *head-modifier* or the *governor-dependent* relationship. The head is also sometimes called the parent and the modifier is also sometimes called the child or argument. We will always refer to words in a dependency relationship as the head and modifier.

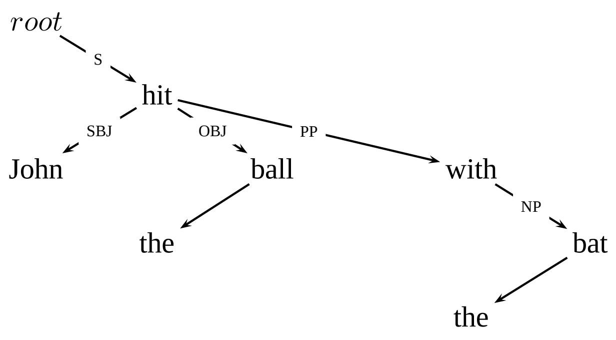


Figure 1.4: An example of a labeled dependency graph.

Dependency structures can be labeled to indicate grammatical, syntactic and even semantic properties of the head-modifier relationships in the graph. For instance, we can add syntactic/grammatical function labels to the structure in Figure 1.2 to produce the graph in Figure 1.4.

We should note that we are interested in producing *syntactic* dependencies in this work and not semantic ones. A distinction that will be made clearer in the next section.

1.3.2 A Brief History

Dependency grammars and dependency parsing have a long history in both the formal linguistic and computational linguistic communities. A common starting point on modern linguistic theories of dependency representations is that of Tesnière [140] who, in 1959, wrote:

> *The sentence is an organized whole, the constituent elements of which are words. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence. The structural connections establish dependency relations between the words. Each connection in principle unites a superior term and an inferior term. The superior term receives the name [head]. The inferior*

receives the name [modifier]. [Translation by Joakim Nivre [103]]

The definition given earlier for dependency graphs is clearly evident in this passage. After the work of Tesnière there was a number of studies in the theoretic linguistics community on dependency representations and their relationships to other formalisms, most notably by Hays [61] and Gaifman [53]. However, it would be another quarter of a century before dependency representations of sentences became wide spread in the computational linguistics community. Perhaps the two most well known works in this respect are Hudson's Word Grammar [67] and Mel′čuk's Meaning Text Theory [96]. Since then, a variety of computational syntactic dependency formalisms have been proposed. Most notable amongst them is the work on constraint based dependency parsing [85], which treats the parsing of dependency graphs as a constraint satisfaction problem. This framework has been extended theoretically [85, 60] as well as applied in practical evaluations [51, 149], providing some of the best empirical support for any grammar-based dependency formalism. Another important framework is Functional Generative Description [119], which provides the core theoretical foundation for the Prague Dependency Treebank [9] – the largest dependency treebank currently in use. Finally, work on context-sensitive formalisms such as those in the TAG family [70] or CCGs [131] can also be viewed as producing dependency graphs of sentences through their derivation trees. However, these trees typically represent semantic dependencies, not syntactic ones.

When constructing a dependency graph there are many issues one must address. Chief amongst them is the definition of the *head* and *modifier* in a relation. Some classes of relations are relatively easy to define. For instance, it seems clear that both subjects and objects are modifying a verb (or sets of verbs). Similarly, adjectives and adverbials play the obvious role of modifier. However, what about prepositions or relative clauses? Does the preposition/complementizer govern the noun/verb? Vice-versa? In this case it is often the word whose identity determines the syntactic category of the subgraph that is chosen as the head (i.e., the preposition/complementizer). There is disagreement on this issue, in particular in the empirical parsing community where data sparseness must be traded off against informative features. Treating a preposition as the head would result in less sparse parameter estimation. However, having the noun as the head would naturally provide more information.

Most theories assume a distinction between various levels of dependency representation. Meaning Text Theory argues that there are essentially three layers of representation, the morphological, syntactic and semantic. Similarly, the Functional Generative Description framework assumes both syntactic and semantic layers. This distinction can be beneficial when determining the head-modifier relationship. For instance, at the syntactic level, the preposition would govern the noun since it is the preposition that determines the syntactic category of the relationship with the verb. However, at the semantic level the opposite is true since it is the noun that is filling the semantic template of the verb.

Another important question is whether one word may modify two words in a dependency graph, which would break the tree constraint. For example, Hudson's word grammar allows for this when dealing with conjunctions or relative clause constructions. However, the common argument against this representation is that such phenomena should be represented in the semantic dependency graph and not the syntactic.

Nivre [103] presents a more detailed discussion on the history and aspects of formal dependency parsing, on which this section has been based.

1.3.3 Data-Driven Dependency Parsing

The formal properties of dependency-based representations depend entirely on the underlying grammar that produces them. For instance, many dependency grammars that enforce projectivity are easily shown to be context-free [53, 127], whereas constraint dependency grammars [60] and discontinuous grammars [78] have been shown to be formally more powerful in terms of strong generative capacity since they can model non-projectivity.

In this work we focus on parsing models that discriminate between better and worse parses for a given input sentence[1]. Thus, there is no notion of a parser accepting the language $a^n b^n$ or $a^n b^n a^n b^n$. In fact, our parser uses a grammar that accepts the set of all possible strings. The goal of parsing will be to search the set of all valid structures and return the structure with highest score – it is given that the sentence under consideration should be accepted. The Collins parser [25] is a well known model of this form. It searches the entire space of phrase-structures for a given sentence without the use of an underlying grammar. For dependency parsing, this translates to searching the space of projective or non-projective trees and returning the most likely one. This form of parsing is often referred to as *data-driven parsing*, since parsing decisions are made based on models trained on annotated data alone without an underlying grammar. Note also that this relieves us of making any of the difficult decisions about the nature of the head-modifier relationship discussed in the last section since we assume this information is contained implicitly in the annotated data.

The data driven view of parsing is also application-oriented. We are concerned with finding syntactic representations for sentences that will provide a relevant structure for further processing in information extraction, machine translation or other common language processing applications. However, even in the application view of parsing there are instances when the ability to accept a sentence under some linguistically motivated grammar is beneficial. Most notably, the language generation problem, which arises in translation and summarization, requires a measurement of a sentence's linguistic plausibility. It is possible to block certain analyses within our parsing framework by artificially assigning them a score of $-\infty$. However, we can only restrict parsing decisions within the local context that the parser considers while searching. This is also true of CFG-based phrase-structure

[1]In fact the parsing models discussed in this work really provide a mechanism for ranking parses.

parsers.

1.4 Comparison to Other Work

Our discriminative learning algorithms for dependency parsing are closely related to those of Collins and Roark [30] and Taskar et al. [138] for phrase-structure parsing. Collins and Roark [30] 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. [138] formulate the parsing problem in the large-margin structured classification setting [137], 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 this work. Eisner [45] gave a generative model with a cubic parsing algorithm based on a graph factorization that very much inspired the core parsing algorithms for this work. Yamada and Matsumoto [151] trained support vector machines (SVM) to make parsing decisions in a shift-reduce dependency parser for English. As in Ratnaparkhi's parser [111], the classifiers are trained on individual decisions rather than on the overall quality of the parse. Nivre and Scholz [105] developed a memory-based learning model combined with a linear-time parser to approximately search the space of possible parses. A significant amount of work has been done by the researchers at Charles University led by Jan Hajič and Eva Hajičová. In addition to developing the Prague Dependency Treebank [56], there has also been extensive research on parsing Czech at that institution [29, 112, 154].

One interesting class of dependency parsers are those that provide labels on edges.

Two well known parsers in this class are the link-grammar system of Sleator and Temperly [127] and the system of Lin [82]. Nivre and Scholz [105] provide two systems, one a pure dependency parser and the other a labeled model that labels edges with syntactic categories. Wang and Harper [149] provide a rich dependency model with complex edge labels containing an abundant amount of lexical and syntactic information drawn from a treebank. Though we focus primarily on unlabeled dependency graphs, we also describe simple extensions to our models that allow for the inclusion of labels.

Previous attempts at broad coverage dependency parsing have primarily dealt with projective constructions. In particular, the supervised approaches of Yamada and Matsumoto [151] and Nivre and Scholz [105] have provided the previous best results for projective dependency parsing. Another source of dependency parsers are lexicalized phrase-structure parsers with the ability to output dependency information [16, 25, 151]. These systems are based on finding phrase structure through nested chart parsing algorithms and cannot model non-projective edges tractably. However, Yamada and Matsumoto [151] 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, tractable inference algorithms have been given by Tapanainen and Järvinen [134] and Kahane et al. [71]. Nivre and Nilsson [104] presented a broad-coverage 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 [149] for English, which presents very good results using a constraint dependency grammar framework that is rich in lexical and syntactic information. One aspect of previous attempts at non-projective parsing is that inference algorithms are typically approximate. A commonly cited result is the proof by Neuhaus and Bröker [101] that non-projective parsing is NP-hard. However, this result assumes the existence of a particular grammar generating the language. In this study we are working within the data driven framework and we will show that this theoretical result does not apply.

The present work is closely related to that of Hirakawa [65] 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 due to non-local parsing constraints and is still in the worst case exponential (though in small scale experiments seems tractable). Furthermore, no justification was provided for the empirical adequacy of equating spanning trees with dependency trees.

The closely related research of Ribarov [112] was developed independently of this work[2]. In that 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 [116], which is a learning algorithm related to the framework presented in Chapter 2. However, his empirical evaluation on the Prague Dependency Treebank [56] 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 morphology and does not consider lexical or contextual information. We also extend the dependency parsing as maximum spanning tree framework to consider trees with larger (and possibly intractable) feature contexts as well as apply the resulting parser to new domains and in real world applications.

1.5 Thesis

In this thesis we develop a discriminative learning method for dependency parsing using online large-margin training combined with spanning tree inference algorithms. We will

[2]Fortunately we were able to make a joint presentation on our work to HLT-EMNLP 2005 [95].

show that this method provides state-of-the-art accuracy, is extensible through the feature set and can be implemented efficiently. Furthermore, we display the language independent nature of the method by evaluating it on over a dozen diverse languages as well as show its practical applicability through integration into a sentence compression system.

We start by presenting an online large-margin learning framework that is a generalization of the work of Crammer and Singer [34, 37] to structured outputs, such as sequences and parse trees. This is a large-margin perceptron-like learning technique which reduces the learning task to one of inference. We argue that this technique is intuitive, flexible, efficient and performs competitively with other discriminative learning algorithms. We display this empirically on a number of standard NLP data sets.

This will lead to the heart of this thesis – discriminative dependency parsing. Here we will formulate dependency parsing in a spanning tree framework, yielding efficient parsing algorithms for both projective and non-projective tree structures. We will then extend the parsing algorithm to incorporate features over larger substructures without an increase in computational complexity for the projective case. Unfortunately, the non-projective problem then becomes NP-hard so we provide an approximate algorithm which is motivated from the knowledge that, in practice, non-projective trees can typically be converted to a projective tree using only a small number of edge transformations. Having defined a set of parsing algorithms, we will also define a rich feature set and train various parsers using the online large-margin learning framework. We then compare our trained dependency parsers to other state-of-the-art parsers on English, Czech and Chinese data sets to show that our discriminative model provides efficient parsing coupled with high accuracy. Furthermore, we show how to extend the models to include syntactic edge labels and present additional detailed results for English. One advantage of our parsing models is that they rely on little to no language specific optimizations. To show the language independence of our parsing models we further evaluate its parsing accuracy on 14 diverse languages: Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, English, German, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish.

Having built an efficient and accurate discriminative dependency parser, this thesis will then turn to improving and applying the parser. First we will show how additional resources can provide useful features to increase parsing accuracy and to adapt parsers to new domains. In particular, we will display that features defined on the output of in and out of domain parsers can improve accuracy considerably. We will also argue that the robustness of discriminative inference-based learning algorithms lend themselves well to dependency parsing when feature representations or structural constraints do not allow for tractable parsing algorithms. Finally, we integrate our parsing models into a state-of-the-art sentence compression system to show its applicability to a real world problem.

It is important to note that this thesis will not argue for dependency representations of language from a linguistic perspective. We focus on dependency representations primarily for empirical and didactic reasons: dependency structures have been shown to be useful for many language processing tasks and their computational properties allow us to explore discriminative parsing to its fullest.

1.6 Summary of Document

This document is organized as follows:

- Chapter 2 outlines the learning method used throughout this work. In particular, we discuss an online large-margin learning algorithm and compare it favorably to other discriminative learning frameworks.

- Chapter 3 formulates dependency parsing as the maximum spanning tree problem. We show how an edge based score factorization leads to tractable parsing algorithms for both the projective and non-projective case. Extending the factorization over pairs of edges leads to tractable algorithms for projective structures. However, non-projective parsing becomes NP-hard. We discuss a rich feature set under these factorizations that include both edge and edge-context information. Furthermore, we show how to extend the parsing models to produce dependency edge labels. This can be done either through a joint model or through a second-stage labeler.

- Chapter 4 presents extensive results for projective and non-projective parsing for benchmark English, Czech and Chinese data sets. We compare the models favorably to previous methods. In addition, we present results comparing joint and two-stage labeling.

- Chapter 5 presents parsing results on 14 diverse languages using a single parsing model. This section illustrates the language independence of the parser.

- In Chapter 6 we give a thorough analysis of the parsing errors and feature space of the dependency parser for English. We show that parser errors are similar to other parsing systems, e.g., preposition and conjunction attachment. An important aspect of this section is that we tease apart the contribution of each feature type used in the representation. We argue that edge-context features are important to simulate higher order features over larger structures of the parse space and are thus key to achieving high parsing accuracy. We also briefly discuss sources of error for other languages.

- In Chapter 7 we display the flexible nature of discriminative parsing models by easily incorporating features over auxiliary classifiers to improve the accuracy of both in domain and out of domain parsers. Then, in Chapter 8, we argue that the online large-margin learning algorithms of Chapter 2 are robust to approximate parsing algorithms by showing this empirically for higher-order non-projective parsing as well as non-tree dependency graph parsing.

- To show the applicability of the dependency parser in a real world problem, we incorporate it into a sentence compression system in Chapter 9. The resulting sentence compressor yields highly accurate compressions.

- Finally, we summarize the major contributions of this work and discuss related, ongoing and future work in Chapter 10 and Chapter 11.

Chapter 2

Online Large-Margin Learning

Parts of this chapter are drawn from material in [35].

In this chapter we present the learning algorithms that we will use for the rest of this work. One crucial property of these learning algorithms is that they are inference based, that is, to create trained models they only require the ability to find the highest scoring output given an input. This will be exploited throughout this work.

2.1 Structured Classification

Structured classification is a subfield of machine learning that develops theory and algorithms for learning how to label inputs with non-atomic outputs such as sequences and trees. After the introduction of conditional random fields (CRFs) [80], several researchers developed margin-based learning alternatives, in particular maximum margin Markov networks (M^{3}Ns) [137] and the related methods of Tsochantaridis et al. [143]. These algorithms have proven successful in several real world applications including sequential classification [86, 93, 120, 137], image labeling [62], natural language parsing [138, 143] and Web page classification [137]. All of these methods are in theory batch learning algorithms, in which the training objective is optimized with respect to all training instances simultaneously. In practice, however, the large-margin methods are often adapted to optimize with respect to a small number of instances at a time in order to handle large training sets.

This work focuses on purely online learning techniques. Unlike batch algorithms, online algorithms consider only one training instance at a time when optimizing parameters. This restriction to single-instance optimization might be seen as a weakness, since the algorithm uses less information about the objective function and constraints than batch algorithms. However, we will argue that this potential weakness is balanced by the simplicity of online learning, which allows for more streamlined training methods. We focus here on variants of the perceptron algorithm [116], which inherit its conceptual and mathematical simplicity and scale up to large problems much better than batch algorithms.

Online learning with perceptron-style algorithms has recently gained popularity due to the work of Collins [26], who uses an approximation to the voted perceptron algorithm [52], called here the averaged perceptron algorithm, for sequential classification problems. This method has since been successfully adapted to parsing [30], language modeling [115] and very recently word alignment [97]. Perceptron-based approaches have gained a wide acceptance since they reduce learning to inference, which is needed anyway, and they routinely provide state-of-the-art performance.

One problem with the perceptron algorithm is that it does not optimize any notion of classification margin, which is widely accepted to reduce generalization error [10]. As a result, ad-hoc approximations such as parameter averaging are required. Here, we propose a large-margin online algorithm that generalizes the multi-class classification algorithm MIRA (Margin Infused Relaxed Algorithm [34, 37, 33]) to structured outputs, which in essence is a large-margin perceptron variant. The generalization is achieved by using k-best structural decoding to approximate the large-margin updates of MIRA. We will argue that MIRA for structured outputs has many desirable properties – including simplicity, accuracy and scalability – all of which make it a suitable learning method for complex structured outputs like dependency trees.

2.2 Online Learning

First, we define a linear score function for input/output pairs,

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

where $\mathbf{f}(\boldsymbol{x}, \boldsymbol{y})$ is a high dimensional feature representation of input $\boldsymbol{x}$ and output $\boldsymbol{y}$ and $\mathbf{w}$ is a corresponding weight vector. The goal will be to learn $\mathbf{w}$ so that correct outputs are given a high score and incorrect outputs a low score. 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 an input $\boldsymbol{x}_t$ and its correct output $\boldsymbol{y}_t$. Though these algorithms work for a variety of outputs, we focus on the case when the output space is the set of dependency parses for a given input sentence $\boldsymbol{x}$.

In this work we focus on online-learning algorithms that are instances of the algorithm schema in Figure 2.1. A single training instance is examined at each iteration, and the weight vector is updated by an algorithm-specific rule. The auxiliary vector $\mathbf{v}$ accumulates the 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 [26].

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

```
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 2.1: Generic online learning algorithm.

2.2.1 Margin Infused Relaxed Algorithm (MIRA)

Crammer and Singer [36] present a natural approach to large-margin multi-class classification, which was later extended by Taskar et al. [137] to structured classification:

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

where $L(\boldsymbol{y}, \boldsymbol{y}')$ is a real-valued loss for the parse $\boldsymbol{y}'$ relative to the correct parse $\boldsymbol{y}$. Informally, this minimizes the norm of the weight vector subject to *margin constraints* that keep the score of the correct parse above the score of each incorrect one by an amount given by the loss of the incorrect parse.

The Margin Infused Relaxed Algorithm (MIRA) [34, 37, 33] 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 parsing the instance under consideration with a margin given by the loss of the incorrect parses. This can be formalized by substituting the following update into line 4 of the generic online algorithm from Figure 2.1,

$$\begin{aligned}
&\mathbf{w}^{(i+1)} = \arg\min_{\mathbf{w}^*} \left\|\mathbf{w}^* - \mathbf{w}^{(i)}\right\| \\
&\text{such that } s(\boldsymbol{x}_t, \boldsymbol{y}_t) - s(\boldsymbol{x}_t, \boldsymbol{y}') \geq L(\boldsymbol{y}_t, \boldsymbol{y}'), \text{ with respect to } \mathbf{w}^* \\
&\forall \boldsymbol{y}' \in \text{parses}(\boldsymbol{x}_t)
\end{aligned} \tag{2.1}$$

This update attempts to minimize the change made to the weight vector subject to the set of margin constraints for the instance under consideration. This quadratic programming problem (QP) can be solved using Hildreth's algorithm [15]. Crammer and Singer [37] and Crammer et al. [34, 33] provide an analysis of both the online generalization error and convergence properties of MIRA.

For the dependency parsing problem, we defined the loss of a graph to be the number of words with incorrect incoming edges relative to the correct parse. This is closely related to the Hamming loss that is often used for sequences [137]. For instance, consider the correct graph in Figure 1.2 versus the incorrect one in Figure 2.2. The loss of the incorrect graph relative to the correct one is 2 since *with* and *bat* are both incorrectly labeled as modifiers of *ball*. Note that this definition assumes dependency graphs are always trees. This is just one possible definition of the loss. Other possibilities are the 0-1 loss [136] or another more linguistically motivated loss that penalizes some errors (say conjunction and preposition dependencies) over others. We use Hamming loss primarily since standard evaluation of dependency parsers (see Chapter 4) is based on the percentage of words that modify the correct head in the graph. Thus, the Hamming loss directly relates our training optimization to our final evaluation metric. It has been argued that learning model weights relative to some loss function is advantageous. For many applications, there are certain parts of the output structure 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 the best of our knowledge, only Finley and Joachims [48] have made novel use of the loss function for

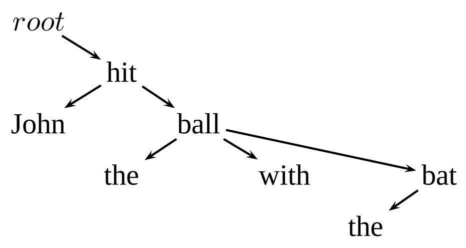


Figure 2.2: An example incorrect dependency parse relative to that in Figure 1.2. The loss of this parse is 2 since *with* and *bat* are incorrectly identified as modifiers of *ball*.

structured classification. In that work it was shown that setting the loss function to the final evaluation metric helped significantly for co-reference resolution.

To use these algorithms for structured classification, we follow the common method of equating structure prediction to multi-class classification, where each structure is a possible class for a sentence. As a result we inherit all the theoretical properties of multi-class classification algorithms. 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 sequential classification as well as dependency parsing.

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 parses $\boldsymbol{y}$ with the highest scores $s(\boldsymbol{x}, \boldsymbol{y})$. The resulting online update (to be inserted in Figure 2.1, line 4) would then be:

$$\begin{aligned}
&\mathbf{w}^{(i+1)} = \arg\min_{\mathbf{w}*} \left\|\mathbf{w}* - \mathbf{w}^{(i)}\right\| \\
&\text{such that } s(\boldsymbol{x}_t, \boldsymbol{y}_t) - s(\boldsymbol{x}_t, \boldsymbol{y}') \geq L(\boldsymbol{y}_t, \boldsymbol{y}'), \text{ with respect to } \mathbf{w}* \\
&\forall \boldsymbol{y}' \in \text{best}_k(\boldsymbol{x}_t; \mathbf{w}^{(i)})
\end{aligned}$$

In Section 2.3 we show that this relaxation works well and even small values of k yield near optimal performance. This is also true for dependency parsing, as is shown in Section 4.2.1.

We call this algorithm *k-best MIRA*. Throughout the rest of this document all experimental results for MIRA will be with 1-best MIRA unless stated otherwise.

This formulation of large-margin learning for structured outputs is highly related to that of Tsochantaridis et al. [143]. In that work a learning algorithm repeatedly runs inference over training examples to create a growing set of constraints. Parameter optimization is then run over all collected constraints. Since this optimization incorporates constraints from all the instances in training, it is primarily a batch learning algorithm. However, since the method used to collect the constraints is essentially online, one can consider it a hybrid.

Factored MIRA

Another option would be to factor the constraints relative to the structure of the output to produce an equivalent polynomial sized set of constraints. Taskar et al. [137, 138] showed that this can be done for both sequences and phrase-structure trees, providing that the loss function can also factor relative to the structure of the output. The advantage of this approach is that it provides an exact solution to the QP given by (2.1). Even though the resulting set of constraints is still polynomial, it is typically linear or squared in the length of the input and can lead to large QP problems. For these reason we restrict ourselves to k-best MIRA solutions.

Non-separability and Kernelization

Many large-margin learning algorithms have benefited from introducing slack variables that trade-off good margin properties versus a separating hyperplane [137] as a way to guarantee convergence when the data is not separable as well as reduce overfitting when outliers exist. In the original formulation of MIRA for multi-class classification [37] slack variables are included. We implemented a version of MIRA with slack variables but found it had negligible impact on performance, so we leave it out of discussion for simplicity.

Another advantage of linear classifier models such as the perceptron and SVMs is that they can be kernelized, that is, they can be reformulated so that all learning and inference is calculated by a similarity metric (a kernel) between input points [98]. This can often make large (even infinite) feature space calculations tractable. MIRA can easily be kernelized. However, we focus on the feature space representation of inputs that is more common in the NLP community. Defining interesting kernels compatible with the algorithms we provide here is an area of research beyond the scope of this work.

2.3 Empirical Comparison for Sequence Classification

Unfortunately there are no readily available implementations of CRFs or M^{3}Ns for parsing and to implement them is a non-trivial engineering task. In order to compare MIRA to other learning frameworks we will look at sequential classification tasks that represent non-trivial language processing problems and for which implementations are available.

Our first set of experiments are on the handwriting recognition task used by Taskar et al. [137] to evaluate M^{3}Ns, which is a subset of an earlier evaluation collection [72]. We use exactly the same data set, training and test splits, instance representation, and first and second degree kernel data representations. For this experiment we compared averaged perceptron, k-best MIRA ($k = 20$), CRFs and M^{3}Ns. Looking at Table 9.1(a), we can see that both MIRA and M^{3}Ns outperform the averaged perceptron, most likely due to the fact that they aggressively maximize margin. Furthermore, we can see the performance of MIRA is comparable to that of M^{3}Ns for the degree 2 kernel[1]. In the impoverished degree 1 kernel, both CRFs and M^{3}Ns appear to be the best option. Interestingly, CRFs perform as well as M^{3}Ns for the degree 1 kernel, but do much worse than all methods for the degree 2 kernel. The reason for this is the use of feature normalization [137], which improves

[1]We simulate the degree 1 and 2 kernel in the primal feature space.

the performance of all methods dramatically except for CRFs, whose performance seems independent of normalization. Feature normalization artificially increases the value of the edge bias feature (i.e., the state transition feature) to prevent it from being overwhelmed by the overall magnitude of the vertex features in the model, in particular for the second-degree kernel.

Our next experiments involved two larger sequential classification problems: noun-phrase chunking and named-entity recognition. For these problems, we used a feature-based primal formulation, which is commonly used in natural language processing and easier to implement. For noun-phrase chunking we implemented a first-order Markov model using the features of Sha and Pereira [120] on the CoNLL 2000 data set [141]. For named-entity recognition we used standard word and sub-word features on the CoNLL-2003 data set [142], including word, part-of-speech, suffix and prefix identity as well as orthographic features (e.g. word is capitalized), with all features over a window of two words previous to two words next.

These data sets contain thousands of sentences (roughly 9000 for chunking and 14,000 for named-entity recognition). For both tasks we compared three methods: averaged perceptron, k-best MIRA ($k = 5$) and CRFs. We did not have access to an implementation of M^{3}Ns for chunking or entity recognition. Training takes under 5 hours for all the other algorithms. The results are shown in Table 9.1(b-c). We can see that MIRA slightly outperforms the averaged perceptron and begins to bridge the performance gap with batch learners such as CRFs. According to McNemar significance tests, the difference between MIRA and averaged perceptron is only significant for named-entity recognition ($p < 0.0001$) and the difference between MIRA and CRFs is not significant.

For the larger data sets (chunking and entity extraction) we observed that maximum performance is achieved with $5 \leq k \leq 10$, with sometimes diminishing returns when $k > 20$, providing evidence that the approximation is reasonable. All results for MIRA

(a) **Handwriting** (degree 1/degree 2)

	Accuracy
Avg. Perc.	0.781 / 0.859
MIRA	0.785 / 0.870
CRF	0.802 / 0.818
M^3Ns	0.803 / 0.869

(b) **NP chunking**

	F-Meas
Avg. Perc.	0.941
MIRA	0.942
CRFs	0.943

(c) **NE recognition**

	F-Meas
Avg. Perc.	0.823
MIRA	0.830
CRFs	0.831

Table 2.1: Structured classification experimental results.

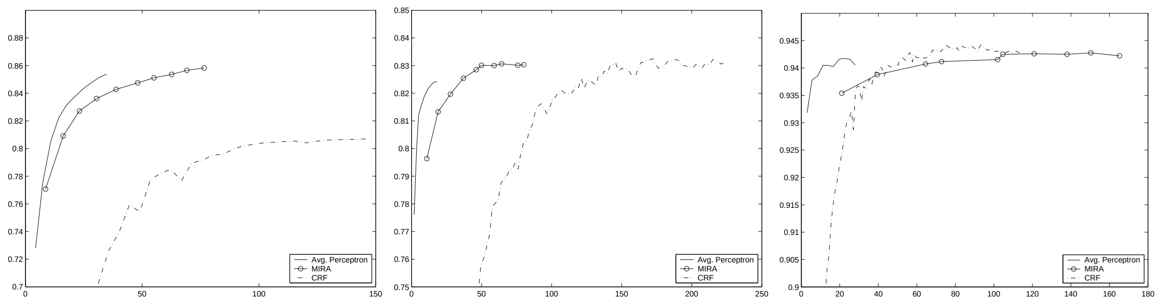


Figure 2.3: Handwriting recognition (degree 2 kernel), named-entity recognition and noun-phrase chunking. The plots show performance on testing vs. training time in CPU minutes.

also include parameter averaging. Though important for both models, averaging had a much smaller effect on accuracy for MIRA.

2.3.1 Performance versus Training Time

Due to the complexity of the output space, learning with structured outputs is inherently more computationally demanding than simple classifier learning. In fact, many studies on batch learning for structured outputs use small or reduced data sets to fit within available computing budgets [137, 138, 143]. In contrast, the online learning methods studied here require only k-best inference on a single instance in addition to solving a small quadratic program. Figure 6.1 plots the test accuracy for each method against CPU training time for all three tasks studied in this paper. We only compare averaged perceptron, MIRA and CRFs, since we did not have an implementation of M^3Ns. Results for these plots were gathered on a dual processor 1.3GHz 32-bit Pentium with 2GB of memory.

These plots show that learning with k-best MIRA and perceptron is typically much less costly than learning with CRFs. A training epoch has roughly the same computational cost, due to inference, for the three methods (Viterbi for perceptron and MIRA, forward-backward for CRFs). However, MIRA and perceptron quickly converge after a few iterations, whereas CRFs require many iterations before attaining high test accuracy.

2.4 Why MIRA for Dependency Parsing?

When it comes to discriminative training, there are many options. In this section we summarize why we believe MIRA is good discriminative learning framework for dependency parsing.

- **Accuracy:** MIRA performs just as well or better than the leading batch learning algorithms, CRFs and M^{3}Ns while routinely outperforms the averaged perceptron algorithm. Of course, this only applies to sequence classification problems for which we can compare these methods.

- **Efficiency:** With the exception of the perceptron algorithm, MIRA is the fastest global discriminative learner, mainly because it only takes a few iterations to achieve optimal performance.

- **Simplicity:** MIRA relies only on inference and Hildreth's algorithm to solve the quadratic programming problem. Together, this typically represents only a few hundred lines of code and both are relatively straightforward to implement – unlike forward-backward or the inside-outside algorithms (required for CRFs), computing marginal distributions (required for M^{3}Ns) or complicated convex optimization code (required for CRFs).

In addition to these three main points we should also note that MIRA, like the perceptron and M^{3}Ns, do not require the calculation of a normalization function in order to return a proper probability distribution. It has been shown that for some problems inference is tractable, but computing the normalization is not [136]. Hence, methods like MIRA have an additional advantage over CRFs since they do not need to compute a normalization factor. Furthermore, MIRA is very flexible with respect to the loss function. Any loss function on the output is compatible with MIRA since it does not require the loss to factor according to the output, unlike M^{3}Ns.

In summary, we have argued that the online large-margin learning algorithm of MIRA is a learning framework suitable for structured outputs like dependency graphs.

Chapter 3

Dependency Parsing

Parts of this chapter are drawn from material in [91, 95, 94, 92].

This chapter describes both the parsing models (Section 3.1, Section 3.2 and Section 3.3) and inference algorithms (Section 3.1 and Section 3.3) that constitute the core of our dependency parser.

3.1 Dependency Structures as Maximum Spanning Trees

In this section we translate the problem of dependency parsing into that of finding maximum spanning trees for directed graphs. This formulation provides a unified theoretical framework for discussing the algorithmic properties of inference in projective and non-projective parsing.

3.1.1 First-Order Spanning Tree Parsing

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.

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 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,

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

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 $\boldsymbol{x}$. This is true for learning as well since we focus on an inference based online learning framework (Chapter 2). We should note that the feature representation $\mathbf{f}(i,j)$ can also include arbitrary features on the sentence $\boldsymbol{x}$ since it always fixed as input. To indicate this fact, a more appropriate representation of the feature function would be $\mathbf{f}(\boldsymbol{x},i,j)$. However, for notational simplicity we will just define $\mathbf{f}(i,j) = \mathbf{f}(\boldsymbol{x},i,j)$.

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, $s(\cdot,\cdot)$ is not symmetric. 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 r-arborescence or maximum branching problem [135].

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

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

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. By definition, a spanning tree of G is a sub-graph G' with nodes $V' = V$ and edges $E' \subseteq E$, such that G' is weakly connected and all the nodes in V' have an in-degree of exactly 1 except the unique root node with in-degree 0. This definition is equivalent to being a dependency graph satisfying the tree constraint (Section 1.3.1). 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. Thus by factoring the score of the tree into the sum of edge scores we have made dependency parsing equivalent with finding maximum spanning trees.

Throughout this work we will refer to this particular spanning tree formulation as the *first-order* spanning tree problem (or *first-order* dependency parsing problem). This is because the score factors as a sum of individual edge scores. Of course, we can factor the score of the tree any way we wish, though not all factorizations lead to efficient parsing algorithms. In Section 3.1.2 we will indeed modify how we factor the score of the tree to incorporate features over pairs of edges.

In what follows, we make the assumption that calculating $s(i,j)$ is $O(1)$. In fact, this is slightly misleading since **w** and **f** typically have a dimension in the millions. As usual, sparse vector representations are used to reduce the calculation to linear in the number of features that are active for a given edge. We can view this calculation as some form of grammar constant, which is a common notion for most parsing formalisms. We will argue

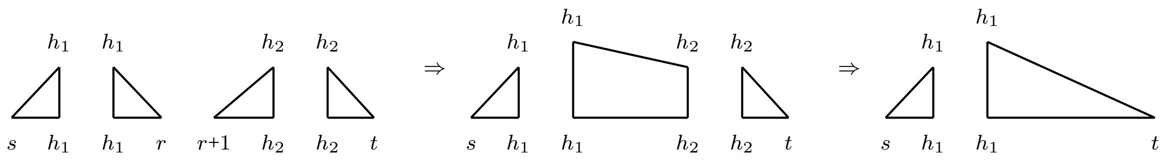


Figure 3.1: Cubic parsing algorithm of Eisner [45].

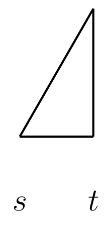
in Section 3.2 that this constant is typically very small (roughly 100), especially when compared to grammar constants in phrase-based models, which can be on the order of tens of thousands when extracted from a large treebank.

Projective Parsing Algorithms

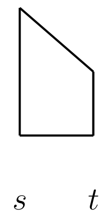
Using a slightly modified version of the CKY [152] chart parsing algorithm, it is possible to generate and represent all projective dependency trees in a forest that is $O(n^5)$ in size and takes $O(n^5)$ time to create, which is equivalent to context-free phrase-structure parsing. However, Eisner [45] 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 3.1 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, all items are complete, which is represented by each right angle triangle. The algorithm then creates an incomplete item from the words h_1 to h_2 with h_1 as the head of h_2. This item is eventually completed at a later stage. As with normal CKY parsing, larger items are created from pairs of smaller items in a bottom-up fashion.

It is relatively easy to augment this algorithm so that each chart item also stores the score of the best possible subtree 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.

We will now describe the Eisner parsing algorithm in more detail. 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. The variable $d \in \{\leftarrow, \rightarrow\}$ 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. The variable $c \in \{0, 1\}$ 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,



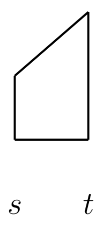
The Eisner algorithm fills in the dynamic programming table bottom-up just like the CKY parsing algorithm [152] by finding optimal subtrees for substrings of increasing increasing length. Pseudo code for filling up the dynamic programming table is in Figure 3.2.

Consider the line in Figure 3.2 indicated by (*). This says that to find the best score for

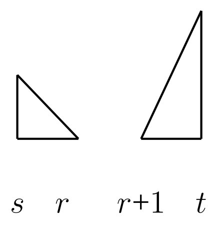
```
Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$
for $k$ : $1..n$
  for $s$ : $1..n$
    $t = s + k$
    if $t > n$ then break

     % First: create incomplete items
    $C[s][t][\leftarrow][0] = \max_{s \leq r < t} \ (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t,s))$   (*)
    $C[s][t][\rightarrow][0] = \max_{s \leq r < t} \ (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s,t))$

     % Second: create complete items
    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} \ (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$
    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} \ (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

  end for
end for
```

Figure 3.2: Pseudo-code for bottom-up Eisner cubic parsing algorithm.

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 $C[1][n][\rightarrow][1]$. This can be shown easily by structural induction using the inductive hypothesis that the chart stores the best score over all strings of smaller length. A quick look at the pseudo-code shows that the run-time of the Eisner algorithm is $O(n^3)$.

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 a multiplicative factor of $O(k \log k)$ [66].

Non-projective Parsing Algorithms

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 [31], as well as k-best extensions to them [47]. Efficient algorithms for the directed case are less well known, but they exist. We will use here the Chu-Liu-Edmonds algorithm [21, 43], sketched in Figure 3.3 following Georgiadis [54]. 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

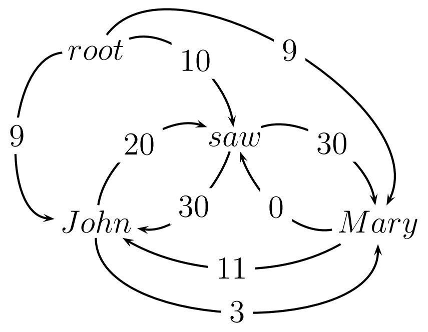
```
**Chu-Liu-Edmonds**$(G, s)$
        Graph $G = (V, E)$
        Edge weight function $s : E \rightarrow \mathbb{R}$
1.      Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2.      Let $G_M = (V, M)$
3.      If $G_M$ has no cycles, then it is an MST: return $G_M$
4.      Otherwise, find a cycle $C$ in $G_M$
5.      Let $< G_C, c, ma >=$ contract$(G, C, s)$
6.      Let $\boldsymbol{y} =$ Chu-Liu-Edmonds$(G_C, s)$
7.      Find vertex $x \in C$
          such that $(x', c) \in \boldsymbol{y}$ and $ma(x', c) = x$
8.      Find edge $(x'', x) \in C$
9.      Find all edges $(c, x''') \in \boldsymbol{y}$
10.     $\boldsymbol{y} = \boldsymbol{y} \cup \{(ma(c, x'''), x''')\}_{\forall (c,x''') \in \boldsymbol{y}}$
              $\cup\, C \cup \{(x', x)\} - \{(x'', x)\}$
11.     Remove all vertices and edges in $\boldsymbol{y}$ containing $c$
12.     return $\boldsymbol{y}$
```

```
**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
             $ma(c, x) = \arg\max_{x' \in C} s(x', x)$
             $x' = ma(c, x)$
             $s(c, x) = s(x', x)$
4.    For $x \in V - C : \exists_{x' \in C}(x, x') \in E$
          Add edge $(x, c)$ to $G_C$ with
             $ma(x, c) = \arg\max_{x' \in C} [s(x, x') - s(a(x'), x')]$
             $x' = ma(x, c)$
             $s(x, c) = [s(x, x') - s(a(x'), x') + s(C)]$
                where $a(v)$ is the predecessor of $v$ in $C$
                and $s(C) = \sum_{v \in C} s(a(v), v)$
5.    return $< G_C, c, ma >$
```

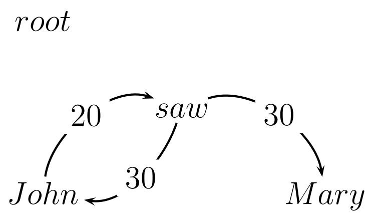
Figure 3.3: Chu-Liu-Edmonds algorithm for finding maximum spanning trees in directed graphs.

shown that a maximum spanning tree on the resulting contracted graph is equivalent to a maximum spanning tree in the original graph [54]. 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 [135] gives an efficient implementation of the algorithm with $O(n^2)$ time complexity for dense graphs, which is what we need here. These algorithms can be extended to the k-best case [14] with a run-time of $O(kn^2)$.

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 this on the simple example $\boldsymbol{x} =$ *John saw Mary*, with directed graph representation $G_{\boldsymbol{x}}$,

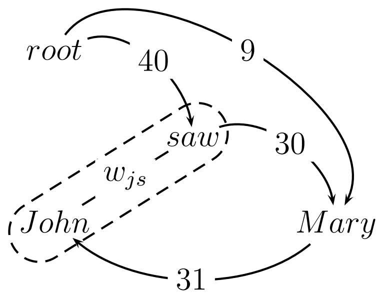


The first step of the algorithm is to find, for each word, the highest scoring incoming edge



If the result of greedily choosing the highest scoring incoming edge to every node results in a tree, it would have to be a maximum spanning tree. To see this, consider a tree T constructed by greedily choosing the highest scoring incoming edge for every word. Now consider a tree T' such that $T \neq T'$ and T' is the maximum spanning tree. Find edges $(i, j) \in T$ and $(i', j) \in T'$ such that $i \neq i'$. We know by the definition of T that the score of (i, j) is at least as large than the score of (i', j). So we can simple make the change $T' = T' \cup \{(i, j)\} - \{(i', j)\}$ and T' will be a graph of a least equal weight. If we repeat this process, we will eventually converge to the tree T and we are always guaranteed that the resulting graph will have a score at least as large as that of T'. Thus, either T' could not have been the maximum spanning tree, or both T and T' are trees of equal weight. Either way, T is a maximum spanning tree.

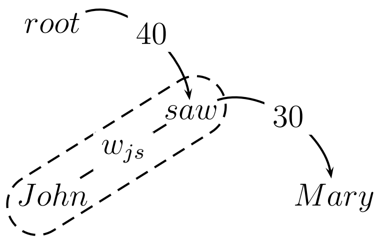
In the current example there is a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 3.3.



The new vertex w_{js} represents the contraction of vertices *John* and *saw*. The edge from w_{js}

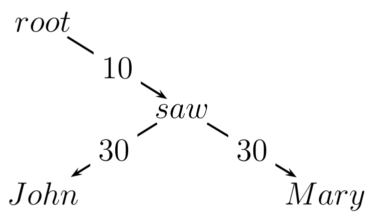
to *Mary* is 30 since that is the highest scoring edge from any vertex in w_{js}. The edge from *root* into w_{js} is set to 40 since this represents the score of the best spanning tree originating from *root* and including the vertices in the cycle represented by w_{js}. The same leads to the edge from *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 [54]. The proof of this fact follows from the lemma that, after the greedy step, all the edges of any cycle must exist in some MST, except a single edge. That single edge is one that must be removed to break this cycle and satisfy the tree constraint. Knowing this lemma, we can observe that in the contracted graph, the weight of edges going into the contracted node represent, exactly, the best score of that edge entering the cycle and breaking it. For example, the edge from *root* into w_{js} is 40 representing that edge entering the node *saw* and breaking the cycle by removing the single edge from *John* to *saw*.

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 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. Again, we have assumed a data driven approach to parsing and appeal to the properties of the training data to eliminate such cases. We address this concern in Chapter 4.

3.1.2 Second-Order Spanning Tree Parsing

Restricting scores to a single edge in a dependency tree is a very impoverished view of dependency parsing. Yamada and Matsumoto [151] showed that keeping a small amount of parsing history was crucial to improving performance for their locally trained shift-reduce SVM parser. It is reasonable to assume that other parsing models will benefit from features over previous decisions.

Here we will focus on methods for parsing *second-order* spanning trees. These models factor the score of the tree into the sum of adjacent edge pairs. To quantify this, consider the example from Figure 1.2, with words indexed: root(0) John(1) hit(2) the(3) ball(4) with(5) the(6) bat(7). Using a first-order spanning tree formulation, the score of this tree would be,

$$\begin{gathered} s(0,2) + s(2,1) + s(2,4) + s(2,5) \\ + s(4,3) + s(5,7) + s(7,6) \end{gathered}$$

However, in our second-order spanning tree model, the score of this tree would be,

$$s(0,-,2) + s(2,-,1) + s(2,-,4) + s(2,4,5)$$
$$+ s(4,-,3) + s(5,-,7) + s(7,-,6)$$

Here we have changed the score function to $s(i,k,j)$, which is the score of creating a pair of adjacent edges, from word x_i to words x_k and x_j. For instance, $s(2,4,5)$ is the score of creating a the edges from *hit* to *with* and from *hit* to *ball*. The score functions are relative to the left or right of the head and we never score adjacent edges that are on different sides of the head (e.g. $s(2,1,4)$ for the adjacent edges from *hit* to *John* and *ball*). This left/right independence assumption is common and will allow us to define polynomial second-order projective parsing algorithms. We let $s(i,-,j)$ be the score when x_j is the first left/right dependent of word x_i. For example, $s(2,-,4)$ indicates the score of creating a dependency from *hit* to *ball*, where *ball* is the first modifier to the right of *hit*. More formally, if the word x_{i_0} has the modifiers as shown,

$$\begin{tikzcd}
 & & & x_{i_0} \arrow[dlll, bend right] \arrow[dl] \arrow[dr] \arrow[drrr, bend left] & & & \\
x_{i_1} & \dots & x_{i_j} & & x_{i_{j+1}} & \dots & x_{i_m}
\end{tikzcd}$$

the score factors as follows:

$$\textstyle\sum_{k=1}^{j-1} s(i_0, i_{k+1}, i_k) + s(i_0, -, i_j)$$
$$+ s(i_0, -, i_{j+1}) + \textstyle\sum_{k=j+1}^{m-1} s(i_0, i_k, i_{k+1})$$

A second-order MST is mathematically a richer factorization, since the score function can just ignore the middle modifier, or *sibling*, argument and it would be reduced to the standard first-order model. In fact our second order score do incorporate first-order information, $s(i,k,j) = s(i,k,j) + s(i,j)$. Here the first term includes features over the pairs of adjacent edges and the second over features of a single edge. It is also important to note that $s(i,k,j) \neq s(i,j,k)$. In fact, the order of the two adjacent modifiers is determined by there relative location in the sentence to the head. The closer modifier is always the first argument. Furthermore, for features over pairs of edges the relative order of the modifiers is always incorporated.

The score of a tree for second-order parsing is now,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,k,j) \in \boldsymbol{y}} s(i, k, j)$$

Which is the sum of adjacent edge scores in $\boldsymbol{y}$.

Essentially the second-order model allows us to condition on the most recent parsing decision, i.e. the last dependent picked up by a particular word. This is analogous to the Markov conditioning of the Charniak parser [16] for phrase-structure parsing.

When moving to this second-order factorization we have introduced the notion of edge adjacency in a tree. This notion is only meaningful when there is a fixed order on the vertexes in the graph, as is the case with dependency parsing. It is with respect to this restricted formulation that we consider maximum spanning tree parsing in this section.

A Projective Parsing Algorithm

In this section we describe a $O(n^3)$ second-order parsing algorithm that works by breaking up dependency creation in the first-order algorithm into two steps - sibling creation followed by head attachment. This cubic extension to the second-order case was in the original work of Eisner [45]. Graphically the intuition behind the algorithm is given in Figure 3.4. The key insight is to delay completion of items until all the dependents of the head have been gathered. This allows for the collection of pairs of sibling dependents in a single stage while maintaining a cubic time parsing algorithm. We will define a new item type called a *sibling* type (in addition to the usual *complete* and *incomplete* types).

The algorithm works by defining an almost identical bottom-up dynamic programming table as the original Eisner algorithm. The only difference is the addition the new *sibling*

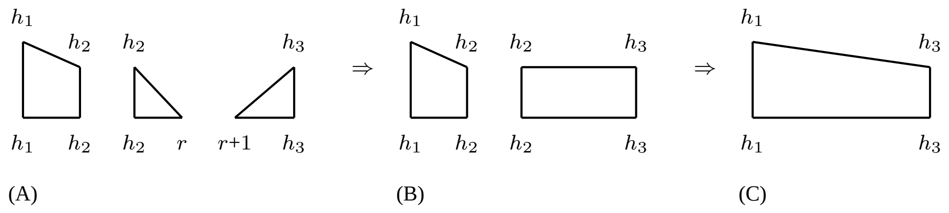


Figure 3.4: An extension of the Eisner algorithm to second-order dependency parsing. This figure shows how h_1 creates a dependency to h_3 with the second-order knowledge that the last dependent of h_1 was h_2. This is done through the creation of a *sibling* item in part (B).

type. Pseudo-code for the algorithm is given in Figure 3.5. As before, we 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. In the second-order case we let $c \in \{0, 1, 2\}$ to indicate if a subtree is complete ($c = 1$, no more dependents), incomplete ($c = 0$, needs to be completed), or represents sibling subtrees ($c = 2$). Sibling types have no inherent direction, so we will always assume that when $c = 2$ then $d =$ null (-). As in the first-order case, the proof of correctness is done through structural induction. Furthermore, back-pointers can be included to reconstruct the highest scoring parse and the k-best parses can be found in $O(k \log(k) n^3)$.

An Approximate Non-projective Parsing Algorithm

Unfortunately second-order non-projective MST parsing is NP-hard. We prove this fact with a reduction from 3-dimensional matching.

3DM: Disjoint sets, X, Y, Z each with m distinct elements, and a set $T \subseteq X \times Y \times Z$. Question: is there a subset $S \subseteq T$ such that $|S| = m$ and each $v \in X \cup Y \cup Z$ occurs in exactly one element of S.

Reduction: Given an instance of 3DM we define a graph in which the vertices are the elements of $X \cup Y \cup Z$ as well as an artificial *root* node. We insert edges from *root* to all $x \in X$ as well as edges from all $x \in X$ to all $y \in Y$ and $z \in Z$. We order the words s.t.

```
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

     % Create Sibling Items
    $C[s][t][\text{-}][2] = \max_{s \leq r < t} \ \{C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1]\}$

     % First Case: head picks up first modifier
    $C[s][t][\leftarrow][0] = C[s][t-1][\rightarrow][1] + C[t][t][\leftarrow][1] + s(t, \text{-}, s)$
    $C[s][t][\rightarrow][0] = C[s][s][\rightarrow][1] + C[s+1][t][\leftarrow][1] + s(s, \text{-}, t)$

     % Second Case: head picks up a pair of modifiers (through a sibling item)
    $C[s][t][\leftarrow][0] = \max \ \{C[s][t][\leftarrow][0], \ \max_{s \leq r < t} \ \{C[s][r][\text{-}][2] + C[r][t][\leftarrow][0] + s(t, r, s)\}\}$
    $C[s][t][\rightarrow][0] = \max \ \{C[s][t][\rightarrow][0], \ \max_{s < r \leq t} \ \{C[s][r][\rightarrow][0] + C[r][t][\text{-}][2] + s(s, r, t)\}\}$

     % Create complete items
    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} \ \{C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0]\}$
    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} \ \{C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1]\}$

  end for
end for
```

Figure 3.5: Pseudo-code for bottom-up second-order Eisner parsing algorithm.

the root is on the left followed by all elements of X, then Y, and finally Z. The order of elements within each set is unimportant. We then define the second-order score function as follows,

$s(root, x, x') = 0,\ \forall x, x' \in X$

$s(x, -, y) = 0,\ \forall x \in X, y \in Y$

$s(x, y, z) = 1,\ \forall (x, y, z) \in T$

All other scores are defined to be $-\infty$, including for edges pairs that were not defined in the original graph.

Theorem: *There is a 3D matching iff the second-order MST has a score of m.*

Proof: First we observe that no tree can have a score greater than m since that would require more than m pairs of edges of the form (x, y, z). This can only happen when some x has multiple $y \in Y$ modifiers or multiple $z \in Z$ modifiers. But if this were true then we would introduce a $-\infty$ scored edge pair (e.g. $s(x, y, y')$). Now, if the highest scoring second-order MST has a score of m, that means that every x must have found a unique pair of modifiers y and z_k which represents the 3D matching, since there would be m such triples. Furthermore, y and z could not match with any other x' since they can only have one incoming edge in the tree. On the other hand, if there is a 3DM, then there must be a tree of weight m consisting of second-order edges (x, y, z) for each element of the matching S. Since no tree can have a weight greater then m, this must be the highest scoring second-order MST. Thus if we can find the highest scoring second-order MST in polynomial time, then 3DM would also be solvable. Note that this proof works for both dependency parsing with the left/right modifier independent assumption and without. ■

Thus, the Chu-Liu-Edmonds algorithm most likely cannot be extended polynomially to handle second-order feature representations. This is an important result, since it shows that even for data driven parsing, non-projective exact search becomes intractable for any

```
2-order-non-proj-approx($\boldsymbol{x}, s$)
      Sentence $\boldsymbol{x} = x_0 \dots x_n$, $x_0 = root$
      Weight function $s : (i, k, j) \to \mathbb{R}$
 1.   Let $\boldsymbol{y}$ = 2-order-proj($\boldsymbol{x}, s$)
 2.   while true
 3.     $m = -\infty, c = -1, p = -1$
 4.     for $j : 1 \cdots n$
 5.       for $i : 0 \cdots n$
 6.         $\boldsymbol{y}' = \boldsymbol{y}[i \to j]$
 7.         if $\neg\text{tree}(\boldsymbol{y}')$ or $\exists k : (i, k, j) \in \boldsymbol{y}$ continue
 8.         $\delta = s(\boldsymbol{x}, \boldsymbol{y}') - s(\boldsymbol{x}, \boldsymbol{y})$
 9.         if $\delta > m$
10.           $m = \delta, c = j, p = i$
11.       end for
12.     end for
13.     if $m > 0$
14.       $\boldsymbol{y} = \boldsymbol{y}[p \to c]$
15.     else $return\ \boldsymbol{y}$
16.   end while
```

Figure 3.6: Approximate second-order non-projective parsing algorithm.

factorization other than first-order[1]. To combat this, we will create an approximate algorithm based on the $O(n^3)$ second-order projective parsing algorithm just provided. The approximation will work by first finding the highest scoring projective parse. It will then rearrange edges in the tree, one at a time, as long as such rearrangements increase the overall score and do not violate the tree constraint. We can clearly motivate this approximation by observing that even in non-projective languages like Czech and Dutch, most trees are primarily projective with just a few non-projective edges [104]. Thus, by starting with the highest scoring projective tree, we are typically only a small number of transformations away from the highest scoring non-projective tree. Pseudo-code for the algorithm is given in Figure 3.6.

The expression $\boldsymbol{y}[i \to j]$ denotes the dependency graph identical to $\boldsymbol{y}$ except that x_j's head is x_i instead of what it was in $\boldsymbol{y}$. The test $\text{tree}(\boldsymbol{y})$ is true iff the dependency graph $\boldsymbol{y}$

[1]Even though the above reduction was for pairwise adjacent edge factorization, it is easy to extend the reduction for arbitrary constraints over more than one edge.

satisfies the tree constraint.

In more detail, line 1 of the algorithm sets $\boldsymbol{y}$ to the highest scoring second-order projective tree. The loop of lines 2-16 exits only when no further score improvement is possible. Each iteration seeks the single highest-scoring change in dependency within $\boldsymbol{y}$ that does not break the tree constraint. To that effect, the nested loops starting in lines 4 and 5 enumerate all (i, j) pairs. Line 6 sets $\boldsymbol{y}'$ to the dependency graph obtained from $\boldsymbol{y}$ by changing x_j's head to x_i. Line 7 checks that the move from $\boldsymbol{y}$ to $\boldsymbol{y}'$ is valid and that x_j's head was not already x_i and that $\boldsymbol{y}'$ is a tree. Line 8 computes the score change from $\boldsymbol{y}$ to $\boldsymbol{y}'$. If this change is larger then the previous best change, we record how this new tree was created (lines 9-10). After considering all possible valid edge changes to the tree, the algorithm checks to see that the best new tree does have a higher score. If that is the case, we change the tree permanently and re-enter the loop. Otherwise we exit since there are no single edge changes that can improve the score.

This algorithm allows for the introduction of non-projective edges because we do not restrict any of the edge changes except to maintain the tree property. In fact, if any edge change is ever made, the resulting tree is guaranteed to be non-projective, otherwise there would have been a higher scoring projective tree that would have already been found by the exact projective parsing algorithm.

It is clear that this approximation will always terminate – there are only a finite number of dependency trees for any given sentence and each iteration of the loop requires an increase in score to continue. However, the loop could potentially take exponential time, so we will bound the number of edge transformations to a fixed value M. It is easy to argue that this will not hurt performance. Even in freer-word order languages such as Czech, almost all non-projective dependency trees are primarily projective, modulo a few non-projective edges. Thus, if our inference algorithm starts with the highest scoring projective parse, the best non-projective parse only differs by a small number of edge transformations.

Furthermore, it is easy to show that each iteration of the loop takes $O(n^2)$ time, resulting in a $O(n^3 + Mn^2)$ runtime algorithm. In practice, the approximation terminates after a small number of transformations and we do not bound the number of iterations in our experiments. In fact, the run-time of this algorithm is dominated by the call to **2-order-proj** (see Chapter 4).

We should note that this is one of many possible approximations we could have made. Another reasonable approach would be to first find the highest scoring *first-order* non-projective parse, and then re-arrange edges based on second order scores in a similar manner to the algorithm we described. We implemented this method and found that the results were slightly worse.

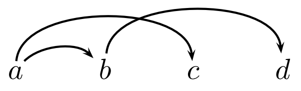
It is also easy to find examples in which this approximation will fail to find the highest scoring non-projective parse. Consider a sentence $\boldsymbol{x} = a\ b\ c\ d$, i.e. $root = a$. We set the following edge weights,

$$\begin{array}{l} s(a, b, c) = 10 \\ s(b, -, d) = 10 \\ s(d, c, b) = 11 \end{array}$$

Let every other second-order edge have a weight of 0. Now, note that any pair of the three second-order edges with score greater than 0 either cross or result in a cycle, hence the highest scoring projective tree can contain at most one of these. This also means that the highest scoring projective tree can have a score of at most 11. It turns out that there is exactly one projective tree with a score of 11,

$$\begin{tikzcd} a \arrow[rrr, bend left] & b & c & d \arrow[ll, bend right] \arrow[l, bend right] \end{tikzcd}$$

Now, if we enter the edge transformation step of the algorithm, we can look for changes of heads in b, c or d that lead to a higher scoring tree, since a is the root. Of course, there are only two non-zero second-order edges not in this tree, (a, b, c) and $(b, -, d)$. The former edge cannot be constructed in a single edge transformation since it results in a change of head for both b and c and the latter will result in a cycle. Thus, there is no single edge change that will lead to a strictly higher scoring tree, so the algorithm stops. However, consider the tree,



This tree has a score of 20, which is obviously a higher scoring tree than the one returned by the algorithm. This example relies on the property that very similar second-order edges will have a wide range in scores and that very different edges (e.g. reversal of the head and modifier) will have very similar scores. Both of these properties are unlikely in practice.

We should note the similarity of this approximate dependency parsing algorithm with that of Foth et al. [51]. In that work they describe an algorithm for constraint based dependency parsing [85, 60] in which a suboptimal solution is initially found and subsequent local constraint optimizations attempt to push the algorithm near the global optimum. As is the case with our algorithm it is possible for this method to get stuck in a local maxima. Their main motivation to designing this algorithm was to overcome difficulties in a standard constraint based dependency grammar when parsing spoken dialogue.

3.1.3 Summary: Dependency Parsing as MST

In this section we defined dependency parsing as the search for maximum spanning trees in directed graphs constructed from sentences. This formulation naturally led to first and second-order parsing algorithms for projective structures that were originally defined by Eisner [45]. More surprisingly, a polynomial first-order parsing algorithm was provided for non-projective trees that was based on the Chu-Liu-Edmonds MST algorithm. Unfortunately, it has been shown that second-order non-projective MST parsing is NP-hard and thus we were forced to define an approximate algorithm. However, in Chapter 4 we will see that this approximation still leads to state-of-the-art results.

The major contribution of this formulation is that it provides a uniform framework for defining and parsing both projective and non-projective languages using efficient cubic parsing techniques. Furthermore, it unifies previous work on parsing dependencies into a single framework, including the work of Eisner [45], Foth et al. [51], Hirakawa [65] and Ribarov [112].

3.2 Defining the Feature Space

In the last section, 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). The basic set of features we use are shown in Table 3.1a 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 head-modifier pairs in the tree is not enough for high accuracy since all attachment decisions are 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 3.1c. The first new feature class recognizes word types that occur between the head and modifier words in an attachment decision. These features take the form of POS trigrams: the POS of the head, that of the modifier, and that of a word in between, for all distinct POS tags for the words between the head and the modifier. These features were particularly helpful for nouns to select their heads correctly, since they help reduce the score for attaching a noun to another noun with a verb in between, which is a relatively infrequent a)

Basic Uni-gram Features
x_i-word, x_i-pos
x_i-word
x_i-pos
x_j-word, x_j-pos
x_j-word
x_j-pos

b)

Basic Bi-gram Features
x_i-word, x_i-pos, x_j-word, x_j-pos
x_i-pos, x_j-word, x_j-pos
x_i-word, x_j-word, x_j-pos
x_i-word, x_i-pos, x_j-pos
x_i-word, x_i-pos, x_j-word
x_i-word, x_j-word
x_i-pos, x_j-pos

c)

In Between POS Features
x_i-pos, b-pos, x_j-pos
Surrounding Word POS Features
x_i-pos, x_i-pos+1, x_j-pos-1, x_j-pos
x_i-pos-1, x_i-pos, x_j-pos-1, x_j-pos
x_i-pos, x_i-pos+1, x_j-pos, x_j-pos+1
x_i-pos-1, x_i-pos, x_j-pos, x_j-pos+1

d)

Second-order Features
x_i-pos, x_k-pos, x_j-pos
x_k-pos, x_j-pos
x_k-word, x_j-word
x_k-word, x_j-pos
x_k-pos, x_j-word

Table 3.1: Features used by system, $f(i,j)$, where x_i is the head and x_j the modifier in the dependency relation. x_i-word: word of head in dependency edge. x_j-word: word of modifier. x_i-pos: POS of head. x_j-pos: POS of modifier. x_i-pos+1: POS to the right of head in sentence. x_i-pos-1: POS to the left of head. x_j-pos+1: POS to the right of modifier. x_j-pos-1: POS to the left of modifier. b-pos: POS of a word in between head and modifier.

configuration. The second class of additional features represents the local context of the attachment, that is, the words before and after the head-modifier pair. These features take the form of POS 4-grams: The POS of the head, modifier, word before/after head and word before/after modifier. 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 B shows the feature representation for our example sentence over the edge (*hit*,*with*) for the example sentence in Figure 1.2.

As mentioned earlier, all of the runtime analysis relied on the fact that the calculation of $s(i,j)$ was $O(1)$, when in fact it is really linear in the number of features that are active for each edge. Table 3.1 shows that for each edge there are only a handful of bigram and unigram features as well as context POS features. More troubling are the POS features for all the words in-between the two words in the edge - this in fact makes the calculation of $s(i,j)$ at least $O(n)$ making the projective parsing algorithms $O(n^4)$ and the non-projective parsing algorithm $O(n^3)$. However, a feature can be active at most once for each distinct POS, e.g., if there are two proper nouns (*NNP*) between x_i and x_j, the feature is active only once.

We define a table $pos(i, j)$ that is the set of POS tags for all the words in-between x_i and x_j. This table can be calculated statically before parsing in $O(n^2)$ using a dynamic programming algorithm that fills in the table for successively larger sub-strings. It is easy to see that $pos(i, j)$ is equal to $pos(i, j-1)$ plus the POS of x_{j-1}, if it is not already in $pos(i, j-1)$, which can be calculated in $O(1)$ using a hash map. We have now only added (not multiplied) a factor of $O(n^2)$ to the runtime. Using this table we can now calculate $s(i, j)$ without enumerating all words in-between.

The result is that our grammar constant is now, in the worst case, on the order of the number of distinct POS tags, which is typically around 40 or 50, plus the handful of unigram, bigram and context features. When compared to the grammar constant for phrase-structure parsers this is still very favorable.

3.2.1 Second-Order Features

Since we are also building a second-order parsing model, we must define $\mathbf{f}(i, k, j)$. We let the first set of features be all those in the definition of $\mathbf{f}(i, j)$. This is possible by simply ignoring the middle index and creating features only on the original head-modifier indexes. In addition to these features, we add the features in Table 3.1d.

These new features have two versions. The first is exactly as described in the table. The second conjoins them with the distance between the two siblings as well as the direction of attachment (from the left or right). These features were tuned on a development set. We tried additional features, such as the POS of words in-between the two siblings, but the set defined here seemed to provide optimal performance.

Again, Appendix B provides a concrete example of this feature representation.

3.2.2 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 are used for all of the languages for which we study in this work. The only difference between the parsers is that they are trained on language specific data sets. In Chapter 5 we discuss the addition of morphological features, which can be shown to be useful for highly inflected languages.

3.3 Adding Labels

Though most large scale evaluations of dependency parsers have dealt with unlabeled dependency accuracies, it is clear that labeled dependency structures like those in Figure 1.4 are more desirable for further processing since they identify not only the modifiers of a word, but also their specific syntactic or grammatical function. As a result, many standard dependency parsers already come with the ability to label edges [82, 105, 127]. In this section we extend the algorithms previously presented to include syntactic labels. We assume throughout this section that there is a known set $t \in T$ of labels and that our training data is annotated with this information.

One simple approach would be to extract the highest scoring unlabeled trees and then run a classifier over its edges to assign labels. Dan Klein recently showed that labeling is relatively easy and that the difficulty of parsing lies in creating bracketings [75], providing evidence that a two-stage approach may prove good enough. However, for the sake of completeness, we will provide details and experiments on learning dependencies trees with labels in a single stage as well as a two-stage system.

3.3.1 First-Order Labeling

For first-order parsing we will change our edge score function to include label information,

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

In other words, we now define the score of the edge as the dot product between a weight vector and a high dimensional feature representation of the edge *and that edges label*. Hence the score of a dependency tree will now be,

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

Both the Eisner projective and the Chu-Liu-Edmonds non-projective parsing algorithm can be modified so that only an $O(|T|n^2)$ factor is added (not multiplied) to the run-time.

Consider a label t for an edge (i, j) such that,

$$t = \arg\max_{t'} \ \mathbf{w} \cdot \mathbf{f}(i, j, t')$$

It is easy to show that, if the highest scoring tree $\boldsymbol{y}$ under some weight vector $\mathbf{w}$ contains the edge (i, j), then the label of this edge must be t. Consider some $\boldsymbol{y} = \arg\max_{\boldsymbol{y}} s(\boldsymbol{x}, \boldsymbol{y})$ and an arbitrary edge $(i, j, t) \in \boldsymbol{y}$. Assume that $t \neq \arg\max_{t'} \mathbf{w} \cdot \mathbf{f}(i, j, t')$. We can simply replace the label of this edge with the label that maximizes the edge score to produce a higher scoring tree. Thus, by contradiction, it must be the case that $t = \arg\max_{t'} \mathbf{w} \cdot \mathbf{f}(i, j, t')$.

Using this fact, we can define a table $bt(i, j)$ such that each entry stores the best label for that particular edge, i.e.,

$$bt(i,j) = \arg\max_{t'} \mathbf{w} \cdot \mathbf{f}(i,j,t')$$

This table takes $O(|T|n^2)$ to calculate and can be calculated statically before parsing. During parsing we define $s(i,j) = s(i,j,bt(i,j))$ and we can run the algorithm as before without increasing complexity. Thus the new complexity for the projective parsing algorithm is $O(n^3 + |T|n^2)$ and $O(|T|n^2)$ for the non-projective algorithm.

3.3.2 Second-Order Labeling

We redefine the second-order edge score to be,

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

This is the score of creating an edge from word x_i to x_j with edge label t such that the last modifier of x_j was x_k. It is easy to show that we can use the same trick here and statically calculate,

$$bt(i,k,j) = \arg\max_{t'} \mathbf{w} \cdot \mathbf{f}(i,k,j,t')$$

and set $s(i,k,j) = s(i,k,j,bt(i,k,j))$ to allow us to apply our old parsing algorithms[2]. The result is a $O(|T|n^3)$ complexity for the second-order projective extension since it will take $O(|T|n^3)$ to compute $bt(i,k,j)$.

We could have defined our second-order edge score as,

$$s(i,k,j,t',t) = \mathbf{w} \cdot \mathbf{f}(i,k,j,t',t)$$

[2]Additional care is required in the non-projective second-order approximation since a change of one edge could result in a label change for multiple edges.

where t' is the label for the edge (i, k). This would allow us to model common sibling edge labels, e.g., possibly preventing a verb from taking adjacent subjects. However, inference under this definition becomes $O(|T|^2 n^3)$, which can be prohibitive if the number of labels is large.

3.3.3 Two-Stage Labeling

As mentioned earlier, a simple solution would be to create a second stage that takes the output parse $\boldsymbol{y}$ for sentence $\boldsymbol{x}$ and classifies each edge $(i, j) \in \boldsymbol{y}$ with a particular label t. Though one would like to make all parsing and labeling decisions jointly to include the shared knowledge of both decisions when resolving any ambiguities, joint models are fundamentally limited by the scope of local factorizations that make inference tractable. In our case this means we are forced only to consider features over single edges or pairs of edges in the tree. Furthermore, the complexity of inference increases by a factor of the number of possible labels, which can be very detrimental if the label set is large. However, in a two-stage system we can incorporate features over the entire output of the unlabeled parser since that structure is fixed as input. The simplest two-stage method would be to take as input an edge $(i, j) \in \boldsymbol{y}$ for sentence $\boldsymbol{x}$ and find the label with highest score,

$$t = \arg\max_{t} \; s(t, (i, j), \boldsymbol{y}, \boldsymbol{x})$$

Doing this for each edge in the tree would produce the final output. Such a model could easily be trained using the provided training data for each language. However, it might be advantageous to know the labels of other nearby edges. For instance, if we consider a head x_i with dependents $x_{j_1}, \ldots, x_{j_M}$, it is often the case that many of these dependencies will have correlated labels. To model this we treat the labeling of the edges $(i, j_1), \ldots, (i, j_M)$

as a sequence labeling problem,

$$(t_{(i,j_1)}, \ldots, t_{(i,j_M)}) = \boldsymbol{t} = \arg\max_{\boldsymbol{t}} \; s(\bar{t}, i, \boldsymbol{y}, \boldsymbol{x})$$

We use a first-order Markov factorization of the score

$$\boldsymbol{t} = \arg\max_{\boldsymbol{t}} \; \sum_{m=2}^{M} s(t_{(i,j_m)}, t_{(i,j_{m-1})}, i, \boldsymbol{y}, \boldsymbol{x})$$

in which each factor is the score of assigning labels to the adjacent edges (i, j_m) and (i, j_{m-1}) in the tree $\boldsymbol{y}$. We attempted higher-order Markov factorizations but they did not improve performance uniformly across languages and training became significantly slower.

For score functions, we use the standard dot products between high dimensional feature representations and a weight vector. Assuming we have an appropriate feature representation, we can find the highest scoring label sequence with Viterbi's algorithm. We use the MIRA online learner to set the weights since we found it trained quickly and provide good performance. Furthermore, it made the system homogeneous in terms of learning algorithms since that is what is used to train our unlabeled parser. Of course, we have to define a set of suitable features. We used the following:

- **Edge Features:** Word/pre-suffix/POS feature identity of the head and the modifier (suffix lengths 2 and 3). Does the head and its modifier share a prefix/suffix. Attachment direction. s the modifier the first/last word in the sentence?
- **Sibling Features:** Word/POS/pre-suffix feature identity of the modifiers left/right siblings in the tree (siblings are words with same head in the tree)? Do any of the modifiers siblings share its POS?
- **Context Features:** POS tag of each intervening word between head and modifier. Do any of the words between the head and the modifier have a head other than the head? Are any of the words between the head and the modifier not a descendent of the head (i.e. non-projective edge)?

- **Non-local:** How many modifiers does the modifier have? Is this the left/right-most modifier for the head? Is this the first modifier to the left/right of the head?

Various conjunctions of these were included based on performance on held-out data. Note that many of these features are beyond the scope of the edge based factorizations of the unlabeled parser. Thus a joint model of parsing and labeling could not easily include them without some form of re-ranking or approximate parameter estimation.

3.4 Summary of Chapter

In this chapter we presented the primary contribution of this work – the formulation of dependency parsing as the maximum spanning tree problem. The MST view of dependency trees has many advantages. Primarily it leads to efficient first and second-order projective parsing algorithms as well as efficient first-order non-projective parsing algorithms. Furthermore, we also gave an approximate second-order non-projective parsing algorithm for which we will empirically justify in the next chapter. We also showed how all algorithms can be extended to provide labeled dependencies. The MST view thus gives a uniform and algorithmically justifiable view of dependency parsing for natural language, which we will exploit together with the learning algorithms in Chapter 2 to produce efficient and accurate parsers for a variety of languages.

Chapter 4

Dependency Parsing Experiments

Parts of this chapter are drawn from material in [91, 95, 94].

4.1 Data Sets

We performed these experiments on three sets of data, the Penn English Treebank [84], the Czech Prague Dependency Treebank (PDT) v1.0 [56, 57] and the Penn Chinese Treebank [150]. For the English data we extracted dependency trees using the rules of Yamada and Matsumoto [151], which are similar, but not identical, to those used by Collins [25] and Magerman [83]. These rules are given in Appendix A. Because the dependency trees are extracted from the phrase-structures in the Penn Treebank, they are by construction 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 [151] and did not include punctuation in the calculation of accuracies. For the test set, the number of words without punctuation is 49,892. Since our system assumes part-of-speech information as input, we used the maximum entropy part-of-speech tagger of Ratnaparkhi [110] to provide tags for the development and testing data. The number of features extracted from the Penn Treebank were $6,998,447$ for the first-order model and $7,595,549$ for the second-order model.

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 for the 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. [29]. The number of features extracted from the PDT training set were $13,450,672$ for the first-order model and $14,654,388$ for the second-order model.

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 Chinese data set was created by extracting dependencies from the Penn Chinese Treebank [150] using the head rules that were created by a native speaker primarily for the purpose of building a machine translation system. Again, because the dependency trees are extracted from the phrase-structures, they are by construction exclusively projective. We split the data into training and testing by placing every tenth sentence in the data into the test set. We use gold POS tags for this data set since we have not yet trained a Chinese POS tagger. The number of features extracted from the Penn Chinese Treebank training set were $2,985,843$ for the first-order model and $3,346,783$ for the second-order model. Unlike English and Czech, we did not include any 5-gram prefix features.

4.2 Unlabeled Dependencies

This section reports on the most extensive experiments we have completed to date on unlabeled dependencies. It is primarily divided into two sections, projective and non-projective results. For the non-projective results we focus on the Czech data since it contains this particular phenomenon.

The first two sections compare pure dependency parsers only, i.e., those parsers trained only on dependency structures. We include a third section that compares our parsers to lexicalized phrase-structure parsers, which have been shown to produce state-of-the-art dependency results [151].

4.2.1 Projective Parsing Results

We compare five systems,

- **Y&M2003:** The Yamada and Matsumoto parser [151] is a discriminative parser based on local decision models trained by an SVM. These models are combined in a shift-reduce parsing algorithm similar to Ratnaparkhi [111].
- **N&S2004:** The parser of Nivre and Scholz [105] is a memory based parser with an approximate linear parsing algorithm.
- **N&N2005:** The parser of [104], which is an extension of N&S2004 to Czech. This paper presents both a projective and non-projective variant. We report the non-projective results in the next section.

	English		Czech-A		Chinese	
	Accuracy	**Complete**	**Accuracy**	**Complete**	**Accuracy**	**Complete**
Y&M2003	90.3	38.4	-	-	-	-
N&S2004	87.3	30.4	-	-	-	-
N&N2005	-	-	78.5	20.6	-	-
1^{st}-order-proj	90.7	36.7	83.0	30.6	79.7	27.2
2^{nd}-order-proj	91.5	42.1	84.2	33.1	82.5	32.6

Table 4.1: Unlabeled projective dependency parsing results. *Accuracy* is the percentage of words modifying the correct head. *Complete* is the percentage of sentences for which the entire predicted dependency graph was correct.

- **1st-order-proj:** This parser uses the Eisner first-order projective parsing algorithm combined with the MIRA learning framework.
- **2nd-order-proj:** This parser uses the second-order extension of the Eisner algorithm combined with the MIRA learning framework.

Results are shown in Figure 4.1. Not all systems report all results. Across all languages the parsers we have developed here provide state-of-the-art performance without any language specific enhancements. It can be argued that the primary reason for this improvement is the parsers ability to incorporate millions of rich dependent features, which is not possible in for the history based models [104, 105]. The Yamada and Matsumoto [151] SVM parser also has this ability. However, their locally trained model can suffer from the label bias problem [80] as well as error propagation during their shift-reduce search. Furthermore, we can also see that the introduction of second-order features improves parsing substantially for all languages, as expected.

k-best MIRA Approximation

We need to determine how justifiable is the k-best MIRA approximation, in particular when $k = 1$. Table 4.2 indicates the test accuracy and training time on English for the k-best MIRA first-order model with $k = 1, 2, 5, 10, 20$. Even though k-best parsing multiplies asymptotic parsing complexity by a $k \log k$ factor, empirically the training times seem to

	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 4.2: Evaluation of k-best MIRA approximation. These experiments were run on a 2.4GHz 32-bit machine with 2G of memory.

scale linearly with k. Peak performance is achieved for low k with a slight degradation around $k = 20$. We speculate that the 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.

4.2.2 Non-projective Parsing Results

As mentioned earlier, 23% of the sentences in the PDT contain at least one non-projective dependency and roughly 2% of all dependencies are non-projective. In this section we examine the performance of our non-projective parsers on the entire PDT (data set *Czech-A*) as well as a subset containing only those sentences with non-projective dependencies (data set *Czech-B*).

We compare five systems,

- **N&N2005:** The parser of Nivre and Nilsson [104] is a memory based parser like [105]. This parser models non-projective dependencies through edge transformations encoded into labels on each edge. For instance a label can encode a parental raises in the tree (when a edge is raised along the spine towards the root of the tree).
- **1st-order-proj:** The first-order projective parser from Section 4.2.1.
- **2nd-order-proj:** The second-order projective parser from Section 4.2.1.
- **1st-order-non-proj:** This parser uses the Chu-Liu-Edmonds MST algorithm as described in Section 3.1.1.
- **2nd-order-non-proj:** This parser uses the approximate second-order non-projective parsing algorithm described in Section 3.1.2.

	Czech-A		Czech-B	
	Accuracy	**Complete**	**Accuracy**	**Complete**
N&N2005	80.0	31.8	-	-
1^{st}-order-proj	83.0	30.6	74.4	0.0
2^{nd}-order-proj	84.2	33.1	74.6	0.0
1^{st}-order-non-proj	84.1	32.2	81.0	14.9
2^{nd}-order-non-proj	85.2	35.9	81.9	15.9

Table 4.3: Unlabeled non-projective dependency parsing results.

Results are shown in Figure 4.3. This table shows us that for both the first and second-order models, modeling non-projective dependencies leads to an improvement in performance of around 1% absolute. Especially surprising is that the second-order approximate algorithm leads to such a large improvement. The most likely reason is that the approximate post-process edge transformations are incorporated into the online learning algorithm, which allows the model to adjust its parameters for common mistakes made during the approximation. Thus the algorithm learns quickly that the best non-projective tree is typically only one or two edge transformations away from the highest scoring projective tree. The robustness of discriminative online learning algorithms to approximate inference will be discussed further in Chapter 8.

As mentioned earlier, we have not been able to put a worst-case complexity on our approximate second-order non-projective parsing algorithm. However, in terms of runtime, our projective $O(n^3)$ second-order model runs in 16m32s and our non-projective approximation in 17m03s on the Czech evaluations data. Clearly, the post-process approximate step of inference does not in practice add too much time. This is because each sentence typically contains only a handful of non-projective dependencies. As a result the algorithm will learn not to adjust too many edges after the initial projective parsing step.

	English			
	Accuracy	**Complete**	**Complexity**	**Time**
Collins [7, 25]	91.4	42.6	$O(n^5)$	98m 21s
Charniak [16]	92.1	45.3	$O(n^5)$	31m 24s
2^{nd}-order-proj	91.5	42.1	$O(n^3)$	8m46s

Table 4.4: Results comparing our system to those based on the Collins and Charniak parsers. *Complexity* represents the computational complexity of each parser and *Time* the CPU time to parse sec. 23 of the Penn Treebank.

4.2.3 Lexicalized Phrase-Structure Parsers as Dependency Parsers

It is well known that dependency trees extracted from lexicalized phrase-structure parsers [16, 25] are typically more accurate than those produced by pure dependency parsers [151]. We compared our system to the Charniak parser [16] and the Bikel re-implementation of the Collins parser [7, 25] trained with the same head rules as our system. By taking the phrase-structure output of the parser and running the automatic head rules over it, we were able to extract the dependencies. Table 4.4 shows the results comparing our second-order projective system to the Charniak and Collins parser for English. All systems are implemented in Java and run on the same machine, except for the Charniak parser which was implemented in C. The Charniak parser clearly has an accuracy advantage over both our parser and the Collins parser, whose performance is indistinguishable from our discriminative parser. In terms of complexity and runtime, our system is a large improvement over both parsers. In particular, the parser is still significantly faster than the Charniak parser that is implemented in C.

We should note that comparing our parser to dependencies automatically extracted from phrase-structure is misleading in a number of ways. First of all our parser was trained and designed to maximize dependency accuracy, whereas the parsers of Collins and Charniak were trained and designed to maximize phrase-structure accuracy. Furthermore, extracting dependencies from phrase-structure assumes that our head rules are a priori correct.

	Czech-A	
	Accuracy	**Complete**
Collins-proj [29, 154]	82.5	-
Charniak-proj [59, 154]	84.3	-
Charniak-non-proj [59]	85.1	-
2^{nd}-order-proj	84.2	33.1
2^{nd}-order-non-proj	85.2	35.9

Table 4.5: Results comparing our system to those based on extensions of the Collins and Charniak parsers to Czech.

A better means might be to actually extract the lexicalization produced by these parsers. However, this is difficult since these parsers by default do not produce their internal lexicalization and sometimes exclude certain punctuation from the output since accuracies are calculated independently of punctuation.

Thus, a better comparison would be on the Czech data set since it contains hand annotated dependencies and there has been extensive work on extending the generative models of Collins and Charniak to Czech [29, 59, 154]. We compare three systems to our second-order models in Table 4.5. *Collins-proj* is the Collins parser extended to Czech as described by [29]. *Charniak-proj* is the Charniak parser extended to Czech as reported in [59, 154]. Both of these models are based on phrase-structure parsers and are limited to producing projective trees. *Charniak-non-proj* is the recently published parser of Hall and Nóvák [59] that takes the best parse from the Charniak parser and creates a new set of parses based on small transformations that can introduce non-projective edges. A maximum entropy re-ranker then picks the best parse from this new set. We do not report parsing times since code for these systems is unavailable.

We can see that our second-order approximate parser performs slightly better than the Charniak parser modified to include non-projective edges. This is particularly promising since our parser is a single model trained with our approximate inference algorithm, whereas Hall and Nóvák require two models, the original Charniak parser plus a non-projective re-ranker.

It is rather disappointing that our discriminative parser does not in general outperform these generative models (in particular the Charniak parser). However, as we will show in Chapter 7, we can incorporate features into our discriminative model that represent the parsing decisions of both the Collins and Charniak parsers leading to a significant improvement in accuracy.

4.2.4 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 15 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. Section 6.1.2 addresses this issue directly.

4.3 Labeled Dependencies

In this section we report results for the first-order labeled dependency models described in Section 3.3.1 as well as a two-stage labeling system, i.e., one that learns a model to label the output of an unlabeled dependency parser. We report results for English on the WSJ using sections 02-21 for training, section 22 for development and section 23 for evaluation. To extract labeled dependency trees from this data, we took the label of the highest

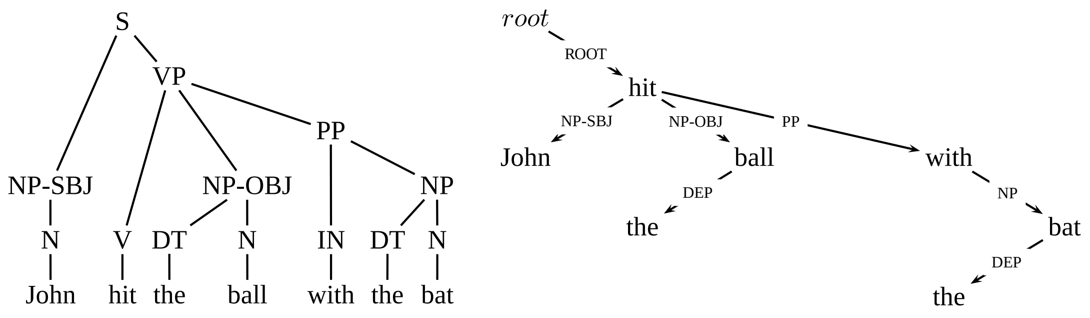


Figure 4.1: Converting a phrase-structure tree to a labeled dependency tree.

ADJP	LST	NX	S	VP
ADVP	NAC	PP	SBAR	WHADVP
CONJP	NP	PRN	SBARQ	WHNP
DEP	NP-OBJ	PRT	SINV	X
FRAG	NP-PRD	QP	SQ	ROOT
INTJ	NP-SBJ	ROOT	UCP	

Figure 4.2: Labels extracted from WSJ using [106].

node in the phrase-structure tree for which that word is the lexical head. For example, the phrase-structure tree for *John hit the ball with the bat* would be transformed into a labeled dependency tree as shown in Figure 4.1. Running an extraction script (we used Penn2Malt [106]) resulted in the set of 29 labels shown in Figure 4.2. The labels are standard from the Penn Treebank, except the labels 'DEP', which is meant to represent a generic dependency, and 'ROOT', which is designated for modifiers of the artificial root node.

In the next sections we report results for English on *Labeled Accuracy* and *Unlabeled Accuracy*. The former measures the number of words who correctly identified their head and assign the correct label to the edge and the latter measure normal unlabeled dependency parsing accuracy (as discussed in the last section). We always use the projective parsing algorithms in this evaluation since the English data set is exclusively projective.

In Chapter 5 we present labeled experiments for languages other than English.

	English	
	Labeled Accuracy	**Unlabeled Accuracy**
1^{st}-order-proj with joint labeling	88.7	90.9

Table 4.6: First-order labeling results for English.

4.3.1 First-Order Results

Results for the first-order labeling model (Section 3.3.1) are shown in Table 4.6. The first thing to note is that even with a large set of possible labels (28), overall accuracy drops only 2% absolute, which roughly says that the labeling accuracy is 97.6% accurate over correctly identified dependencies. This is a very promising result. However, we will see in Chapter 5 that labeling for English is typically much easier than for all other languages and can usually be deterministically decided based on the part-of-speech tags of the modifier and head words in the dependency.

Another interesting property is that the unlabeled accuracy actually improves (from 90.7 to 90.9). This is consistent with previous results [105] and displays that learning to label and find dependencies jointly will help overall performance. However, this benefit does come at the expensive of computation, since the training and inference have an added $O(Tn^2)$ term, which in practice leads to roughly to run times on the order of 3 times slower than the unlabeled system. The fact that second-order joint parsing and labeling results in a run-time complexity of $O(Tn^3)$ made it unreasonable to train large models in a practical amount of time. In the next section, it will be shown that learning to label and find dependencies separately does not degrade performance and has much nicer computational properties.

	English	
	Labeled Accuracy	**Unlabeled Accuracy**
1^{st}-order-proj with joint labeling	88.7	90.9
1^{st}-order-proj with 2-stage labeling	88.8	90.7
2^{nd}-order-proj with 2-stage labeling	89.4	91.5

Table 4.7: Two-stage labeling results for English.

4.3.2 Two-Stage Results

Results for two-stage labeling (Section 3.3.3) are shown in Table 4.7. From this table, we can see that a two-stage labeler with a rich feature set does just as well as a joint labeler that is restricted to features over local factorizations (88.8 vs. 88.7). The advantage of the two stage labeler is that it is much quicker to train and run, with a complexity of $O(n^3 + T^2n)$, where the T^2 factor comes from the fact that we have to run Viterbi's algorithm. Furthermore, the complexity for the second-order model is identical at $O(n^3 + T^2n)$ and can be trained very efficiently. Results for this system are also shown in Table 4.7 and once again display the advantage of a second-order model.

4.4 Summary of Chapter

In this chapter we presented an empirical evaluation of the dependency parsers described in this work. For unlabeled dependencies we compared them to the current best systems favorably. Furthermore, we showed that they have competitive performance with lexicalized phrase-based parsers, but are much more computationally efficient. Finally, we presented results for the labeled parser and showed that two-stage parsing can provide equal or better performance to a joint parser and labeler due to its ability to include features over the entire dependency tree.

Chapter 5

Parsing 14 languages with One Model

Part of the material in this chapter is drawn from [92].

An important question for any parsing model is, how well does it apply to new languages? In this section we aim to show that the models described in this work are, for the most part, language independent. We do this by evaluating the models on 14 diverse languages. This data set includes the 13 standard dependency data sets provided by the organizers of the 2006 CoNLL shared task [13] plus the English data set we described in Section 4.1. We show that our standard parser with little to no language specific enhancements achieves high parsing accuracies across all languages (relative to state-of-the-art). This is a very promising result and a strong argument for the applicability of the parsers in this work. We used the two-stage parsing model described in Section 3.3.3 for all experiments in this chapter.

language	Ar	Ch	Cz	Da	Du	Ge	Ja	Po	Sl	Sp	Sw	Tu	Bu	En
genres	1: ne	6	3	8+	5+	1: ne	1: di	1: ne	1: no	9	4+	8	12	1: ne
annotation	d	c+f	d	d	dc+f	dc+f	c+f	dc+f	d	c(+f)	dc+f/d	d	c+t	c+f
training data														
tokens (k)	54	337	1249	94	195	700	151	207	29	89	191	58	190	950
%non-scor.	8.8	0.8	14.9	13.9	11.3	11.5	11.6	14.2	17.3	12.6	11.0	33.1	14.4	9.1
sents (k)	1.5	57.0	72.7	5.2	13.3	39.2	17.0	9.1	1.5	3.3	11.0	5.0	12.8	39.8
tokens/sent	37.2	5.9	17.2	18.2	14.6	17.8	8.9	22.8	18.7	27.0	17.3	11.5	14.8	23.9
LEMMA	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	No	No
CPOSTAGs	14	22?	12	10	13	52	20	15	11	15	37	14	11	36
POSTAGs	19	303?	63	24	302	52	77	21	28	38	37	30	53	36
FEATS	19	0	61	47	81	0	4	146	51	33	0	82	50	0
DEPRELs	27	82	78	52	26	46	7	55	25	21	56	25	18	28
D. H.=0	15	1	14	1	1	1	1	6	6	1	1	1	1	1
%HEAD=0	5.5	16.9	6.7	6.4	8.9	6.3	18.6	5.1	5.9	4.2	6.5	13.4	7.9	4.2
%H. left	82.9	24.8	50.9	75.0	46.5	50.9	8.9	60.3	47.2	60.8	52.8	6.2	62.9	47.0
%H. right	11.6	58.2	42.4	18.6	44.6	42.7	72.5	34.6	46.9	35.1	40.7	80.4	29.2	48.8
H.=0/sent	1.9	1.0	1.0	1.0	1.2	1.0	1.5	1.0	0.9	1.0	1.0	1.0	1.0	1.0
%n.p. edges	0.4	0.0	1.9	1.0	5.4	2.3	1.1	1.3	1.9	0.1	1.0	1.5	0.4	0.0
%n.p. sents	11.2	0.0	23.2	15.6	36.4	27.8	5.3	18.9	22.2	1.7	9.8	11.6	5.4	0.0

Table 5.1: Properties of the data for the 14 languages. This table has been taken and modified from [13], with permission.

5.1 Data Sets

In this section we briefly describe the source and size of each data set. For further details see [13]. Each of these data sets consist of a number of sentences annotated with labeled dependency graphs satisfying the tree constraint.

5.1.1 General Properties of the Data

Table 5.1 outlines general properties of the data for the 14 languages: Arabic (Ar), Chinese (Ch), Czech (Cz), Danish (Da), Dutch (Du), German (Ge), Japanese (Ja), Portuguese (Po), Slovene (Sl), Spanish (Sp), Swedish (Sw), Turkish (Tu), Bulgarian (Bu) and English (En). Table 5.1 contains the following information,

- **genres:** The number of genres in the data set. ne: news, no: novel/literature, di:dialogue (other if not specified).
- **annotation:** The original annotation in the treebank. d=dependency, c=constituents, dc=discontinuous constituents, +f=with functions, +t=with types (labels).
- **tokens:** The number of tokens in the data set.
- **%non-scor:** Number of non-scoring tokens in the data (usually punctuation).

- **sents:** Number of sentences in the data set.
- **LEMMA:** Does the data set contain lemmas (in addition to inflected forms)?
- **CPOSTAGs:** Number of coarse-grained part-of-speech tags in data.
- **POSTAGs:** Number of part-of-speech tags in data.
- **FEATS:** Does the data set contain morphological feature information?
- **DEPRELs:** The number of edge labels in the data set.
- **D. H.=0:** The number of labels for dependencies involving the root of the graph.
- **%HEAD=0:** Percentage of tokens whose head is the root of the graph.
- **%H. left:** Percentage of tokens whose head is to its left (not including tokens whose head is the root).
- **%H. right:** Percentage of tokens whose head is to its right (not including tokens whose head is the root).
- **H.=0/sent:** Average number of words per sentence whose head is the root of the graph.
- **%n.p. edges:** Percentage of non-projective edges in data.
- **%n.p. sents:** Percentage of sentences with at least one non-projective edge.

5.1.2 Data Sources and Specifics

In this section we briefly describe the source of each data set used as well as if any transformations were required. Note that the size of each test set was chosen to approximately make the number of scoring tokens equivalent for each language, except English, which is not officially one of the CoNLL data sets.

Arabic

Dependencies were taken from the Prague Arabic Dependency Treebank [58, 129], which is a proper dependency treebank. The data is drawn from a variety of news sources including Agence France Presse, Al Hayat News Agency, Ummah Press Service, An Nahar News

Agency, Xinhua News Agency. The training set consists of 1460 sentences and the test set consists of 146 sentences.

Bulgarian

Dependencies were taken from the Bulgarian Treebank [124, 123, 126, 125, 108], which was converted to dependencies from a Head-Driven Phrase Structure format. The data is drawn from a wide variety of sources including, news text, textbooks and literature. The training set consists of 12,823 sentences and the test set consists of 398 sentences.

Chinese

Dependencies were taken from the Sinica Treebank [19], which is syntactically and semantically annotated corpus of phrasal structure including head and modifier information. The data is drawn from a wide variety of genres in the Sinica corpus [18]. The training set consists of 56,957 sentences and the test set consists of 867 sentences.

Czech

Dependencies were taken from the Prague Dependency Treebank [9], which is a proper dependency treebank. The data is drawn from from Czech newspapers and academic publishers. The training set consists of 72,703 sentences and the test set consists of 365 sentences.

Danish

Dependencies were drawn from the Danish Dependency Treebank [79], which is a proper dependency treebank. This data set also consists of a set of *secondary* edges, which break the tree constraint. For these experiments, only primary edges are considered. In Section 8.2 we discuss extensions to our parsing algorithms with the ability of producing secondary edges. This data is drawn from the PAROLE corpus which contains sentences from a variety of sources. The training set consists of 5,190 sentences and the test set consists of 322 sentences.

Dutch

Dependencies were drawn from the Alpino Treebank [148, 147], which is a corpus of syntactically annotated phrasal information. Dependencies were generated using standard head selection rules. The data is drawn from news text, spoken dialogue, sentences that are questions, and even sentences from a Dutch yellow pages. The training set consists of 13,349 sentences and the test set consists of 386 sentences.

English

Dependencies were created from the Penn Treebank [84] using the method described in Section 4.1. The training set consists of 39,832 sentences and the test set consists of 2,416 sentences.

German

Dependencies were taken from the TIGER Treebank [11], which is a corpus of sentences annotated under a Lexical Functional Grammar formalism. Dependencies are easily extracted in this data using deterministic head rules. The data is drawn from the German news source Frankfurter Rundschau. The training set consists of 39,216 sentences and the test set consists of 357 sentences.

Japanese

Dependencies were taken from the Verbmobil Treebank [73], which was converted from phrase-structure using pre-annotated head associations. The data is drawn from two way conversations negotiating business meetings. The training set consists of 17,044 sentences and the test set consists of 709 sentences.

Portuguese

Dependencies were taken from the Bosque subset of the Floresta Sintá(c)tica Treebank project [2], which was converted from phrase-structure to dependencies using a fixed set of head rules. The data is drawn from two news and journal sources. The training set consists of 9,071 sentences and the test set consists of 288 sentences.

Slovene

Dependencies were taken from the Slovene Dependency Treebank [41], which is a pure dependency treebank annotated in the same fashion as the Prague Dependency Treebank and Prague Arabic Dependency Treebank. The text contains sentences from a Slovene translation of *1984* by George Orwell. The training set consists of 1,534 sentences and the test set consists of 402 sentences.

Spanish

Dependencies were taken from the Cast3LB Treebank [23, 99, 22], which was converted from phrase-structure using a fixed set of head rules. The data is drawn from journals, literature and scientific text from Spain as well as South America. The training set consists of 3,306 sentences and the test set consists of 206 sentences.

Swedish

Dependencies were taken from Talbanken05 [102], which is a modern version of one of the oldest treebanks in existence, Talbanken76 [44, 139]. Talbanken05 is annotated with dependencies so no conversion was necessary. The data consists of both written text and transcribed speech. The training set consists of 11,042 sentences and the test set consists of 389 sentences.

Turkish

Dependencies were drawn from the METU-Sabanci Turkish Treebank [107, 4], which annotates dependencies between inflectional groups. Within an inflectional group all words modify the last. The data is drawn from a wide body of genres including news and literature. The training set consists of 4,997 sentences and the test set consists of 623 sentences.

5.2 Adding Morphological Features

One advantage of the CoNLL data sets is that they came with derived morphological features for each language. The types of features differed by data set so we incorporated them into our models in a general way.

For the unlabeled dependency parser we augmented the feature representation of each edge. Consider a proposed dependency of a modifier x_j for the head x_i, each with morphological features M_j and M_i respectively. We then add to the representation of the edge: M_i as head features, M_j as modifier features, and also each conjunction of a feature from both sets. These features play the obvious role of explicitly modeling consistencies and commonalities between a head and its modifier in terms of attributes like gender, case, or number.

For the second-stage labeler we used the following feature set,

- **Edge Features:** Word/pre-suffix/POS/morphological feature identity of the head and the modifier (suffix lengths 2 and 3). Does the head and its modifier share a prefix/suffix. Attachment direction. What morphological features do head and modifier have the same value for? Is the modifier the first/last word in the sentence?

- **Sibling Features:** Word/POS/pre-suffix/morphological feature identity of the modifiers left/right siblings in the tree (siblings are words with same head in the tree)? Do any of the modifiers siblings share its POS?
- **Context Features:** POS tag of each intervening word between head and modifier. Do any of the words between the head and the modifier have a head other than the head? Are any of the words between the head and the modifier not a descendent of the head (i.e. non-projective edge)?
- **Non-local:** How many modifiers does the modifier have? What morphological features does the grandparent and the modifier have identical values? Is this the left/right-most modifier for the head? Is this the first modifier to the left/right of the head?

This is identical to the old feature set, except where morphology features have been included.

5.3 Results

Based on performance from a held-out section of the training data, we used non-projective parsing algorithms for Czech, Danish, Dutch, German, Japanese, Portuguese and Slovene, and projective parsing algorithms for Arabic, Bulgarian, Chinese, English, Spanish, Swedish and Turkish[1]. Furthermore, for Arabic and Spanish, we used lemmas instead of inflected word forms since this seemed to alleviate sparsity in parameter estimates for these languages.

Results on the test sets are given in Table 9.1. Performance is measured through unlabeled accuracy, which is the percentage of words that correctly identify their head in the dependency graph, and labeled accuracy, which is the percentage of words that identify their head and label the edge correctly in the graph. Punctuation is ignored for all languages. For all languages except English, a token is considered punctuation if and only if

[1]Using the non-projective parser for all languages does not effect performance significantly.

	UA	LA
Arabic	79.3	66.9
Bulgarian	92.0	87.6
Chinese	91.1	85.9
Czech	87.3	80.2
Danish	90.6	84.8
Dutch	83.6	79.2
English	91.5	89.4
German	90.4	87.3
Japanese	92.8	90.7
Portuguese	91.4	86.8
Slovene	83.2	73.4
Spanish	86.1	82.3
Swedish	88.9	82.5
Turkish	74.7	63.2
Average	87.4	81.4

Table 5.2: Dependency accuracy on 14 languages. Unlabeled (UA) and Labeled Accuracy (LA).

all of its characters are unicode punctuation characters. For English we define punctuation identical to Yamada and Matsumoto [151].

These results show that a two-stage system can achieve a relatively high performance. In fact, for every language our models perform significantly higher than the average performance for all the systems reported in the CoNLL 2006 shared task [13] and represent the best reporting system for Arabic, Bulgarian, Czech, Danish, Dutch, German, Slovene and Spanish (English was not included in the shared task).

5.4 Summary of Chapter

In this chapter we showed that the discriminative spanning tree parsing framework is easily adapted across all these languages. Only Arabic, Turkish and Slovene have parsing accuracies significantly below 80%, and these languages have relatively small training sets and/or are traditionally difficult languages to parse. These results are very promising. In fact, they are state-of-the-art when compared to other parsers evaluated in the CoNLL shared task

[13].

Section 6.2 and Appendix C provide a detailed error analyses.

Chapter 6

Analysis of Parsing Results

Parts of the material in this chapter is drawn from [92].

In this chapter we attempt to provide an analysis of common errors in both the English parsing models as well as a brief error analysis of the models for all other languages. Furthermore, we present experiments that tease apart the contribution of various features in the models. For a detailed quantitative error analysis for all sections, see Appendix C.

6.1 English

6.1.1 Error Analysis

For these experiments, we used sections 02-21 of the Penn Treebank to train a parsing model for English. We then used the standard test data to analyze the errors (section 23). Throughout the development of the parser, we had only used the development section (22) to analyze errors and modify the models. Now that we have fixed the parsing model, we wish to gain an insight on the kinds of errors that are being made for the reported results.

	English
	Accuracy
Standard	91.5
Top2POSTraining	91.9
GoldPOS	93.1

Table 6.1: English parsing errors relative to part-of-speech tagging errors.

Part-of-speech tagging errors

One aspect of our system that differs from at least one state-of-the-art parsing model (Charniak [16]) is that part-of-speech (POS) tags for the input sentence are fixed before parsing. This of course can lead to a propagation of errors if the POS tagger is not accurate. Fortunately, POS tagging is very well studied with the best parsers reporting accuracies near human capabilities. For instance, the POS tagger we used [110] was 96.3% accurate on the test data. Though this is a high number, the fact that parsing accuracies are also over 90% suggests that this could be a significant source of error, which in fact it is.

Table 6.1 shows the parsing accuracies of three systems. The first system, *Standard*, is the English second-order model from Section 4. The final system, *GoldPOS*, is identical to Standard, except that at test time we use the true POS tags for the data. This final system represents the upper-bound on parsing accuracy relative to POS tag information.

It is clear from the difference in accuracies between *Standard* and *Gold* that propagated POS tagging errors account for a large chunk of the remaining parsing errors, around 20% of it in fact. This result definitely suggests that a parsing model that predicts POS tags and dependencies jointly should be investigated to alleviate this. However, once there is POS ambiguity, exact parsing for the second-order projective model becomes $O(|P|^3 n^3)$, where $|P|$ is the number of unique POS tags. This presents a difficulty since $|P|$ is usually around than 40. However, one interesting approach to try is to fix the number of possible POS tags for a given token to a small set of likely candidates during training/testing, say of size p. These fixed tags can be provided from the p-best outputs of most statistical classification algorithms. The resulting parsing algorithm would be $O(p^3n^3)$, which seems a little more reasonable.

It is easy to motivate such an approach by observing that current POS taggers get near 99% recall if we consider the top 2 POS tags for each token. However, even with $p = 2$, we are multiplying the run-time by a factor of 8. Training already takes over 10 hours for English, thus multiplying this by 8 is cumbersome for use in current systems. However, for the sake of completeness, we implemented and trained a second-order parsing model that takes as input the top 2 POS tags for each token based on a maximum entropy tagger implemented in MALLET [87]. It took approximately 5 days to train the new parsing model, which resulted in an improvement in parsing accuracy from 91.5 to 91.9 (Table 6.1, system *Top2POSTraining*). For the context POS features used in the model, we simply fixed them as the single most likely POS tag in order to make inference tractable.

Unfortunately, it is easy to show that incorporating POS ambiguity into the first-order non-projective MST parsing algorithm makes the problem NP-hard.

Errors by Sentence Length

Another quantitative measure of error is by sentence length. It is natural for longer sentences to contain more errors absolute, since there are more simply more parsing decisions. Similarly, it is also natural for parsers to perform on average worse for longer sentences. However, the latter is usually for two reasons,

1. Longer sentences are more likely to have conjunctions, prepositions and multi-verb constructions, which typically cause most parsing errors (see the rest of this section for more details).

2. Often, greedy parsing algorithms will make early mistakes, causing the propagation of errors. Similarly, parsers based on a pruned dynamic programming search

	English		
	Accuracy	**Complete**	**Root**
0-10	94.3	85.6	96.3
11-20	92.7	54.2	95.8
21-30	91.5	33.7	94.0
31-40	91.1	21.7	93.3
41-50	90.4	8.9	93.3
> 50	88.7	9.4	97.2

Table 6.2: English parsing errors by sentence length.

may prematurely eliminate correct solutions early, again leading to error propagation. Longer sentences result in more opportunities to make propagated search mistakes.

There is no real guard against the former, except creating richer parsing models. However, for the latter, we would expect our system to be relatively immune since we use exact search without pruning and without greedy parsing decisions.

Table 6.2 presents parsing accuracy for sentences of varying length. In terms of standard dependency accuracy, we see an expected drop off in performance as sentence length increases. For complete parse accuracy we see a massive drop off, where the model rarely gets longer sentences correct and almost always gets shorter sentences correct. However, note that the accuracy for the root dependency (column Root) does not drop off as drastically as complete sentence accuracy for larger sentences. In fact, root accuracy drops at a rate very close to overall accuracy. This result suggests that, even though more mistakes are made for longer sentences, errors are not necessarily being propagated by bad mistakes made early, since the parsing model still tends to have good performance on identifying the root of the tree.

Errors due to Training Sentence Order and Online Learning

One important aspect of our learning algorithm is its efficiency. This is primarily a result of its online nature resulting in single instance parameter optimization. A natural concern

		English	
		Accuracy	**Complete**
Averaging	Original	91.5	42.1
	Random 1	91.5	41.7
	Random 2	91.5	41.9
	Random 3	91.5	41.9
	Random 4	91.5	41.6
	Random 5	91.6	42.5
No Averaging	Original	90.3	37.7
	Random 1	89.5	34.9
	Random 2	89.9	35.9
	Random 3	89.8	35.9
	Random 4	90.2	36.2
	Random 5	89.8	36.1

Table 6.3: English parsing errors relative to permutations of sentence order in the training set.

would be whether or not the order in which instances are seen during training has an impact on the final parsing accuracies. In particular, one would expect those sentences seen near the end of training to contribute more than those seen earlier. In this section we will provide results alleviating such concerns and argue that parameter averaging solves this problem.

To show this we created five new training sets, each a random permutation of the original training set. We then trained two models per data set (including the original data set). The first model is standard MIRA with parameter averaging and the second model is MIRA without parameter averaging. Results are shown in Table 6.3.

This table clearly indicates that with parameter averaging, sentence order has no effect on the accuracy of the final model. However, once averaging is turned off, we see that a disparity between accuracy does occur (nearly 1% absolute). Note that the best parser without averaging is the original. We speculate that this is because the sentences in that data set are chronological (sections 02-21 of the treebank). Thus, the latter sentences (from section 21) should be closer in distribution to the test data (section 23).

Errors by Modifier Type

In this section we aim to get a better picture of the kinds of dependency errors occurring in the model. In particular, dependency accuracy is defined as the percentage of words that modify the correct head in the corpus. A natural analysis would be to look at these errors by part-of-speech tag. This analysis is given in Table 6.4, which is sorted by the total number of incorrect head attachment decisions for each POS tag. In other words, tags near the top of the table account for the largest amount of errors made by the parser.

Let's first look at unlabeled accuracy (column with UA header). The results here are not surprising at all. These show, for English, that most modification errors are for prepositions (IN) and coordinating conjunctions (CC and ,). These have repeatedly been shown to be the most difficult phenomena to parse in English [25]. Errors in prepositions are often caused by inherent ambiguities in attachment decisions. However, this is not usually the case for conjunctions, whose errors are often the result of locality restrictions on the feature representation. Note that nouns also account for a large number of absolute parsing mistakes. However, this is really due to their abundance in the corpus. In fact, unlabeled parsing accuracies for nouns are well above the average.

A common coordinating conjunction error occurs with coordination within a base-NP [1] as in, *That was modestly higher than the 8.8 % and 9.2 % levels in August and September of 1987*. Here the conjunction phrase *8.8 % and 9.2 %* modifies the noun *levels* (i.e., ((*8.8 %*) and (*9.2 %*)) *levels*). NP coordinating conjunctions in our English data set modify the final element of the conjunction, which in this case is the token *%* as head of the noun-phrase *9.2 %*. However, the parser makes a simple mistake and produces *9.2 % levels* as a noun phrase. As a result it incorrectly makes the conjunction a modifier of *levels*. This situation could benefit from some regular expression features that notice a repeat in pattern, i.e., *[num] %*, which might help with the recognition of the proper boundaries of the conjunction phrase.

[1]Exemplified by the NX tag in the Penn Treebank.

Another common error is sentence initial sentence initial CCs attaching to the wrong verb in multi-verb sentences. For example, *But a strong level of investor withdrawls is much more unlikely this time around, fund managers said*. Here, the parser incorrectly attaches the sentence initial conjunction to its closest verb *is*, instead of the main verb of the sentence *said*. This case also is an example of a genuine ambiguity, in which knowledge of the discourse is required to parse correctly. Part-of-speech errors also cause problems. For instance, in the sentence *Mr. Shidler 's company specializes in commercial real-estate investment and claims to have ...* . The word *claims* is mislabeled a noun due to the financial nature of the corpus. Here the conjunction should modify the first verb in the conjunction, *specializes*, but gets confused due to the fact that it is surrounded by two nouns and thus modifies the second of these nouns. Finally, another interesting example is *It is now the United Kingdom of Great Britain and Northern Ireland, comprising of Wales, Northern Ireland, Scotland, and ... oh yes, England, too*. There are actually two noun phrases here (*United Kingdom of Great Britain and Northern Ireland* and *Wales, Northern Ireland, Scotland, and ... oh yes, England, too*), but the parser mistakenly treats it as an entire phrase (i.e., *comprising of Wales* mistakenly modifies the first *Northern Ireland*)

One obvious solution to preposition and conjunction errors is to identify specific and more global features for these cases. For example, if we are considering a dependency in which a preposition will modify a noun, we can heuristically find the first verb to the left of the noun and the first noun phrase to the right of the noun to create features over the tuple (V,N,P,N) to help with attachment ambiguities between the verb and the noun. If the heuristic is reasonable, then we have simulated the classification environment of PP-attachment [27]. We can do a similar thing for conjunctions by using regular expressions to find all the base-nouns or verbs participating in the conjunction, as mentioned earlier. Since conjunctions typically modify the last or first argument, then such features might improve performance. We tried introducing these features, but they did not improve accuracy. Even breaking results down by part-of-speech tag showed that accuracies for prepositions and conjunctions were not significantly effected. This could be due to errors in the heuristics to extract those features.

The final major source of unlabeled accuracy error comes from adverbials (RB). This is a result of a number of things, including reduced relative clauses, inherent ambiguities in the sentence as well as inconsistencies between the head of adjective phrases resulting from the fixed head percolation rules. A frequent source of error also arose due to ADJPs in comparative constructions like *as high as*, where the first instance of *as* should modify the adjective *high* to create an ADJP *as high*. This was often parsed with either the first or last *as* being the head of an adverbial phrase.

In terms of labeling accuracy, which is the percentage of modifiers for which the incoming edge in the tree has the correct label (not necessarily the correct head), we see that the largest sources of errors are nouns (NN, NNS), verbs (VB, VBZ, VBN, VBG, VBP), adverbials (RB) and adjectives (JJ). This is not surprising for nouns and verbs since edges involving these words have the most ambiguity (e.g., the labels SBJ, OBJ, PRD, NP, SBAR, S, VP, DEP). On the other hand, part-of-speech types such as determiners, prepositions, conjunctions and punctuation can usually be labeled with a single rule (e.g., DT $\rightarrow$ DEP), thus have labeling accuracies greater than 95%. Verb subject errors are most common due to subjects following the verb, such as *Not included in the bid are Bonwit Teller or B. Altman & Co., L.J. Hooker's department store chains*. Here, the conjoined phrase was parsed correctly, however, it incorrectly modified the verb *are* as a NP-PRD.

For NP-OBJ a common error seems to be from nouns modifying verbs in what is defined in the Penn Treebank as the "Closely Related" relationship, for example, *he has had trouble finding the stocks he likes*. Here the noun *trouble* is considered closely related to the verb *had* and is annotated essentially as the fixed phrase *had trouble*. The parser mistakes this for a direct object. The opposite also happens, where the parser treats the noun as a closely related modifier, when it really should be the object. Some errors are also due to the heuristic nature of the inclusion of the NP-OBJ label in the extraction script, since OBJ is not a function tag originally annotated in the Penn Treebank. The OBJ label was included for any NP under a VP that lacks an adverbial function [106]. For example, in *Revenue gained 6 % to $ 2.55 billion from $ 2.4 billion*, the extraction script labeled the dependency from *6 %* to *gained* as an NP-OBJ since it is an NP within a VP. However, *6 %* is not a direct object, but simply a result of the gaining event and was labeled as an NP by the dependency parser.

For the label ADVP a large number of errors result from incorrect part-of-speech tagging. For instance, in ... *having much of an effect* ... the adjective *much* is mislabeled as an adverb. It is correctly parsed as the head of the phrase *much of an effect* (according to the head rules), but that phrase is incorrectly labeled an ADVP and not a NP-OBJ[2]. Other common errors are due to adverbs heading adjective phrases, such as *who are [ADJP [ADVP only casually] related [PP to the magazine]]*. Here *only* heads the adjective phrase (according to the Yamada and Matsumoto head rules), but since it is an adverb, the edge is labeled an ADVP. This error causes many of the ADJP phrase labeling problems as well.

The final column in Table 6.4 is labeled accuracy (LA), which combines both unlabeled accuracy and labeling accuracy. The primary property of interest in this column is to note that the differences in unlabeled and labeling accuracy often multiply. For instance, nouns and verbs have high unlabeled accuracy but low labeling accuracy, resulting in average labeled accuracy, and vice-versa for conjunctions and prepositions. The worst is adverbs, whose low unlabeled accuracy and low labeling accuracy multiply together to result in a labeled accuracy of 76%, well below the average.

[2]This seems like another case of an error in assigning nouns the OBJ label.

Part-of-speech Tag	words	correct head	incorrect head	UA	correct label	labeling accuracy	correct head and label	LA
IN	5934	5039	895	85%	5549	94%	4814	81%
,	3064	2431	633	79%	3062	100%	2431	79%
NN	7841	7246	595	92%	7227	92%	7012	89%
RB	1991	1661	330	83%	1673	84%	1517	76%
NNS	3561	3264	297	92%	3265	92%	3164	89%
NNP	5500	5215	285	95%	5206	95%	5070	92%
JJ	3663	3439	224	94%	3374	92%	3272	89%
CC	1369	1168	201	85%	1358	99%	1164	85%
CD	1943	1792	151	92%	1855	95%	1767	91%
DT	4834	4692	142	97%	4784	99%	4667	97%
VBG	856	722	134	84%	735	86%	680	79%
.	2363	2231	132	94%	2363	100%	2231	94%
VBD	1814	1695	119	93%	1717	95%	1685	93%
VBZ	1239	1126	113	91%	1139	92%	1110	90%
“	531	426	105	80%	531	100%	426	80%
VB	1549	1446	103	93%	1445	93%	1404	91%
VBN	1190	1092	98	92%	1074	90%	1036	87%
:	324	235	89	73%	324	100%	235	73%
TO	1240	1151	89	93%	1206	97%	1144	92%
VBP	811	727	84	90%	731	90%	710	88%
’	512	436	76	85%	511	100%	435	85%
MD	583	530	53	91%	546	94%	525	90%
WDT	276	234	42	85%	266	96%	231	84%
JJR	190	151	39	79%	150	79%	133	70%
WRB	132	96	36	73%	123	93%	93	70%
$	376	342	34	91%	342	91%	334	89%
POS	548	524	24	96%	538	98%	522	95%
RBR	107	85	22	79%	88	82%	78	73%
-RRB-	72	53	19	74%	72	100%	53	74%
PRP	1050	1032	18	98%	1033	98%	1025	98%
WP	112	95	17	85%	104	93%	92	82%
PRP$	511	494	17	97%	511	100%	494	97%
-LRB-	72	61	11	85%	72	100%	61	85%
JJS	128	118	10	92%	116	91%	112	88%
NNPS	118	112	6	95%	109	92%	108	92%
UH	10	5	5	50%	8	80%	4	40%
WP$	21	17	4	81%	21	100%	17	81%
PDT	21	17	4	81%	21	100%	17	81%
RBS	27	24	3	89%	26	96%	24	89%
LS	4	2	2	50%	2	50%	0	0%
FW	4	2	2	50%	1	25%	1	25%
EX	58	56	2	97%	58	100%	56	97%
RP	130	129	1	99%	105	81%	105	81%
#	5	4	1	80%	5	100%	4	80%

Table 6.4: Head modification accuracy by modifier part-of-speech tag. UA is unlabeled accuracy, labeling accuracy is the percentage of modifiers whose incoming edge has the correct label (though not necessarily the correct head), and LA is labeled accuracy.

Dependency Label	gold	correct	incorrect	system	recall(%)	precision(%)
DEP	25522	23278	2244	25767	91.21	90.34
NP	7449	6586	863	7434	88.41	88.59
PP	5429	4574	855	5582	84.25	81.94
SBAR	1757	1275	482	1638	72.57	77.84
S	2774	2375	399	2800	85.62	84.82
ADJP	769	456	313	592	59.3	77.03
NP-SBJ	4111	3809	302	4039	92.65	94.31
ADVP	1166	908	258	1199	77.87	75.73
NP-OBJ	2011	1775	236	2075	88.26	85.54
VP	2232	2057	175	2264	92.16	90.86
ROOT	2416	2288	128	2416	94.7	94.70
PRN	141	77	64	116	54.61	66.38
NP-PRD	346	284	62	351	82.08	80.91
PRT	159	106	53	130	66.67	81.54
NX	44	3	41	6	6.82	50.00
QP	187	154	33	190	82.35	81.05
UCP	30	3	27	7	10	42.86
FRAG	19	1	18	3	5.26	33.33
CONJP	21	4	17	14	19.05	28.57
NAC	30	16	14	17	53.33	94.12
SINV	11	2	9	3	18.18	66.67
WHNP	30	22	8	24	73.33	91.67
INTJ	10	4	6	8	40	50.00
X	5	1	4	2	20	50.00
LST	4	0	4	2	0	0.00
WHADVP	8	5	3	5	62.5	100.00
SBARQ	2	0	2	0	0	NaN
SQ	1	0	1	0	0	NaN

Table 6.5: Labeled precision/recall of dependency edges by edge label.

Dependency Errors by Label Type

Another method of analyzing dependency parsing performance is to measure labeled precision and recall, which is similar, but not analogous, to performance metrics for phrase-based parsing [25]. Labeled precision for an edge label is the percentage of edges we predicted with that label that were valid, and recall is the percentage of valid edges with that label that were predicted. For English, results are shown in Table 6.5.

Here the errors are also not surprising. For instance, the label SBAR has both a poor precision and recall due to its confusion versus both the labels S and VP, which occur more frequently. Similarly, the modifiers of verbs typically have a low precision and recall. In particular, the labels NP-OBJ, NP-PRD, NP and DEP are often confused since they are typically to the right of the verb, whereas NP-SBJ is almost exclusively to its left (i.e., the direction of attachment feature plus modifier POS tag would typically be enough to get

NP-SBJ edges correct).

Again we note that both adverbial and adjective phrasal labels have particularly bad precision and recall. Finally, we note that the label ROOT does not have a precision/recall of 1.0 due to unlabeled parsing errors (i.e., the wrong verb in a multi-clause sentence being attached to the root) and not because of labeling errors.

6.1.2 Feature Space Analysis

One powerful aspect of discriminative models is their ability to incorporate rich sets of highly dependent features. In previous sections the models we have described have clearly taken advantage of this property by incorporating millions of features over parsing decisions and the input in order to produce dependency scores. One very natural question to ask is, *how does each feature affect performance?*

For generative parsing models, Bikel [6] provided a detailed analysis of the contribution of each class of distribution to the generative power of the model. One conclusion drawn from Bikel's experiments was that bilexical distributions are very similar to back-off distributions that do not use bilexical information, which explained the long-known property that generative parsers have little performance degradation once bilexical information is removed. Ideally we would like to do a similar analysis of our dependency parsing models. However, the non-probabilistic nature of our models prevents that kind of comparison.

A simple method might be to look at the weight of each individual feature to determine its importance to parsing. But again there is a problem. Positive feature weights typically indicate good dependency decisions and negative weights bad decisions. Beyond that though, there is very little one can infer from the weight itself. This is due to the discriminative way in which the weights are learned. Consider features f_1 through f_m that occur frequently, always occur together and only occur for edges that represent valid dependencies. Since we train our models to set weights so that correct decisions are made with the smallest weights possible, it should be the case that the weight for each of these features is low, since all we require is the sum of their weights to be high. On the other hand consider feature f' that occurs once for a correct dependency. Furthermore assume all the other features that are on for this correct dependency usually occur only for incorrect dependencies. The model will naturally set the weight of feature f' high since it needs to overcome the negative influence of all those other features. However, a high weight for feature f' does not necessarily mean it is an important feature, it probably just means that that particular edge is an outlier in the training set. Conversely, the low weights of features $f_1 \ldots f_m$ does not indicate they are less important. In fact, for performance on the test set, it is almost certain that features $f_1 \ldots f_m$ will be of more use.

In this section we describe two experiments. The first is to identify classes of features, such as word or part-of-speech features, and evaluate their impact on overall accuracy by retraining the model without them and measuring the subsequent drop in performance. In the second experiment we compare two feature selection techniques, count cut-off and information gain, and evaluate the performance of each on varying sizes of feature sets.

Leave Out Feature Tests

The most common technique for determining the impact of each feature in a discriminative learning setting is the so called *leave out feature tests*. This method simply identifies common classes of features, retrains the models without these features and measures the drop in subsequent performance. The intuition is simple - the features that are most important to parsing will result in the largest drop in accuracy when they are left out. This analysis is imperfect. Often two feature classes will overlap significantly, thus leaving one class out may not have a huge impact on performance since another class can encode much of the information. We will try to identify such classes when they arise.

For these experiments, we broke our feature set Section 3.2 into the following classes:

	English First-Order		English Second-Order
	Unlabeled Accuracy		Unlabeled Accuracy
Full	90.7	Full	91.5
NoPrefix	90.7	NoPrefix	91.4
NoPOSContext	89.4	NoPOSContext	91.0
NoPOSBetween	90.2	NoPOSBetween	91.3
NoPOSContextBetween	86.0	NoPOSContextBetween	90.0
NoEdge	87.3	NoEdge	89.5
NoBiLex	90.6	NoBiLex	91.5
NoAttachmentOrDistance	88.1	NoAttachmentOrDistance	91.3

Table 6.6: English parsing accuracy results using various subsets of features.

- **Full:** The full set of features.
- **NoPrefix:** The full set, but without prefix features.
- **NoPOSContext:** The full set, but without POS context features, i.e., the features describing the POS tags of the words surrounding the head and modifier word of the dependency.
- **NoPOSBetween:** The full set, but without POS features for the words in-between the head and modifier.
- **NoPOSContextBetween:** The intersection of NoPOSContext and NoPOSBetween. POS context and in-between features overlap significantly. This set is meant to show the performance without either of them.
- **NoEdge:** The full set, but without any features over the head or modifier.
- **NoBiLex:** Identical to NoEdge, however we only omit those features that contain the word identities of *both* the head and modifier. This also includes bilexical prefix features.
- **NoAttachmentOrDistance:** The full set, but we never include any information about dependency attachment direction or the distance between the head and modifier.

Table 6.6 shows the results for each subset of features. We report results for both the first-order model and the second-order model for English.

For both systems the prefix features had very little effect. In fact, the reason the system even includes prefix features is to improve performance on Czech, which had a much larger vocabulary and rich morphology. The prefix feature was specifically designed as an ad-hoc method for extracting the lemma from the inflected form in Czech. The prefix feature for Czech resulted in around a half percent increase in performance absolute. Similarly, the bilexical features had virtually no effect on parsing performance. This is not surprising and confirms previously reported results from the lexicalized phrase-structure community [55].

For the first-order model, knowing the direction of attachment as well as the distance from the head to the modifier is crucial to performance. Without this knowledge performance drops from 90.7 to 88.1. This of course makes sense. Particular word classes (e.g. noun and prepositional modifiers) typically will only have a head that is nearby, whereas others (e.g., main verbs) can have heads at a much larger distance. Direction is important since some modifiers exist almost exclusively to the left or right of the head (e.g., modifiers of nouns are usually to the left, with the exceptions usually only for prepositions or relative clauses). Direction and distance become less important with the second-order model. This is not too surprising. Consider a preposition attaching to a noun that is far away on its left. Without distance we cannot rule out such a case because we cannot learn the regularity that prepositions typically only attach to nouns if they are directly to the right of them. However, if the noun and the preposition are far away, the noun is likely to have taken modifiers between them, which is encoded in the second-order model, which serves as a proxy in this case.

The most interesting case is what happens when part-of-speech context and in-between features are removed. For the first order model, removing either does not ruin the performance of the parser (90.7 to 89.4 and 90.2). However, once both context and in-between features are removed, parsing accuracy drops significantly to 86.0%. Note that even removing all the features of the head and modifier in the dependency does not even hurt performance as much (87.3%). This result is one of the most important in this work, so important that we emphasize it here:

Maximum spanning tree parsing provides efficient inference algorithms for *both the projective and non-projective case. Furthermore, the weaknesses of edge-based factorization can be overcome with discriminative learning that exploits rich feature sets describing properties of each dependency as well as their local context.*

The edge features and part-of-speech context/in-between features are all statistically dependent, but with our discriminative learning framework we can learn weights for each of these features in order to achieve state-of-the-art accuracies. Thus, an aggressive edge based factorization can be overcome with a rich feature representation of the input. As a result we achieve efficient non-projective parsing with high empirical accuracy by using discriminative learning with spanning tree inference algorithms.

The part-of-speech context and in-between features can be viewed as simulating higher order feature representations. First, we note the drop in performance of the second-order model when these features are removed (Figure 6.6). This drop is not nearly as significant as the drop in performance of the first-order model, i.e., there is only a 15% relative change in the second-order model versus a 34% change in the first-order model. The relative unimportance of these features for the second-order model shows that their information overlaps significantly with the second-order features.

Feature Selection

The number of features for the various models we have described is typically between five and fifteen million. Of these millions of features it seems likely that many are unimportant and that even some are misleading (e.g., infrequent features). For feature spaces of this size, people often employ some form of feature selection [50] as means to reduce model size or improve generalization performance.

In this section we consider two simple feature selection criteria for our first-order parsing models. The first is feature count cut-off. Here we simply just take those features that occur in the training set more than a predefined threshold. The second criteria is information gain. Consider a random variable $Y \in \{-1, 1\}$ that is 1 if a proposed dependency edge (i, j) is valid and -1 if it is not. Let $H(Y)$ represent the entropy of $P(Y)$. Now consider a particular feature f and the distribution $P(Y|f)$ with entropy $H(Y|f)$. The information gain of feature f is defined as,

$$IG(f) = H(Y) - H(Y|f)$$

This difference in entropy measures how much information about random variable Y, the feature f encodes. We would expect features with high information gain to encode a lot of information about which dependency edges are valid and which are not. Thus, it seems reasonable that these features should help in parsing dependencies.

Figure 6.1 shows dependency accuracy results for both methods using cut-off or information gain thresholds that result in feature set sizes of roughly 10,000, 100,000 and 500,000 features. This figure shows that with many features there is little difference between the two methods. However, when we look at smaller feature set sizes feature count cut-off seems to outperform information gain. At first glance this seems like a surprising result since count cut-off is one of the crudest methods for feature selection.

A closer look at the actual features returned by each method sheds some light on why count cut-off performs so well. Most of the features that count cut-off returns are based on part-of-speech information and not lexical information. As we showed in the previous section, these features tend to generalize best. Furthermore, since we are using frequency as a means for selection, these features are also more likely to occur in the test set as well. On the other hand, information gain in many cases returned features that are highly indicative when present, but occur less frequently. It appears that having a lot features on per edge is more important than having few informative features that occur infrequently in the test set.

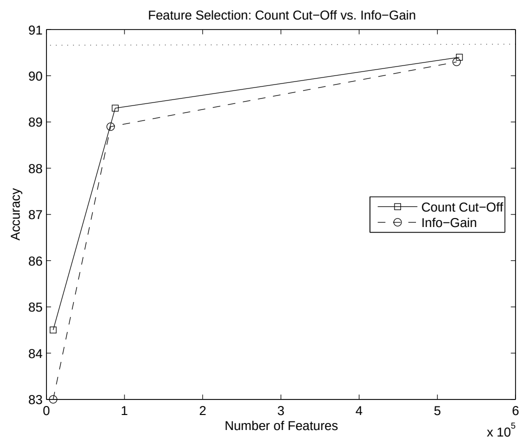


Figure 6.1: Feature selection tests comparing feature count cut-off to information gain. The top dashed line represents the parser when all of the roughly 7,000,000 English features are included.

We should note that other feature selection metrics were attempted including Chi-squared and Bi-normal separation [50]. However, none led to better results than count cut-off or information gain.

In terms of model selection, we can determine from Figure 6.1 that with less than $1/10^{th}$ of the feature set (from 7 million features to 550,000) performance is nearly as good as the full model (90.7% versus 90.2%) using feature count cut-off.

6.2 All Languages

In this section we will present a brief quantitative study of the errors produced by the parsers for languages other than English. The error analysis of this section is much more coarse grained than that for English. It is meant to provide an intuition for what aspects of parsing systems improve performance across languages, as well as what are the properties of a language that make it either difficult or easy to parse.

	Normal	Projective	No-Morph Features	Atomic Labeling	Projective No-Morph Atomic
Arabic	79.6/66.9	79.3/66.9	78.0/65.1	79.3/66.8	78.0/65.0
Bulgarian	92.0/87.6	92.0/87.6	91.9/87.2	92.0/87.3	91.9/86.5
Danish	90.6/84.8	89.8/84.1	90.1/84.0	90.6/84.3	89.0/82.6
Dutch	83.6/79.2	78.8/74.7	82.2/77.8	83.6/79.2	78.3/73.8
Japanese	92.8/90.7	92.9/90.7	92.7/90.6	92.8/90.4	92.6/90.3
Portuguese	91.4/86.8	91.7/87.0	90.5/85.7	91.4/86.5	90.0/84.7
Slovene	83.2/73.4	82.5/72.6	82.2/71.5	83.2/73.3	81.7/70.8
Spanish	86.1/82.3	86.1/82.3	85.1/80.9	86.1/82.0	85.1/80.7
Swedish	88.9/82.6	88.9/82.6	89.2/82.6	88.9/81.7	89.2/82.0
Turkish	74.7/63.2	74.7/63.2	72.8/60.6	74.7/63.4	72.8/60.6
Average	86.3/79.7	85.7/79.2	85.5/78.6	86.3/79.4	84.9/77.7

Table 6.7: Error analysis of parser components averaged over Arabic, Bulgarian, Danish, Dutch, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish. Normal: Reported result, Projective: Only allow projective edges, No-Morph Features: Only features over words and POS tags, Atomic Labeling: Do not use sequence labeling. Each cell contains the unlabeled and labeled accuracy values (UA/LA).

6.2.1 Quantitative Error Analysis

The models described in this work have several components, including the ability to produce non-projective edges, sequential assignment of edge labels instead of individual assignment, and a rich feature set that incorporates derived morphological properties when available. The benefit of each of these is shown in Table 6.7. These results report the labeled and unlabeled precision for the 10 languages with the smallest training sets. This allowed us to train new models quickly.

Table 6.7 shows that each component of our system does not change performance significantly (row labeled *Average*). However, if we only allow projective parses, do not use morphological features and label edges with a simple atomic classifier, the overall drop in performance becomes significant (first column versus last column). Allowing non-projective parses helped with freer word order languages like Dutch ($78.8\%/74.7\%$ to $83.6\%/79.2\%$, unlabeled/labeled accuracy). Including rich morphology features naturally helped with highly inflected languages, in particular Spanish, Arabic, Turkish, Slovene, Dutch and Portuguese. Derived morphological features improved accuracy in all these languages by 1-3% absolute. See Section 6.2.2 for more details.

Sequential classification of labels had very little effect on overall labeled accuracy (79.4% to 79.7%)[3]. The major contribution was in helping to distinguish subjects, objects and other dependents of main verbs, which is the most common labeling error. This is not surprising since these edge labels typically are the most correlated (i.e., if you already know which noun dependent is the subject, then it should be easy to find the object). For instance, sequential labeling improves the labeling of objects from $81.7\%/75.6\%$ to $84.2\%/81.3\%$ (labeled precision/recall) and the labeling of subjects from $86.8\%/88.2\%$ to $90.5\%/90.4\%$ for Swedish. Similar improvements are common across all languages, though not as dramatic. Even with this improvement, the labeling of verb dependents remains the highest source of error.

6.2.2 The Benefit of Morphological Features

The integration of morphology into statistical parsers has been an open issue in the parsing community. In particular, interest in parsing Semitic languages and other highly inflected languages has given rise to the question of what role morphological information will play in parsing. One key aspect of the discriminative parsing models described in this work is that they have a natural mechanism for incorporating such information – the feature set.

Our parsing models simply incorporated the cross-product of morphological information between a head and a modifier as features. This simple and somewhat naive method resulted in an overall improvement in labeled accuracy of 65.1% to 66.9% for Arabic, 77.8% to 79.2% for Dutch, 85.7% to 86.8% for Portuguese, 80.9% to 82.3% for Spanish, 71.5% to 73.4% for Slovene, and 60.6% to 63.2% for Turkish.

[3]This difference was much larger for experiments in which gold standard unlabeled dependencies are used.

Extending the feature set to improve parsing of highly inflected languages is clearly one of the most important areas of future research for discriminative parsing models such as those presented here.

6.2.3 Correlating Language and Data Properties with Accuracy

By building a single system to parse multiple languages, we are placed in a situation to gain insight into the underlying properties of a language that make it either difficult or easy to parse. This is because we can eliminate any variations in performance due to language specific parser optimizations. In this section we compare unlabeled parsing accuracies (including punctuation) relative to four properties,

1. *Average conditional entropy* of the head offset distribution. Consider the distribution $P(O|t,p)$, which is the probability that the head of a token is at some particular offset $O \in \mathbb{Z}$, for the given token and part-of-speech tag (t and p). For instance, $P(-2|\text{Inc.}, \text{N})$ is the probability that the head of the word *Inc.* is two words to its left. One would expect that languages with a non-uniform head offset distribution (e.g., always modify the word to the right) will be easier to parse.

2. *Average sentence length.* This simply measures throughout both the full training and testing sets what the average sentence length is for each language. As mentioned earlier, longer sentences typically cause more errors due to the higher presence of prepositions, conjunctions and multi-clause constructions.

3. *Percentage of unique tokens in training set.* On a normalized training set (i.e., identical length for each language), this measures the number of unique tokens in the data. Data sets with more unique tokens are typically drawn from more diverse sources and can lead to sparse parameter estimations. This also measures lexical sparsity.

4. *Percentage of new tokens in test set*. If the i.i.d. assumption is broken between the training and test set, then we would expect to see a high percentage of unseen tokens in the test set relative to the training set. This also measures lexical sparsity.

Figure 6.2 plots unlabeled dependency accuracy versus the four criteria outlined above. We normalized the training sets for each language so that they roughly contained the same number of tokens. This was done by randomly picking sentences from the data (without replacement) until the desired number of tokens was exceeded.

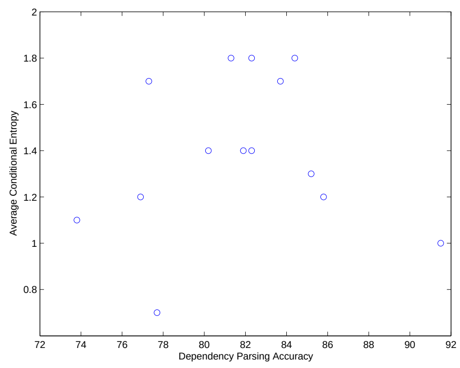
Figure 6.2a, Figure 6.2b, Figure 6.2c and Figure 6.2d show that there is little correlation between parsing accuracy any of the properties we have described. This is not too surprising, since one would expect multiple properties of a language and/or data set to contribute to parsing difficulty. For example, Arabic has a low average conditional entropy (1.1), but has on average the longest sentences (37). Turkish has very short sentences (8), but a high percentage of unique tokens due to the fact that it is highly inflected. Of all the properties, unique tokens in the training set has the highest correlation with parsing accuracy (around 0.56 statistical correlation).

The final plot, Figure 6.2e, is a linear combination of Figure 6.2a, b, c and d. This plot was generated by first normalizing all values using the log of each value relative to the average, i.e., $\log(\text{value}/\text{average value})$[4]. Then, the coefficients for each property in the linear combination were chosen using least-squares linear regression. Figure 6.2e clearly shows that a correlation with parsing accuracy is beginning to appear. This suggests that all of the properties together have a high impact on parsing accuracy, even though no one property is directly correlated. The statistical correlation of this linear combination and parsing accuracy is 0.85.

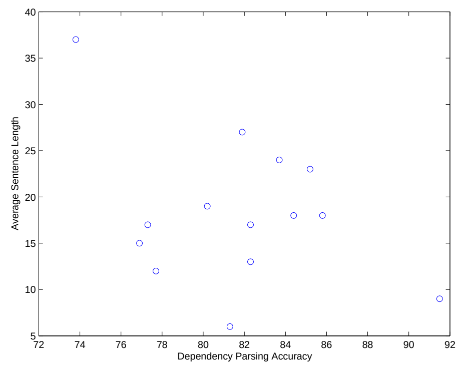
[4]For average conditional entropy we did not take the log since it is already on the log scale. Instead we just subtracted the average conditional entropy of all the languages together.

What conclusions can be drawn? The fact that the we can correlate parsing accuracy with properties of the data suggests that the parser is indeed language independent. However, some of these properties reflect aspects of the language itself, and not just one particular data set. The percentage of unique or unseen tokens will be large for highly inflected languages, which tend to be lexically sparse. We would expect, and in fact it is the case, that performance for these languages is lower than average. This suggests that the parser still needs to be improved to attain total language generality.

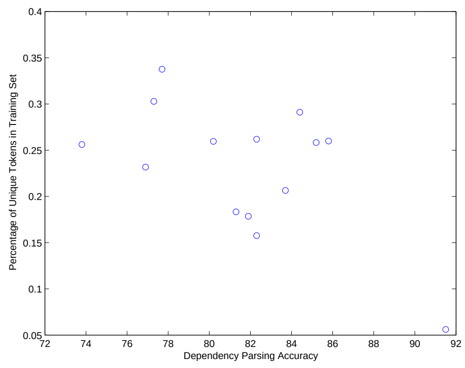
Of course, this analysis is simplistic and there are many more factors that can contribute. These include annotation design decisions, consistency of annotations, head extraction rules for converted treebanks, and the quality of the automatic part-of-speech and morphology tagger.



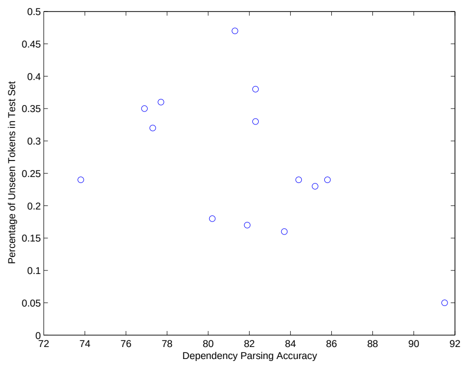
(a)



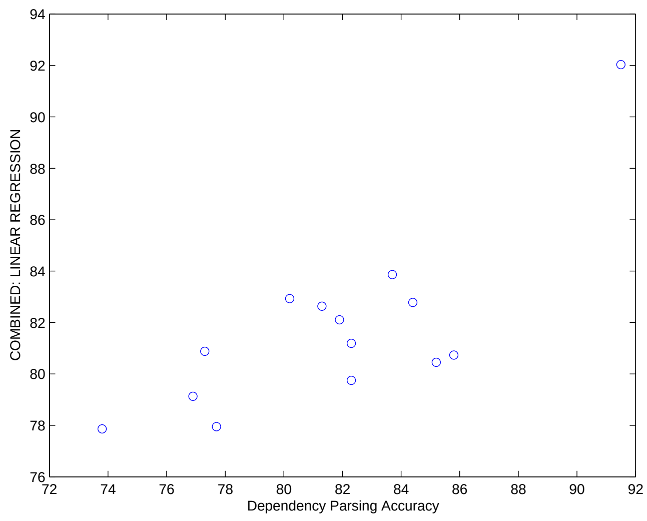
(b)



(c)



(d)



(e)

Figure 6.2: Plots of unlabeled dependency parsing accuracy versus properties of the data for each language. (a) Average conditional entropy of head offset distribution. (b) Average sentence length. (c) Percentage of unique tokens in normalized training set. (d) Percentage of unseen tokens in the test set relative to normalized training set. (e) Normalized least-squares linear regression of a, b, c, and d.

Chapter 7

Improved Parsing with Auxiliary Classifiers

In this chapter we demonstrate how it is possible to define new features indicating the output of auxiliary parsers trained on in and out of domain data sets. The first section deals with improving a WSJ parser by adding features on the output of diverse parsers trained on the same data. The second section shows how it is possible to adapt a WSJ parser to a new domain in which little training data is available.

7.1 Improving a WSJ Parser by Combining Parsers

Combining the outputs of several classifiers has been shown in the past to improve performance significantly for many tasks. One common method is to linearly interpolate the probability distributions of all the classifiers by either choosing or learning the weights on some held-out data set [69]. However, this approach requires that all models return a valid probability distribution and that the best parse can be efficiently found in the interpolated model, both of which are unlikely.

For parsing, a more common approach to combining the output of various parsers is to use voting. Henderson and Brill [64] described a constituent voting scheme for phrase-structure and Zeman [154] describes an edge voting scheme for dependency structures, both of which provide state-of-the-art results. However, voting requires at least three parsers and can involve complicated tie-breaking schemes when parsing decisions differ. Another problem of voting for parsing is that votes are typically accumulated over constituents/edges and often the winning constituents/edges do not constitute a valid tree. In fact, for dependency trees, the approach of Zeman is not guaranteed to obey the tree constraint, a property we wish to maintain. In both cases, heuristics are required to return a consistent parse.

Voting is also deficient since it assumes that we trust each of the parsers equally since each gets an equal vote. However, in practice, we typically prefer the output of some parsers over others. To address this problem Henderson and Brill also present a model that learns voting weights for each parser by using a naive Bayes classifier on a held-out set of data. This way we can add weight to the votes of parsers that are most trustworthy on this new data set. But again, the same problems with consistent constituents will arise.

Fortunately there is a simple mechanism for solving all these problems in the discriminative parsing framework. All one needs to do is define a class of features that indicates parsing decisions for each auxiliary parser and include these features into the discriminative model. For dependency parsing, this amounts to including features indicating whether an auxiliary parser believed a certain dependency or pair of dependencies actually exist in the tree. Once these features are added, we simply need to train the model on the new set of data. Specifically, we add two features. The first is a simple binary feature indicating for each edge (or pair of edges in the second-order case), whether or not the auxiliary parser believes this edge to be part of the correct tree. The second feature is identical to the first, except that we conjoin it with the part-of-speech of the head and modifier in the edge (or the head-sibling-modifier for the second-order case). We also negate these features and

	English	
	Accuracy	**Complete**
Collins	91.5	42.6
Charniak-proj	92.1	45.3
2^{nd}-order-proj^{+}	93.2	47.1

Table 7.1: Improved WSJ parsing performance using auxiliary parsing features.

include features indicating whether an auxiliary parser *did not* believe an edge to be part of the correct tree.

To train the new parser, we need the parsing decisions of the auxiliary classifiers on the training data. The problem with this is that these classifiers are also trained on exactly this data, which means they will be uncharacteristically accurate on this data set and will not represent the kinds of errors these parsers will make on unseen data. To alleviate this problem we divide the training in two and train two separate models for each auxiliary classifier. We then run each model on the half of the data it was not trained on to extract parsing decision features for the training set. This procedure is very similar to collecting training data for parse re-ranking.

Table 7.1 presents results for our second-order projective model on English that has been modified to include new features that incorporate parse decisions of the Collins and Charniak parser. We call the system 2^{nd}-order-proj^{+}. This system can then balance these features with its original feature set to produce an optimal parsing model. The resulting parser is far more accurate then the original second-order model as well as both the Collins and Charniak models. Thus, the core feature representation of a discriminative model allows us to naturally define new features on the output of other parsers to achieve the best reported parsing accuracies without resorting to complex voting schemes.

7.2 Adapting a WSJ Parser to New Domains

In general, large annotated data sets typically are not available to train state-of-the-art parsing models. One interesting problem is how to adapt resources to domains for which there is little to no training data available. In this section we investigate how to adapt a WSJ parser to parse biomedical research literature. Recently, Lease and Charniak [81] provided a set of techniques for modifying a trained WSJ phrase-structure parser when no biomedical training data is available. These techniques are easily be applied to the case of dependency parsing. In general, adapting out of domain or out of task annotated data to new problems is known as *transfer learning*.

In this section we focus on the different problem of parsing biomedical text when there is a small amount of training available. For these experiments we took 2,600 parsed sentences from the biomedical domain related to cancer [109]. We divided the data into 500 training, 100 development and 2000 testing sentences. We created five sets of training data with 100, 200, 300, 400, and 500 sentences respectively. The first experiment we ran was to simply see how well our trained WSJ parser performs on the biomedical text using our second-order projective model. It turns out that it performs reasonably well at 80.6% accuracy. The primary reason for this is that the numerous POS features from the WSJ parser are still beneficial for domains in which the lexicon is significantly different. The next experiments we ran were to train five parsers on the five sets of training data to measure parsing performance on small sets of data. Figure 7.1 plots accuracy as a function of training instances. *WSJ* is the performance of the basic WSJ parser and *Biomed* is the performance of the parsers trained on only the biomedical data. From the plot we can see that a parser trained on even a very small set of biomedical data (around 280 sentences) already outperforms a parser trained on 40,000 WSJ sentences.

We again take advantage of the fact that our discriminative dependency parser can de-

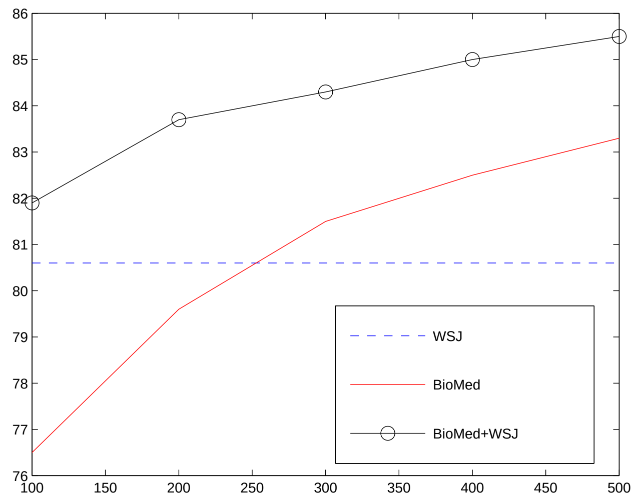


Figure 7.1: Adapting a WSJ parser to biomedical text. *WSJ:* performance of parser trained only on WSJ. *Biomed:* performance of parser trained on only biomedical data. *Biomed+WSJ:* parser trained on biomedical data with auxiliary WSJ parsing features. Figure plots dependency accuracy vs. number of biomedical training instances.

fine rich sets of features by creating a feature for the biomedical parser that indicate parsing decisions of the WSJ parser. This is completely analogous to the last section when we added decision features for auxiliary parsers. This time, our parser will be trained on the biomedical text and the auxiliary parser is the trained WSJ parser. The performance of this new model is plotted in Figure 7.1 as *Biomed+WSJ*. This simple trick leads to an absolute improvement of at least 2.2% in accuracy which represents greater than a 10% reduction in error.

The method of creating features over out of domain predictions is similar to the work of Florian et al. [49] for named-entity extraction. In that work, an extractor trained on one newswire corpus was adapted to another newswire corpus through features defined in a discriminative model. We have shown two things here. First, that this method can be applied to parsing and second, that this method still works when the two domains are very different in both content and writing style.

Another option would have been to combine the biomedical and the WSJ training sentences and train a model over this new set of data. However, this performed only marginally better than the WSJ parser since the number of WSJ training sentences overwhelmed the training procedure. We could run weighted training, in which biomedical training instances are weighted higher than WSJ instances. However, this would require the tuning of a weight parameter, which is unnecessary in the approach we have suggested. The power of using the feature space to incorporate out of domain parsing decisions is that it allows us to focus training *in domain* while incorporating information that can be obtained from a large out of domain annotated corpus.

We also looked at feature bagging [12, 132] approaches to parser adaptation. By training many WSJ parsers on various subsets of the feature space we can often create a diverse, but powerful, set of parsers, each providing a different view of the feature space. We then can add features over the outputs of all these parsers with the hopes that the biomedical parser will learn weights accordingly. Unfortunately, this technique did not lead to improved parser performance. This is not surprising, especially due to the nature of our parsing models. It has been shown that linear classifiers have low variance when trained on different subsets of data or features. When this is coupled with the fact that bagging tends to only work when the different classifiers have high variance, we should not expect these techniques to improve performance.

One new approach to adapting parsers across domains is that of McClosky et al. [89]. That work uses an out of domain parser to annotate a large set of in domain sentences. Using the noisy annotations for the in domain data a new parser is trained. If a lot of partially noisy data is equivalent to a modest amount of clean data, then an in domain parser trained this way could provide good performance. The methods of McClosky et al. are orthogonal to those presented here. We could easily train a parser in the same manner and then define features on the output of that parser when training a new parser on a small amount of annotated in domain data.

Chapter 8

Approximate Dependency Parsing Algorithms

Parsing dependencies has a relatively low parsing complexity of $O(n^3)$ with little to no grammar constant, which allows for searching the entire space of dependency trees during inference and learning. However, for other structures, such as lexicalized phrase-structure, non-projective second-order dependencies and dependency structures with multiple heads, the computational cost can become exponential or just to large to permit an exhaustive search.

Recently there has been much research on learning approximations with discriminative online learning algorithms. Collins and Roark [30] showed that an incremental parsing algorithm with an aggressive pruning criteria can still provide state-of-the-art performance for phrase-structure parsing when combined with perceptron learning. Daumé and Marcu [39] formalized online learning for approximate inference by defining a learning step in which updates are made based on errors in the approximation procedure. The primary reason that these methods work is that discriminative online learning allows the model to set its parameters relative to the inference algorithm. Thus, if the inference algorithm is

	Czech-A	
	Accuracy	**Complete**
Learning w/ Approximation	85.2	35.9
Learning w/o Approximation	69.3	10.7

Table 8.1: Approximate second-order non-projective parsing results for Czech displaying the importance of learning relative to the approximate parsing algorithm.

an approximation, then the model will set its parameters relative to the mistakes such an approximation might make. As Daumé and Marcu noted, the motivation and methods of this learning framework are highly related those proposed by the reinforcement learning community [133].

8.1 Second-order Non-projective Parsing

We have already shown that our approximate second-order non-projective parsing algorithm yields state-of-the-art results. However, we wish to also display the importance of learning relative to the approximation. Table 8.1 shows two systems. The first, *Learning w/ Approximation* is the original results for the second-order non-projective parsing model on Czech, in which the approximate post-process stage of the algorithm is incorporated directly into inference during learning. The second, *Learning w/o Approximation* is the results for a system in which the approximation is only used at test time. During learning, this system simply learn the projective model. What we can see is that it is crucial for approximations at test time to be incorporated during training. It is precisely learning relative to a specific inference algorithm that allows the online learner to adapt to pitfalls in the approximation procedure.

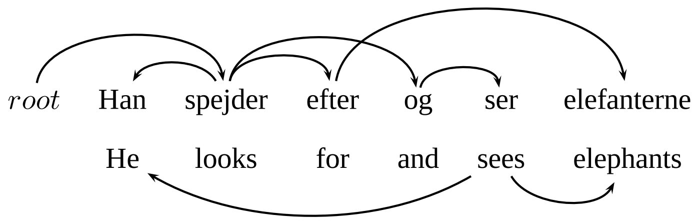


Figure 8.1: An example dependency tree from the Danish Dependency Treebank (from Kromann [79]).

8.2 Non-tree dependency structures: Danish

Kromann [78] argues for a dependency formalism called *Discontinuous Grammar* and annotated a large set of Danish sentences under this formalism (the Danish Treebank [79]). This formalism allows for a word to have multiple heads, e.g., in the case of verb conjunctions where the subject/object is an argument for multiple verbs or relative clauses in which words must satisfy dependencies within the clause and outside of it. An example is shown in Figure 8.1 for the sentence *He looks for and sees elephants*. Here, the pronoun *He* is the subject for both verbs in the sentence, and the noun *elephants* the corresponding object. In the Danish Treebank, roughly 5% of words have more than one head, which breaks the single head (or tree) constraint we have previously required on dependency structures. Kromann also allows for cyclic dependencies, but focus on sentences with acyclic representations. Though less common then trees, dependency representations containing multiple heads are well established in the literature (e.g., Hudson [67]). Unfortunately, the problem of finding the dependency structure with highest score in this setting is intractable [20]. This is true even for the first-order model and even if we can bound the number of heads to a constant k, where $k > 1$. In the first-order case, the problem is trivially polynomial when we remove the acyclicity constraint. The second-order problem is NP-hard for all cases.

To create an approximate parsing algorithm for dependency structures with multiple heads, we start with our approximate second-order non-projective algorithm outlined in

	Danish		
	Precision	**Recall**	**F-measure**
Projective	86.3	81.6	83.9
Non-projective	86.8	82.1	84.3
Non-projective w/ multiple heads	86.5	85.2	85.8

Table 8.2: Parsing results for Danish.

Figure 3.6. We use the non-projective algorithm since the Danish treebank contains a small number of non-projective arcs. We then modify lines 7-10 of this algorithm so that it looks for the change in head *or* the addition of an entirely new edge that causes the highest change in overall score and does not create a cycle. Like before, we make one change per iteration and that change will depend on the resulting score of the new tree. Using this simple new approximate parsing algorithm we can train a new parsing model with our online large-margin learning framework that will allow for the inclusion of multiple heads.

For our experiments we used the Danish Dependency Treebank v1.0. The treebank contains a small number of inter-sentence dependencies and we removed all sentences that contained such structures. The resulting data set contained 5384 sentences. We split the data into an 80/20 training/testing split by putting every fifth sentence into the training set. We used the identical second-order feature set that our English and Czech parser use, which resulted in 1,072,322 distinct features.

We compared three systems, the standard second-order projective and non-projective parsing models, as well as our modified second-order non-projective model that allows for the introduction of multiple heads. All systems use gold-standard part-of-speech since no trained tagger is readily available for Danish. Results are shown in Figure 8.2.

Some things we should note. First of all, the non-projective parser does slightly better then the projective parser since around 1% of the edges are non-projective. We can no longer use accuracy to measure performance since each word may modify an arbitrary number of heads. Instead we use edge precision and recall. Of course, this also means that using the Hamming loss during training no longer makes sense. A natural alternative is to use false positives plus false negatives over edge decisions, which relates the loss to our ultimate performance metric. As expected, for the basic projective and non-projective parsers, the recall is roughly 5% lower than the precision since these models can only pick up at most one head per word. For the parser that can introduce multiple heads, we see an increase in recall of over 3% absolute with only a slight drop in precision. These results are very promising and further show the robustness of discriminative online learning to approximate parsing algorithms.

8.3 Global Features

The trade-off between locality constraints (for tractable inference) and expressiveness is an interesting area of research. In this work we have shown how edge based locality constraints can provide tractable inference. When coupled with a rich feature set, these constraints also provide high empirical performance. However, it seems reasonable to conjecture that features over larger substructures of the dependency tree should improve performance (much in the same way that second-order features did). In this section we describe two kinds of non-local feature experiments. The first deals with global non-projective features. These features will represent global aspects of how a non-projective edge occurs in the sentence relative to other edges. The hope is that such features will prevent our search algorithms from arbitrarily inserting non-projectivity into dependency graphs. The second kind of feature represents the parsing decisions of other siblings (not just the nearest one) as well as the dependency properties of the head word (i.e., what is the head's head, a.k.a., grandparent).

Fortunately, there is a simple method for incorporating global features into the second-order non-projective approximate parsing algorithm given in Figure 3.6. The post processing stage considers all $O(n^2)$ possible edge transformations and recalculates the score of the entire tree for each one. It is trivial in this stage to add features indicating (after an edge transformation) the local and global properties of each edge.

Such global features have been encoded into a re-ranking module [28, 17]. However, we do not focus on re-ranking issues in this work and leave it as future research.

8.3.1 Global Non-projectivity Features

Both the first and second-order non-projective spanning tree algorithms presented in Chapter 3 make no restrictions on the types of non-projectivity that may occur. In particular, the features of the parsing models do not encode in anyway whether a particular edge, or sets of edges are non-projective. In practice this does not seem to adversely effect empirical performance, even though for many languages, non-projective edges typically only occur in restricted situations. It seems reasonable to assume that if we can somehow encode this information into the parsing models, then we might reduce the amount of erroneous non-projectivity in the returned parses. To do this, we added the following binary edge features,

1. Is the edge non-projective?
2. Is the edge non-projective & what are the POS tags of the head and modifier?
3. Is the edge non-projective & what are the POS tags between the head and modifier?

These features are distinct from those previously defined in other models. This is because they require the knowledge of an arbitrary number of edges in the tree to determine if a single edge is non-projective. All of these features were conjoined with direction of attachment and distance between the head and modifier. Note that these features cannot be incorporated into the first-order non-projective parsing model, since these properties rely on knowledge of the entire dependency graph.

	Czech	
	Accuracy	**Complete**
2^{nd}-order-non-proj	85.2	35.9
2^{nd}-order-non-proj approx w/ global features	85.2	35.6

Table 8.3: Benefit of additional global non-projective features.

Results are shown in Table 8.3 for the Czech data. Unfortunately these features made little difference to overall parsing performance. This can be explained by looking at the precision and recall of the non-projective edges in the data set. The parser with the global non-projective features had a higher precision but lower recall since it tended to restrict some non-projective edges from being inserted. This result in many ways justifies the data-driven parsing algorithms without the use of underlying grammars specifying when certain constructions may occur.

8.3.2 Grandparent and Other Sibling Features

In the second-order parsing model we extended the feature representation to include features over pairs of adjacent edges in the tree. In this section, we extend this further and allow features over an pair of edges that modify the same head. We add identical second-order features, except that we also indicate whether the edges are adjacent or not. In addition to this, we also extend the feature representation to incorporate dependency information relating to the head word. That is, we add a features over the identity of the head's head in the dependency graph (also known as the grandparent). This information is designed to disambiguate cases like prepositional attachment. If we consider the sentence *I saw the scientist with the telescope*, this new feature will tell us that the prepositional phrase is much more likely to attach to the noun since the dependency graph path feature *scientist* $\rightarrow$ *with* $\rightarrow$ *telescope* should have a high weight due to the high correlation between *scientist* and *telescope*. In contrast, for the sentence, *I saw the planet with the telescope*, the path feature

	English	
	Accuracy	**Complete**
2^{nd}-order-proj	91.5	42.1
2^{nd}-order-non-proj approx w/ global features	91.4	40.8

Table 8.4: Benefit of additional global sibling and grandparent features.

planet → *with* → *telescope* should have a lower weight since planets rarely are modified by telescopes, but are often seen through them (i.e., a higher weight should be associated with *saw* → *with* → *telescope*).

Results are shown in Table 8.4 for the English data. We added features over the part-of-speech tags of the grandparent, parent and child, the lexical identities of the grandparent and child and conjoin these with direction and distance between grandparent and child. Though we are parsing English, the system we report did not attempt to prevent non-projective edges from occurring. We could have simply added a constraint forcing the parser to only consider edge changes that did not break a projectivity constraint, but we found that this led to slightly worse performance. Unfortunately adding these features did not improve performance over the second-order model. In fact, performance did drop slightly. There may be many reasons for this. First, this information may not be important to improving dependency parsing accuracy. Second, these additional features might be causing the parsing models to overfit. And third, any added benefit from these features may might be washed out by the approximate inference. Previous results presented in this chapter provide evidence against this final reason.

8.3.3 Global Feature Summary

In this section we described some preliminary experiments to include global features of edges including additional sibling and grandparent information as well as global aspects of non-projective edges. Unfortunately these features did not improve performance. This seems to contradict much work on parse re-ranking [28, 17] which suggests global features are important to improving performance of the parsers. However, it can be argued that the real benefit of parse re-ranking is the use of discriminative learning optimizing an objective directly related to prediction performance, which the parsers in this work already benefit from.

One area of future work worth pursuing is to create an approximate greedy parser trained with the learning algorithms of Daumé and Marcu [39]. This algorithms provide a principled approach to incorporating global features into greedy approximate search algorithms and has been shown to be empirically justifiable.

8.4 Summary of Chapter

In this section we described some initial experiments suggesting that discriminative online learning techniques are robust to inference approximations. This is an example of a more general problem - learning and inference algorithms for intractable NLP problems. As the language processing community moves to induce more complex structures and even joint representations of various linguistic phenomenon, it will become increasingly important to develop learning and inference algorithms when traditional dynamic-programming algorithms like Viterbi, CKY, forward-backward or inside-outside fail.

Chapter 9

Application to Sentence Compression

In this chapter we apply our dependency parser to the problem of sentence compression to show its applicability in an important subcomponent of summarization systems. Summarization systems are evaluated on the amount of relevant information retained, the grammaticality of the summary and the compression rate. Thus, returning highly compressed, yet informative, sentences allows summarization systems to return larger sets of sentences and increase the overall amount of information extracted.

We focus on the particular instantiation of sentence compression when the goal is to produce the compressed version solely by removing words or phrases from the original, which is the most common setting in the literature [76, 114, 146]. In this framework, the goal is to find the shortest substring of the original sentence that conveys the most important aspects of the meaning. As is the case throughout this work, we use supervised learning and assume as input a training set $\mathcal{T}=(\boldsymbol{x}_t, \boldsymbol{y}_t)_{t=1}^{|\mathcal{T}|}$ of original sentences $\boldsymbol{x}_t$ and their compressions $\boldsymbol{y}_t$. We use the Ziff-Davis corpus, which is a set of 1087 pairs of sentence/compression pairs. Furthermore, we use the same 32 testing examples from Knight and Marcu [76] and the rest for training, except that we hold out 20 sentences for the purpose of development. A handful of sentences occur twice but with different compressions.

The Reverse Engineer Tool is priced from $8,000 for a single user to $90,000 for a multiuser project site .

The Reverse Engineer Tool is available now and is priced on a site-licensing basis , ranging from $8,000 for a single user to $90,000 for a multiuser project site .

Design recovery tools read existing code and translate it into definitions and structured diagrams .

Essentially , design recovery tools read existing code and translate it into the language in which CASE is conversant – definitions and structured diagrams .

Figure 9.1: Two examples of compressed sentences from the Ziff-Davis corpus. The compressed version and the original sentence are given.

We randomly select a single compression for each unique sentence in order to create an unambiguous training set. Examples from this data set are given in Figure 9.1.

Formally, sentence compression aims to shorten a sentence $\boldsymbol{x} = x_1 \ldots x_n$ into a substring $\boldsymbol{y} = y_1 \ldots y_m$, where $y_i \in \{x_1, \ldots, x_n\}$. We define the function $I(i) \in \{1, \ldots, n\}$ that maps word y_i in the compression to the index of the word in the original sentence. Finally we include the constraint $I(i) < I(i+1)$, which forces each word in $\boldsymbol{x}$ to occur at most once in the compression $\boldsymbol{y}$ and in the correct relative order. Compressions are evaluated on three criteria,

1. Grammaticality: Compressed sentences should be grammatical.

2. Importance: How much of the important information is retained from the original.

3. Compression rate: How much compression took place. A compression rate of 65% means the compressed sentence is 65% the length of the original.

Typically grammaticality and importance are traded off with compression rate. The longer our compressions, the less likely we are to remove important words or phrases crucial to maintaining grammaticality and the intended meaning.

This chapter is organized as follows: Section 9.1 discusses previous approaches to sentence compression. In particular, we discuss the advantages and disadvantages of the models of Knight and Marcu [76]. In Section 9.2 we present a discriminative large-margin model for sentence compression, including an efficient decoding algorithm for searching the space of compressions. Most importantly, we show how to extract a rich feature set that includes surface-level bigram features of the compressed sentence, dropped words and phrases from the original sentence, and features over dependency structures produced by the English parser described in this work. We argue that this rich feature set allows the model to accurately learn which words and phrases should be dropped and which should remain in the compression. Section 9.3 presents an experimental evaluation of our model compared to a model of Knight and Marcu. Furthermore, we also empirically display the importance of the dependency features in producing good compressions.

9.1 Previous Work

Knight and Marcu [76] first tackled this problem by presenting a generative noisy-channel model and a discriminative tree-to-tree decision tree model. The noisy-channel model defines the problem as finding the compressed sentence with maximum conditional probability

$$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y})$$

$P(\boldsymbol{y})$ is the source model, which is a PCFG plus bigram language model. $P(\boldsymbol{x}|\boldsymbol{y})$ is the channel model, the probability that the long sentence is an expansion of the compressed sentence. To calculate the channel model, both the original and compressed versions of every sentence in the training set are assigned a phrase-structure tree. Given a tree for a long sentence $\boldsymbol{x}$ and compressed sentence $\boldsymbol{y}$, the channel probability is the product of the probability for each transformation required if the tree for $\boldsymbol{y}$ is to expand to the tree for $\boldsymbol{x}$.

The tree-to-tree decision tree model looks to rewrite the tree for $\boldsymbol{x}$ into a tree for $\boldsymbol{y}$. The model uses a shift-reduce-drop parsing algorithm that starts with the sequence of words in $\boldsymbol{x}$ and the corresponding tree. The algorithm then either shifts (considers new words and subtrees for $\boldsymbol{x}$), reduces (combines subtrees from $\boldsymbol{x}$ into possibly new tree constructions) or drops (drops words and subtrees from $\boldsymbol{x}$) on each step of the algorithm. A decision tree model is trained on a set of indicative features for each type of action in the parser. These models are then combined in a greedy global search algorithm to find a single compression.

Though both models of Knight and Marcu perform quite well, they do have their shortcomings. The noisy-channel model uses a source model that is trained on uncompressed sentences, even though the source model is meant to represent the probability of compressed sentences. The channel model requires aligned parse trees for both compressed and uncompressed sentences in the training set in order to calculate probability estimates. These parses are provided from a parsing model trained on out of domain data (the WSJ), which can result in parse trees with many mistakes for both the original and compressed versions. This makes alignment difficult and the channel probability estimates unreliable as a result. On the other hand, the decision tree model does not rely on the trees to align and instead simply learns a tree-to-tree transformation model to compress sentences. The primary problem with this model is that most of the model features encode properties related to including or dropping constituents from the tree with no encoding of bigram or trigram surface features to promote grammaticality. As a result, the model will sometimes return very short and ungrammatical compressions.

Both models rely heavily on the output of a noisy parser to calculate probability estimates for the compression. We argue in the next section that ideally, parse trees should be treated solely as a source of evidence when making compression decisions to be balanced with other evidence such as that provided by the words themselves.

Recently Turner and Charniak [146] presented supervised and semi-supervised versions of the Knight and Marcu noisy-channel model. The resulting systems typically return informative and grammatical sentences, however, they do so at the cost of compression rate. Riezler et al. [114] present a discriminative sentence compressor over the output of an LFG parser that is a packed representation of possible compressions. This model is highly related to the system we present here. However, unlike Riezler et al., we do not let the output of an out of domain trained parser to guide our search. Instead, we will simply search the entire space of compressions and use syntactic information as one form of evidence to discriminate between good and bad compressions.

9.2 Sentence Compression Model

For the rest of this section we use $\boldsymbol{x} = x_1 \ldots x_n$ to indicate an uncompressed sentence and $\boldsymbol{y} = y_1 \ldots y_m$ a compressed version of $\boldsymbol{x}$, i.e., each y_j indicates the position in $\boldsymbol{x}$ of the j^{th} word in the compression. We always pad the sentence with dummy start and end words, $x_1 =$ -START- and $x_n =$ -END-, which are always included in the compressed version (i.e. $y_1 = x_1$ and $y_m = x_n$).

In this section we described a discriminative online learning approach to sentence compression, the core of which is a decoding algorithm that searches the entire space of compressions. Let the score of a compression $\boldsymbol{y}$ for a sentence $\boldsymbol{x}$ as

$$s(\boldsymbol{x}, \boldsymbol{y})$$

In particular, we are going to factor this score using a first-order Markov assumption on the words in the *compressed* sentence

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=2}^{|\boldsymbol{y}|} s(\boldsymbol{x}, I(j-1), I(j))$$

Finally, we define the score function to be the dot product between a high dimensional feature representation and a corresponding weight vector

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=2}^{|\boldsymbol{y}|} \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, I(j-1), I(j))$$

Note that this factorization will allow us to define features over two adjacent words in the compression as well as the words in-between that were dropped from the original sentence to create the compression. We will show in Section 9.2.2 how this factorization also allows us to include features on dropped phrases and subtrees from both a dependency and/or a phrase-structure parse of the original sentence. Note that these features are meant to capture the same information in both the source and channel models of Knight and Marcu [76]. However, here they are merely treated as evidence for the discriminative learner, which will set the weight of each feature relative to the other (possibly overlapping) features to optimize the models accuracy on the observed data.

9.2.1 Decoding

We define a dynamic programming table $C[i]$ which represents the highest score for any compression that ends at word x_i for sentence $\boldsymbol{x}$. We define a recurrence as follows

$$C[1] = 0.0$$
$$C[i] = \max_{j<i} C[j] + s(\boldsymbol{x}, j, i) \text{ for } i > 1$$

It is easy to show that $C[n]$ represents the score of the best compression for sentence $\boldsymbol{x}$ (whose length is n) under the first-order score factorization we made. We can show this by induction. If we assume that $C[j]$ is the highest scoring compression that ends at word x_j, for all $j < i$, then $C[i]$ must also be the highest scoring compression ending at word x_i since it represents the max combination over all high scoring shorter compressions plus the score of extending the compression to the current word. Thus, since x_n is by definition in every compressed version of $\boldsymbol{x}$ (see above), then it must be the case that $C[n]$ stores the score of the best compression. This table can be filled in $O(n^2)$.

This algorithm is really an extension of Viterbi to the case when scores factor over dynamic substrings of the text [118, 90]. As such, we can use back-pointers to reconstruct the highest scoring compression as well as standard k-best decoding algorithms.

This decoding algorithm is dynamic with respect to compression rate. That is, the algorithm will return the highest scoring compression regardless of length. This may seem problematic since longer compressions might contribute more to the score (since they contain more bigrams) and thus be preferred. However, in Section 3.2 we define a rich feature set, including features on words dropped from the compression that will help disfavor compressions that drop very few words since this is rarely seen in the training data. In fact, it turns out that our learned compressions have a compression rate very similar to the gold standard.

That said, there are some instances when a static compression rate is preferred. A user may specifically want a 25% compression rate for all sentences. This is not a problem for our decoding algorithm. We simply augment the dynamic programming table and calculate $C[i][r]$, which is the score of the best compression of length r that ends at word x_i. This table can be filled in as follows

$$\begin{aligned}
&C[1][1] = 0.0 \\
&C[1][r] = -\infty \text{ for } r > 1 \\
&C[i][r] = \max_{j<i} C[j][r-1] + s(\boldsymbol{x}, j, i) \text{ for } i > 1
\end{aligned}$$

Thus, if we require a specific compression rate, we simple determine the number of words r that satisfy this rate and calculate $C[n][r]$. The new complexity is $O(n^2 r)$.

9.2.2 Features

So far we have defined the score of a compression as well as a decoding algorithm that searches the entire space of compressions to find the one with highest score. This all relies on a score factorization over adjacent words in the compression, $s(\boldsymbol{x}, I(j-1), I(j)) = \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, I(j-1), I(j))$. In Section 2 we describe an online large-margin method for learning $\mathbf{w}$. Here we present the feature representation $\mathbf{f}(\boldsymbol{x}, I(j-1), I(j))$ for a pair of adjacent words in the compression. These features were selected on a development data set.

Word/POS Features

The first set of features are over adjacent words y_{j-1} and y_j in the compression. These include the part-of-speech (POS) bigrams for the pair, the POS of each word individually, and the POS context (bigram and trigram) of the most recent word being added to the compression, y_j. These features are meant to indicate likely words to include in the compression as well as some level of grammaticality, e.g., the adjacent POS features "JJ&VB" would get a low weight since we rarely see an adjective followed by a verb. We also add a feature indicating if y_{j-1} and y_j were actually adjacent in the original sentence or not and we conjoin this feature with the above POS features. We have not included any bi-lexical features because experiments on the development data showed that lexical information was too sparse and led to overfitting. Instead we rely on the accuracy of POS tags to provide enough evidence.

Next we added features over every dropped word in the original sentence between y_{j-1} and y_j, if there were any. These include the POS of each dropped word, the POS of the dropped words conjoined with the POS of y_{j-1} and y_j. If the dropped word is a verb, we add a feature indicating the actual verb (this is for common verbs like "is", which are typically in compressions). Finally we add the POS context (bigram and trigram) of

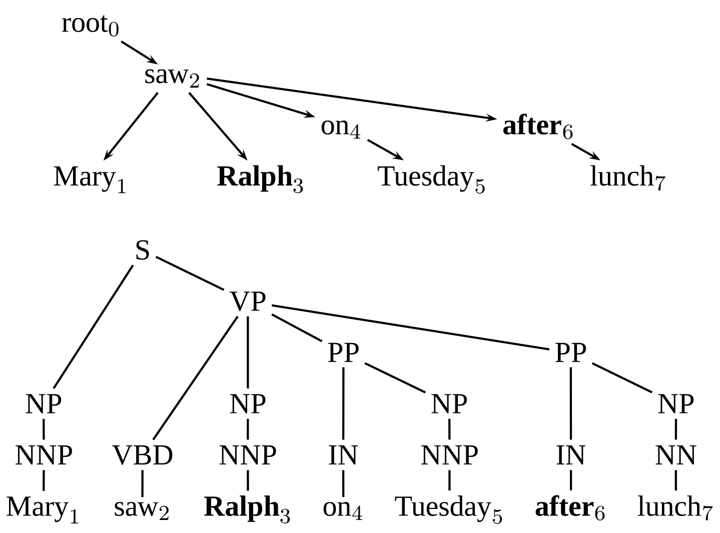


Figure 9.2: An example dependency tree from the dependency parser and phrase structure tree from the Charniak parser [16]. In this example we want to add features from the trees for the case when *Ralph* and *after* become adjacent in the compression, i.e., we are dropping the phrase *on Tuesday*.

each dropped word. These features represent common characteristics of words that can or should be dropped from the original sentence in the compressed version (e.g. adjectives and adverbs). We also add a feature indicating whether the dropped word is a negation (e.g., not, never, etc.).

We also have a set of features to represent brackets in the text, which are common in the data set. The first measures if all the dropped words between y_{j-1} and y_j have a mismatched or inconsistent bracketing. The second measures if the left and right-most dropped words are themselves both brackets. These features come in handy for examples like, *The Associated Press (AP) reported the story*, where the compressed version is *The Associated Press reported the story*. Information within brackets is often redundant.

Syntactic Features

The previous set of features are meant to encode common POS contexts that are commonly retained or dropped from the original sentence during compression. However, they do so without a larger picture of the function of each word in the sentence. For instance, dropping verbs is not that uncommon - a relative clause for instance may be dropped during compression. However, dropping the main verb in the sentence is uncommon, since that verb and its arguments typically encode most of the information being conveyed.

An obvious solution to this problem is to include features over a syntactic analysis of the sentence. To do this we parse every sentence twice, once with the dependency parser described in this work and once with the Charniak phrase-structure parser [16]. These parsers have been trained out of domain on the Penn WSJ Treebank and as a result contain noise. However, we are merely going to use them as additional sources of features. We call this *soft syntactic evidence* since the syntactic trees are not used as a strict gold-standard in our model but just as more evidence for or against particular compressions. The learning algorithm will set the feature weights accordingly depending on each features discriminative power. It is not novel to use soft syntactic features in this way, as it has been done for many problems in language processing. However, we stress this aspect of our model due to the history of compression systems using syntax to provide hard structural constraints on the output. In Section 9.3 we empirically display the importance of the dependency features derived from our parser.

Let us consider the sentence $\boldsymbol{x}$ = *Mary saw Ralph on Tuesday after lunch*, with corresponding parses given in Figure 9.2. In particular, let us consider the feature representation $\mathbf{f}(\boldsymbol{x},3,6)$, i.e., the feature representation of making *Ralph* and *after* adjacent in the compression and dropping the prepositional phrase *on Tuesday*. The first set of features we consider are over dependency trees. For every dropped word we add a feature indicating the POS of the word's head in the tree. For example, if the dropped word's head is *root*, then it is the main verb of the sentence and typically should not be dropped. We also add a conjunction feature of the POS tag of the word being dropped and the POS of its head as well as a feature indicating for each word being dropped whether it is a leaf node in the tree. We also add the same features for the two adjacent words, but indicating that they are part of the compression.

For the phrase-structure features, we find every node in the tree that spans a piece of dropped text and is not a descendant in the tree of another spanning node, in the example this is the PP governing *on Tuesday*. We then add features indicating the context from which this node was dropped. For example we add a feature specifying that a PP was dropped which was the modifier of a VP. We also add a feature indicating that a PP was dropped which was the left sibling of another PP, etc. Ideally, for each production in the tree we would like to add a feature indicating every node that was dropped, e.g. "VP$\rightarrow$VBD NP PP PP $\Rightarrow$ VP$\rightarrow$VBD NP PP". However, we cannot necessarily calculate this feature since the extent of the production might be well beyond the local context of first-order feature factorization. Furthermore, since the training set is so small, these features are likely to be observed too few times.

Feature Set Summary

In this section we have described a rich feature set over adjacent words in the compressed sentence, dropped words and phrases from the original sentence, and properties of syntactic trees of the original sentence. Note that these features in many ways mimic the information already present in the noisy-channel and decision-tree models of Knight and Marcu [76]. Our bigram features encode properties that indicate both good and bad words to be adjacent in the compressed sentence. This is similar in purpose to the source model from the noisy-channel system. However, in that system, the source model is trained on uncompressed sentences and thus is not as representative of likely bigram features for compressed sentences, which is really what we desire.

Our feature set also encodes dropped words and phrases through the properties of the words themselves and through properties of their syntactic relation to the rest of the sentence in a parse tree. These features represent likely phrases to be dropped in the compression and are thus similar in nature to the channel model in the noisy-channel system as well as the features in the tree-to-tree decision tree system. However, we use these syntactic constraints as *soft evidence* in our model. That is, they represent just another layer of evidence to be considered during training when setting parameters. Thus, if the parses have too much noise, the learning algorithm can lower the weight of the parse features since they are unlikely to be useful discriminators on the training data. This differs from the models of Knight and Marcu [76] and Riezler et al. [114], which treat the noisy parses as gold-standard when calculating probability estimates or when searching the space of possible compressions.

An important distinction we should make is the notion of *supported* versus *unsupported* features [120]. Supported features are those that are on for the gold standard compressions in the training. For instance, the bigram feature "NN&VB" will be supported since there is most likely a compression that contains a adjacent noun and verb. However, the feature "JJ&VB" will not be supported since an adjacent adjective and verb most likely will not be observed in any valid compression. Our model includes all features, including those that are unsupported. The advantage of this is that the model can learn negative weights for features that are indicative of bad compressions. This is not difficult to do since most features are POS based and the feature set size even with all these features is only 78,923.

9.2.3 Learning

Having defined a feature encoding and decoding algorithm, the last step is to learn the feature weights **w**. We do this using the MIRA as described in Chapter 2. The k-best compressions can easily be calculated by extending the decoding algorithm with standard Viterbi k-best techniques. On the development data, we found that $k = 10$ provided the best performance, though varying k did not have a major impact overall. Furthermore we found

	Compression Rate	Grammaticality	Importance
Human	53.3%	4.96 ± 0.2	3.91 ± 1.0
Decision-Tree (K&M2000)	57.2%	4.30 ± 1.4	3.60 ± 1.3
This work	58.1%	4.61 ± 0.8	4.03 ± 1.0

Table 9.1: Sentence compression results.

that after only 3-5 training epochs performance on the development data was maximized.

9.3 Experiments

We use the same experimental methodology as Knight and Marcu [76]. We provide every compression to four judges and ask them to evaluate each one for grammaticality and importance on a scale from 1 to 5. For each of the 32 sentences in our test set we ask the judges to evaluate three systems: human annotated, the decision tree model of Knight and Marcu and our system. The judges were told all three compressions were automatically generated and the order in which they were presented was randomly chosen for each sentence. We compared our system to the decision tree model of Knight and Marcu instead of the noisy-channel model since both performed nearly as well in their evaluation, and the compression rate of the decision tree model is nearer to our system (around 57-58%). The noisy-channel model typically returned longer compressions.

Results are shown in Table 9.1. We present the average score over all judges as well as the standard deviation. The evaluation for the decision tree system of Knight and Marcu is strikingly similar to the original evaluation in their work. This provides strong evidence that the evaluation criteria in both cases were very similar.

Table 9.1 shows that all models had similar compressions rates, with humans preferring to compress a little more aggressively. Not surprisingly, the human compressions are practically all grammatical. A quick scan of the evaluations shows that the few ungrammatical human compressions were for a sentence that was not grammatical in the first place. Of greater interest is that the compressions of our system are typically more grammatical than the decision tree model of Knight and Marcu.

When looking at importance, we see that our system actually does the best – even better than humans. The most likely reason for this is that our model returns longer sentences and is thus less likely to prune away important information. For example, consider the sentence

The chemical etching process used for glare protection is effective and will help if your office has the fluorescent-light overkill that's typical in offices

The human compression was *Glare protection is effective*, whereas our model compressed the sentence to *The chemical etching process used for glare protection is effective*.

A primary reason that our model does better than the decision tree model of Knight and Marcu is that on a handful of sentences, the decision tree compressions were a single word or noun phrase. For such sentences, the evaluators typically rated the compression a 1 for both grammaticality and importance. In contrast, our model never failed in such drastic ways and always output something reasonable. This is quantified in the standard deviation of the two systems.

Though these results are promising, more large scale experiments are required to really ascertain the significance of the performance increase. Ideally we could sample multiple training/testing splits and use all sentences in the data set to evaluate the systems. However, since these systems require human evaluation we did not have the time or the resources to conduct these experiments.

9.3.1 Some Examples

Here we aim to give the reader a flavor of some common outputs from the different models. Three examples are given in Table 9.3.1. The first shows two properties. First of all, the decision tree model completely breaks and just returns a single noun-phrase. Our system

Full Sentence	The first new product , ATF Protype , is a line of digital postscript typefaces that will be sold in packages of up to six fonts .
Human	ATF Protype is a line of digital postscript typefaces that will be sold in packages of up to six fonts .
Decision Tree	The first new product .
This work	ATF Protype is a line of digital postscript typefaces will be sold in packages of up to six fonts .
Full Sentence	Finally , another advantage of broadband is distance .
Human	Another advantage is distance .
Decision Tree	Another advantage of broadband is distance .
This work	Another advantage is distance .
Full Sentence	The source code , which is available for C , Fortran , ADA and VHDL , can be compiled and executed on the same system or ported to other target platforms .
Human	The source code is available for C , Fortran , ADA and VHDL .
Decision Tree	The source code is available for C .
This work	The source code can be compiled and executed on the same system or ported to other target platforms .

Table 9.2: Example compressions for the evaluation data.

performs fairly well, although it leaves out the complementizer of the relative clause. This actually occurred in a few examples and appears to be the most common problem of our model. A post-processing rule should eliminate this.

The second example displays a case in which our system and the human system are grammatical, but the removal of a prepositional phrase hurts the resulting meaning of the sentence. In fact, without the knowledge that the sentence is referring to *broadband*, the compressions are meaningless. This appears to be a harder problem – determining which prepositional phrases can be dropped and which cannot.

The final, and more interesting, example presents two very different compressions by the human and our automatic system. Here, the human kept the relative clause relating what languages the source code is available in, but dropped the main verb phrase of the sentence. Our model preferred to retain the main verb phrase and drop the relative clause. This is most likely due to the fact that dropping the main verb phrase of a sentence is much less likely in the training data than dropping a relative clause. Two out of four evaluators preferred the compression returned by our system and the other two rated them equal.

9.3.2 Importance of Dependency Features

Bi-grams versus Syntax

Table 9.3 compares the outputs of two systems. The first, *Only Bigram*, is the sentence compression system describe in this chapter without any syntax features (either dependency or phrase-based). The second, *Normal*, is exactly the sentence compressor described in this chapter. We show examples for which the outputs of the system differed significantly (i.e., 10 out of 32 sentences). Clearly, and not surprisingly, the system is benefitting from syntax. The best example is the final one for the sentence *the source code , which is available for c , fortran , ada and vhdl , can be compiled and executed on the same system or ported to other target platforms*. Here the bigram only system output *ada can be compiled and executed on the same system or ported to other target platforms*. This is obviously unacceptable since *ada* is really part of a noun conjunction phrase within a preposition that modifies the availability of the source code. However, this information cannot be encoded in the bigram only system and as a result, it outputs a meaningless compression (though grammatical).

Another interesting example is the sentence *although it bears the ibm name, it synthesizer circuitry is identical to that of a yamaha fb-01, a popular eight-voice synthesizer module*. The bi-gram only model spits out a very grammatical compression *the ibm circuitry is identical to that of a yamaha fb-01*. However, the semantics of the compression are different from the original, making this compression completely meaningless. On the other hand, the full system outputs *circuitry is identical to that of a yamaha fb-01*. This sentence is slightly ungrammatical and appears to be missing information. However, we can see that the original sentence itself is not grammatical and is missing information, due to an unresolved pronoun.

Full Sentence	standard fonts include courier , line printer (a sans serif face) , times roman and prestige elite .
Only Bigram	standard fonts include , line printer , times roman and prestige elite .
Normal	standard fonts include courier , line printer , times roman and prestige elite .
Full Sentence	with three exabyte drives , it is priced at $17,850 .
Only Bigram	with three is priced at $17,850 .
Normal	it is priced at $17,850 .
Full Sentence	the chemical etching process used for glare protection is effective and will help if your office has the fluorescent-light overkill that 's typical in offices .
Only Bigram	the chemical etching process used for glare protection .
Normal	the chemical etching process used for glare protection is effective .
Full Sentence	apparel makers use them to design clothes and to quickly produce and deliver the best-selling garments .
Only Bigram	clothes and to produce and deliver the best-selling garments .
Normal	apparel makers use to design clothes and to produce and deliver the best-selling garments .
Full Sentence	microsoft alone has lost one-third of its market value .
Only Bigram	microsoft has lost one-third of market value .
Normal	microsoft alone has lost one-third of its market value .
Full Sentence	although it bears the ibm name , it synthesizer circuitry is identical to that of a yamaha fb-01 , a popular eight-voice synthesizer module .
Only Bigram	the ibm circuitry is identical to that of a yamaha fb-01 .
Normal	circuitry is identical to that of a yamaha fb-01 .
Full Sentence	working in the score is not an intuitive process ; it takes a lot of practice .
Only Bigram	working in the score is an intuitive process .
Normal	working in the score is not an intuitive process .
Full Sentence	the utilities will be bundled with quickdex ii in a $90 package called super quickdex , which is expected to ship in late summer .
Only Bigram	the utilities will be bundled with quickdex ii in a quickdex .
Normal	the utilities will be bundled with quickdex ii in a $90 package called super quickdex .
Full Sentence	establishing external (to the application) reference files allows controlled and timely access to all reference data (master files) , regardless of location or application , and enhance data integrity .
Only Bigram	all reference data and enhance data integrity .
Normal	establishing external reference files allows controlled and enhance data integrity .
Full Sentence	the source code , which is available for c , fortran , ada and vhdl , can be compiled and executed on the same system or ported to other target platforms .
Only Bigram	ada can be compiled and executed on the same system or ported to other target platforms .
Normal	the source code can be compiled and executed on the same system or ported to other target platforms .

Table 9.3: Compression system comparison with and without syntactic features.

Only Dependency Features

One interesting question is the effect of phrase-structure features on compression accuracy. If the addition of these features changes performance significantly, it is an argument for the necessity of phrase-based parsing over just labeled dependencies. We trained two systems. One used only labeled dependency features and the other used both dependencies and phrase-structure features (i.e., the normal system). The resulting systems differed in 6 sentences, shown in Table 9.4. *Normal* is the system with both dependencies and phrase-structure features, and *No Phrase-Structure* uses just labeled dependency features.

For the most part the differences between the two systems are small. In Table 9.4, the second, third, fourth and sixth sentences differ little. In the first sentence, *many debugging features , including user-defined break points and variable-watching and message-watching windows , have been added*, the normal system outputs a good compression, *many debugging features have been added*. The dependency only system outputs a grammatical sentence, *many debugging features and variable-watching and message-watching windows have been added,* however, the sentence implies that *variable-watching and message-watching windows* are distinct entities from debugging features, when in fact they are examples of debugging features. In the fifth sentence, the dependency based parser leaves out one of the verbs in this multi-clause sentence. This may be due to the feature indicating whether a verb is not the main verb in the sentence. This feature is usually on for relative clauses, which can often be dropped in the compression. Thus, this feature may have caused the drop of the second verb in the multi-clause sentence, since by convention, the first verb in a verb-conjunction phrase is considered the head.

These results suggest, at least for sentence compression, that labeled dependencies encode sufficient information for accurate compression systems. It would be an interesting future study to compare the difference in performance of labeled dependencies against phrase-structure representations on a variety of language processing problems to determine

Full Sentence	many debugging features , including user-defined break points and variable-watching and message-watching windows , have been added .
No Phrase-Structure	many debugging features and variable-watching and message-watching windows have been added .
Normal	many debugging features have been added .
Full Sentence	microsoft alone has lost one-third of its market value .
No Phrase-Structure	microsoft has lost one-third of its market value .
Normal	microsoft alone has lost one-third of its market value .
Full Sentence	the sas screen is divided into three sections : one for writing programs , one for the system 's response as it executes the program , and a third for output tables and charts .
No Phrase-Structure	the sas screen is divided into three sections .
Normal	the sas screen is divided into three .
Full Sentence	the scamp module , designed and built by unisys and based on an intel process , contains the entire 48-bit a-series processor .
No Phrase-Structure	the scamp module contains the entire 48-bit a-series processor .
Normal	the scamp designed and built by unisys contains the entire 48-bit a-series processor .
Full Sentence	it implements the common cryptographic architecture and offers a comprehensive set of security products that allow users to implement end-to-end secure systems with ibm components .
No Phrase-Structure	it implements the common cryptographic architecture.
Normal	it implements the common cryptographic architecture and offers a comprehensive set of security products .
Full Sentence	the discounted package for the sparcserver 470 is priced at $89,900 , down from the regular $107,795 .
No Phrase-Structure	the discounted package for the 470 is priced at $89,900 .
Normal	the discounted package is priced at $89,900 .

Table 9.4: Compression system comparison with and without phrase-structure features.

which representation is appropriate for what situations.

9.4 Summary of Chapter

In this chapter we have described a new system for sentence compression. The key contribution to this work is that it is a practical example which shows the applicability of our English parser. In particular, we have argued that features over the output of our parser are imperative for accurate compression and even contain most of the information provided by a phrase-based parser.

When compared to previous work on sentence compression, this system has several advantages. First of all, its discriminative nature allows us to use a rich dependent feature set and to optimize a function directly related to compression accuracy during training, both of which have been shown to be beneficial for other problems. Furthermore, the system does not rely on the syntactic parses of the sentences to calculate probability estimates. Instead, this information is incorporated as just another form of evidence to be considered during training. This is advantageous because these parses are trained on out of domain data and often contain mistakes. These syntactic features are available even with an aggressive first-order Markov factorization required for efficient inference.

Chapter 10

Discussion

10.1 Comparison to Recent Work

In the dependency parsing community there has been much recent and related work. Sagae and Lavie [117] use the maximum spanning tree algorithms presented here in a new voted parsing system. The idea is very simple: if one has M different parsers each producing a single best dependency parse, then one can define the score of each edge as the number of parsers that believed that edge was in the correct parse. Using this score function it is possible to find the highest scoring tree. The power behind this method is that it guarantees that graphs satisfying the tree constraint will be returned. Furthermore, weighted votes can easily be determined on a held-out set of data.

The work of Corston-Oliver et al. [32] used the maximum spanning tree framework presented in this work in conjunction with Bayes-Point machines, which is an easy to implement but memory inefficient learning algorithm. One important aspect of that work is that they reimplemented our parsing framework and reported identical results, resulting in an independent verification of many of the claims presented here. Corston-Oliver et al. plan on using their parser to assist in the translation of Microsoft technical documents to other languages.

One interesting area of research not previously mentioned is the work of Klein [75] and Smith and Eisner [128] on unsupervised dependency parsing. Klein and Manning provide a model that uses co-occurrence information coupled with mutual-information between words at possible phrasal boundaries, both of which can be calculated from unlabeled data. Smith and Eisner present a general unsupervised approach to language learning called *contrastive estimation* and present very promising results for dependency parsing.

There has also been new work discriminative phrase-structure parsing. In particular the work of Shen and Joshi [121] extends the discriminative incremental parsing framework of Collins and Roark [30] to a LTAG formalism called LTAG-slim. They present highly accurate results for English. Their parsing framework, being in the TAG family, allows for the modeling of long-distance dependencies and thus non-projectivity. Furthermore, the lexical nature of their formalism allows them to extract dependencies on top of phrase-structure. The work of Turian and Melamed [144, 145] present a discriminative classifier-based parser, which is one of the first discriminative phrase-based parsers to outperform a generative baseline without the explicit use of the baseline parser (e.g., to re-rank or for parameter initialization). The work of Koo and Collins [77] use hidden variables within a discriminative re-ranking framework and show such models can improve parser performance.

One new direction in parsing is self-training methods of McClosky et al. [88, 89]. This method works by taking a discriminative re-ranked parser, running it on millions of new sentences, and retraining a generative parser. The idea is to attempt to alleviate sparsity of the original supervised model by learning parameters over a larger set of training data (albeit noisy). This method was shown to improve both in domain as well as out of domain parser performance.

10.1.1 CoNLL 2006

At the Conference on Natural Language Learning (CoNLL) in 2006, 17 groups submitted parsing systems, which were used to parse 13 different languages [13]. The goal of the experiment was to test various parsing models on a diverse set of languages with varying amounts of training data. Of all the systems submitted, the two most represented parsing frameworks were those of Nivre and his colleagues [105, 104] and the models described in this work. These systems also performed the best empirically in the experiments.

Other systems were based on the work of Yamada and Matsumoto [151], vine parsing algorithms [46], linear integer programming inference techniques and conversions to CFG parsing.

10.2 Future Work

We highlight some areas of future research in this section. This list is not exhaustive, merely just a sample of the most promising next directions to improving and applying the parsing models described here.

Integrated Part-of-Speech Tagging

As noted in Chapter 6 one of the major sources of errors are pipelined errors resulting in noisy part-of-speech tags at test time. The obvious solution to this is to learn to tag and parse in tandem. However, this would result in an increase in complexity by a multiplicative factor cubic in the number of tags. We attempted to minimize this factor by limiting the number of possible tags per token to 2. The resulting parser was slightly more accurate than the original, but has yet to bridge the gap between performance with gold standard tags.

However, other solutions might exist. For example, the work of Daumè and Marcu

[39] on learning as search optimization. In this work a general framework was given for learning relative to approximate and even greedy inference algorithms. One might expect such a framework to be beneficial when searching the joint space of part-of-speech tags and dependency structures. Another option is to train a POS tagger using a dependency parsing loss function. That is, when determining the loss of a particular incorrect tag sequence, we could calculate the number of dependency errors that tag sequence would give rise to in a parsing systems, then update parameters relative to this loss. The hope is that the resulting tagger would be less likely to cause a dependency parser to make errors.

Richer Feature Space

During the course of constructing the parsers of this work, extensive tests over various feature combinations were conducted for English. The resulting feature classes were then use across a plethora of languages with very good results. However, it is reasonable to suspect that language specific feature sets will greatly improve the performance of the parser when extended to other languages. The fact that simple morphological features improved parsing performance of highly inflected languages supports this belief.

Applications

In Chapter 9 we integrated our dependency parser into a sentence compression module to empirically show its applicability to real world problems. This is a rather limited example and a major area of future work is to integrate the parsers described here into new and old NLP applications. In particular, problems such as textual entailment, relation extraction and translation often require a predicate-argument representation of a sentence to accurately process it.

Further Languages

Though we have shown in this work that discriminative spanning tree parsing generalizes well across languages, it is always interesting to test this hypothesis on new languages, and in particular, those languages with little annotated resources. One question is how to bootstrap annotated resources in different languages to improve parsing performance in a resource-poor language, which is similar to domain adaptation. There has been limited but promising work along these lines in the past [68].

Semi-Supervised Dependency Parsing

The recent work of Klein [75] and Smith and Eisner [128] raise questions of whether these methods can be combined with the parsers in this work in a semi-supervised way. One simple technique might be to train an unsupervised parser on a large set of unlabeled data, and use the output of this parser as features in our supervised model. It is unlikely such an approach would work for due to the already large size of the labeled English data set. However, this method might prove successful at improving parsers for languages with much smaller labeled resources.

In addition, the work of Ando and Zhang [3] on semi-supervised discriminative learning might be applied to dependency parsing. In that work, thousands of labeled training examples are automatically generated from unlabeled data, such as *Is the word to the right of the current word a verb?*. Predictors are trained for each of these problems and their feature spaces are used to map inputs to a low dimensional representation that is indicative of good features for common sequence classification problems. This system reported very significant improvements on standard sequence classification tasks. The extension of these models to parsing would be an exciting area of research.

Domain Adaptation

In Chapter 7 we argued that discriminative parsing models have a simple mechanism for adapting out of domain parsers when limited amounts of in domain data is available – the feature set. This is acceptable *if* annotated data does exist, which is not usually the case. Augmenting the feature space or backing off to unsupervised parsing systems will be required to adapt the parser when no annotated data is available. The work of Lease and Charniak [81] address this, however, they still assume some annotated resources in the new domain (i.e., entity and part-of-speech taggers). Recent work by McColosky et al. [89] show that a self-training technique [88] can help to adapt statistical parsers when no annotated resources are available in the new domain.

Another new approach to domain adaptation that assumes no labeled data in the new domain is the work of Blitzer et al. [8]. That work uses large collections of in and out of domain unlabeled data to induce feature correspondences across the two domains. Theoretically, these features are meaningful in both domains and will help an out of domain classifier generalize better to new domains. Blitzer et al. present results for part-of-speech tagging and subsequently use that tagger to improve biomedical parsing with a WSJ parser. However, it is still an open problem of whether such feature correspondences could be learned directly for parsing.

Chapter 11

Summary of Thesis

This chapter summarizes the methods, results and conclusions that have been presented during the course of this work. Each discussed item is one component of the primary argument, that is, *discriminative training algorithms and maximum spanning tree inference algorithms combine to provide a highly accurate dependency parsing framework that works across languages and is easily extensible*.

1. In Chapter 2 we first showed that discriminatively trained models for structure outputs can be achieved through the use of highly efficient and accurate online large-margin techniques. These learning algorithms compare favorably to current state-of-the-art discriminative models such as conditional random fields [80] and M^3 networks [137]. A major advantage of the particular learning algorithm used (MIRA) is that parameter updates are based only on inference (either single-best or k-best), which essentially abstracts away learning to allow us to focus on defining efficient inference algorithms for searching the output space. Thus, probabilistic inference techniques such as forward-backward, inside-outside or marginal distribution calculations need no longer concern us.

2. The core of the work was presented in Chapter 3. Here, three major contributions

were made. First, by factoring the score of a dependency by the score of its edges, we showed that tractable inference is possible for both projective and non-projective dependency trees by appealing to maximum spanning tree algorithms. This result is the first that provides an exact inference algorithm for non-projective structures[1]. In fact, the worst-case run-time complexity for non-projective trees is actually better than the case of projective trees ($O(n^2)$ vs. $O(n^3)$). The second major contribution was to extend the maximum spanning tree parsing framework, allowing scores to incorporate features over pairs of edges in the tree. With this extension, parsing remained polynomial for the projective case, but became NP-hard for the non-projective case. This result essentially shows that only under single edge based factorization can non-projective parsing be obtained tractably. Any attempt to extend scores or constraints beyond a single edge and the problem makes the problem NP-hard. To overcome this, we defined a simple approximation that was experimentally validated in Chapter 4. Finally, we defined a rich feature set that contained features over the words occurring in the dependency relation, but more importantly, features occurring over the words surrounding and between them. These features (as was shown in Chapter 6) are crucial to achieving state-of-the-art performance with a spanning tree parser.

3. Chapter 4 contains a set of detailed experiments running the parsing models on English, Czech and Chinese data sets. It is shown that the parsers provide state-of-the-art performance across all languages. In particular, allowing scores over pairs of edges and introducing non-projective edges increases performance significantly. It was also shown that these models can easily be extended to allow for labeled dependency graphs through either joint or two-stage labeling. In both cases the models perform very well for English. However, two-stage labeling is much more computationally efficient, which led to its use in later experiments from Chapter 5. In

[1]This result was discovered independently in Ribarov [112]

Chapter 6, we analyzed our English parser both in terms of errors and in terms of the feature space. We showed that the parser tends to make common parsing errors on prepositions, conjunctions and multi-verb sentences. One area of future work is to define new features to handle these specific constructions. English parsing performance was also broken down by part-of-speech errors, sentence length and errors due to the online nature of learning. One interesting result is that as sentence length grows, root attachment accuracy scales relative to overall accuracy. It was argued that this was a result of using exact search without any pruning or greedy parse decisions. In the second part of Chapter 6 we analyzed the feature space. The primary result is that context features are much more important to first-order parsing models than second-order parsing models. This indicates that these features are in many ways simulating higher-order parsing decisions and help to explain the powerful performance of the aggressive edge based factorization model. Finally, we attempted feature selection and showed that the simplest method, count cut-off, performed better than information gain.

4. One of the more exciting aspects of the models presented in this work is their language generality. In Chapter 5 we presented parsing results for 14 diverse languages using a single learning method and no language specific enhancements. These results are competitive across all 14 languages (see Buckholz et al. [13]). Furthermore, we displayed that these discriminative parsing models are easily extensible by defining new features on derived morphological information for highly inflected languages. These features improved accuracy by 1-3% absolute for these languages. Due to the discriminative nature of the learning technique, additional features are easily incorporated. In contrast, generative models usually require new levels of back-off to be designed to overcome sparsity. In Chapter 6 we presented a brief analysis of common parsing errors across languages and attempted to explain the variability in parser

performance across languages. In particular, we showed that parser variability can mostly be explained through properties of the data including sentence length, lexical sparsity and invalid i.i.d. assumptions between the training and test sets. These results lend weight to our argument that the parsers are indeed language independent.

5. In Chapter 7 we further showed the extensibility of discriminative parsing models by incorporating features over auxiliary classifiers. To improve the performance of our English WSJ parser we incorporated features over the decisions of the Collins [25] and Charniak [16] parsing models. Including these features improved parsing accuracy to 93.2%, which is significantly higher than the accuracy of any single individual model. Adding features based on the output of auxiliary classifiers also improved parser performance for out of domain data. We showed that training a parser on a small amount of biomedical training data can be significantly improved by incorporating features over the output of a WSJ parser. This simple technique should help to quickly create state-of-the-art parsing models for domains in which little training data is available.

6. One advantageous aspect of inference-based online learning is its robustness to approximate inference [30, 39], due to the fact that parameters are set relative to inference. Thus, if inference is approximate, then model parameters can be set to offset common errors introduced due to search approximations. This feature is important considering our second-order non-projective parsing model is approximate. We further displayed this advantage by building a model to parse general dependency graphs that do not necessarily satisfy the tree constraint. It has been argued [67] that non-tree graphs are appropriate for secondary modifiers resulting from relative clause or verb conjunctions. Unfortunately, producing high scoring non-tree dependency graphs can be shown to be NP-hard, thus we devised an approximate algorithm to find them.

This model was tested empirically on a set of Danish dependency graphs and showed improved performance over the tree based models.

7. The final contribution of this work was to show that the dependency models created here could be used in a standard NLP task. We chose to look at sentence compression. First, we defined a search algorithm that allowed us to search through the space of compressions efficiently, while at the same time allowing for the introduction of relevant dependency information. Using this inference algorithm we trained a model and presented empirical results showing state-of-the-art performance on a small data set. Furthermore, we argued (through the output of the system) that most of the necessary information for creating accurate compressions is contained in labeled dependency graphs, and that phrase-structure information does not significantly improve performance. Finally, we showed that if dependency information is not used, the output of the sentence compressor suffered greatly.

Appendix A

English Head Percolation Rules

These are the head percolation rules used to convert the Penn Treebank to depedencies in this work, based on those in [151]. For each non-terminal (on the left of each rule) we start searching from the right or left (r or l) for the first element, then the second, etc. If elements are seperated by a bar, then we search for any of the elements. These rules are run recursively until pre-terminal nodes are reached and those lexical items become the head of the phrase.

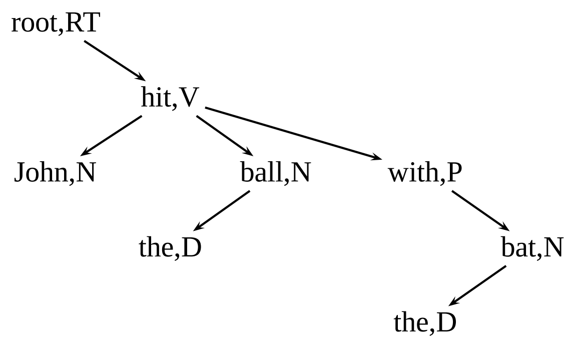
```
NP      r  POS|NN|NNP|NNPS|NNS NX JJR CD JJ JJS RB QP NP
ADJP    r  NNS QP NN \$ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP    l  RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
CONJP   l  CC RB IN
FRAG    l
INTJ    r
LST     l  LS :
NAC     r  NN|NNS|NNP|NNPS NP NAC EX \$ CD QP PRP VBG JJ JJS JJR ADJP FW
PP      l  IN TO VBG VBN RP FW
PRN     r
PRT     l  RP
QP      r  \$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
RRC     l  VP NP ADVP ADJP PP
S       r  TO IN VP S SBAR ADJP UCP NP
SBAR    r  WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
SBARQ   r  SQ S SINV SBARQ FRAG
SINV    r  VBZ VBD VBP VB MD VP S SINV ADJP NP
SQ      r  VBZ VBD VBP VB MD VP SQ
UCP     l
VP      l  VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP
WHADJP  r  CC WRB JJ ADJP
```

```
WHADVP  l  CC WRB
WHNP    r  WDT WP WP\$ WHADJP WHPP WHNP
WHPP    l  IN TO FW
NX      r  POS|NN|NNP|NNPS|NNS NX JJR CD JJ JJS RB QP NP
X       r
```

Appendix B

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(i, j) for the edge (hit,with)

Basic Features

x_i-word="hit", x_i-pos="V", x_j-word="with", x_j-pos="P"

x_i-pos="V", x_j-word="with", x_j-pos="P"

x_i-word="hit", x_j-word="with", x_j-pos="P"

x_i-word="hit", x_i-pos="V", x_j-pos="P"

x_i-word=“hit”, x_i-pos=“V”, x_j-word=“with”

x_i-word=“hit”, x_j-word=“with”

x_i-pos=“V”, x_j-pos=“P”

x_i-word=“hit”, x_i-pos=“V”

x_j-word=“with”, x_j-pos=“P”

x_i-word=“hit”

x_i-pos=“V”

x_j-word=“with”

x_j-pos=“P”

Extended Features

x_i-pos=“V”, b-pos=“D”, x_j-pos=“P”

x_i-pos=“V”, b-pos=“N”, x_j-pos=“P”

x_i-pos=“V”, x_i-pos+1=“D”, x_j-pos-1=“N”, x_j-pos=“P”

x_i-pos=“V”, x_j-pos-1=“N”, x_j-pos=“P”

x_i-pos=“V”, x_i-pos+1=“D”, x_j-pos=“P”

x_i-pos-1=“N”, x_i-pos=“V”, x_j-pos-1=“N”, x_j-pos=“P”

x_i-pos=“V”, x_j-pos-1=“N”, x_j-pos=“P”

x_i-pos-1=“N”, x_i-pos=“V”, x_j-pos=“P”

x_i-pos=“V”, x_i-pos+1=“D”, x_j-pos=“P”, x_j-pos+1=“D”

x_i-pos=“V”, x_j-pos=“P”, x_j-pos+1=“D”

x_i-pos=“V”, x_i-pos+1=“D”, x_j-pos=“P”

x_i-pos-1=“N”, x_i-pos=“V”, x_j-pos=“P”, x_j-pos+1=“D”

x_i-pos=“V”, x_j-pos=“P”, x_j-pos+1=“D”

x_i-pos-1=“N”, x_i-pos=“V”, x_j-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,

x_i-word:5="smash", x_j-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,

x_i-word="hit", x_j-word="with"

the system would also have the feature,

x_i-word="hit", x_j-word="with", dir=R, dist=3

to indicate that the modifier *with* is 3 words to the right of the head *hit*. Distances were calculated into buckets with thresholds of 1, 2, 3, 4, 5 and 10.

B.1 Second-Order Features

If we consider the second-order edge, (hit,ball,with) we will add the following features, $\mathbf{f}(i, k, j)$:

Second-Order Features

x_i-pos="V", x_k-pos="N", x_j-pos="P"

x_k-pos="N", x_j-pos="P"

x_k-word="ball", x_j-pos="P"

x_k-pos="N", x_j-word="with"

x_k-word="ball", x_j-word="with"

The features are again conjoined with attachment direction as well as distance between the siblings unlike the first-order features, which use distance to head. The distances are bucketed as before. For instance, the representation would contain the following feature

x_k-pos="N", x_j-pos="P", dir=R, dist=1

For the cases when a word is the first left/right modifier of a head, the feature representation encodes this. For instance, if we consider the second order edge, (hit,-,ball), we add the features:

Second-Order Features

x_i-pos="V", x_k-pos="N", x_j-pos="N"

x_k-pos="first-RHS-POS", x_j-pos="N"

x_k-word="first-RHS", x_j-pos="N"

x_k-pos="first-RHS-POS", x_j-word="with"

x_k-word="first-RHS", x_j-word="with"

Appendix C

Detailed Experimental Results

Tables in this appendix were produced using the CoNLL 2006 shared task [13] evaluation script (http://nextens.uvt.nl/˜conll/software.html).

C.1 Arabic

```
Labeled   attachment score: 3339 / 4990 * 100 = 66.91 %
Unlabeled attachment score: 3959 / 4990 * 100 = 79.34 %
Label accuracy score:       3967 / 4990 * 100 = 79.50 %

================================================================================

Evaluation of the results in arabic.proj.pred
vs. gold standard arabic_PADT_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 383

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4990 |  3959 |  79% |  3967 |  79% |  3339 |  67%
-----------+-------+-------+------+-------+------+-------+-------
N          |  2007 |  1775 |  88% |  1462 |  73% |  1363 |  68%
P          |   800 |   493 |  62% |   737 |  92% |   467 |  58%
A          |   432 |   395 |  91% |   404 |  94% |   376 |  87%
```

```
V          |   417 |   295 |  71% |   331 |  79% |   258 |  62%
C          |   392 |   250 |  64% |   314 |  80% |   233 |  59%
S          |   349 |   298 |  85% |   290 |  83% |   261 |  75%
Z          |   192 |   162 |  84% |   139 |  72% |   124 |  65%
F          |   143 |   116 |  81% |   121 |  85% |   112 |  78%
X          |   140 |   107 |  76% |    99 |  71% |    86 |  61%
D          |    74 |    38 |  51% |    42 |  57% |    33 |  45%
Q          |    42 |    29 |  69% |    27 |  64% |    25 |  60%
-          |     1 |     1 | 100% |     1 | 100% |     1 | 100%
I          |     1 |     0 |   0% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4990 |  1031 |  21% |  1023 |  21% |   403 |   8%
-----------+-------+-------+------+-------+------+-------+-------
N          |  2007 |   232 |  12% |   545 |  27% |   133 |   7%
P          |   800 |   307 |  38% |    63 |   8% |    37 |   5%
A          |   432 |    37 |   9% |    28 |   6% |     9 |   2%
V          |   417 |   122 |  29% |    86 |  21% |    49 |  12%
C          |   392 |   142 |  36% |    78 |  20% |    61 |  16%
S          |   349 |    51 |  15% |    59 |  17% |    22 |   6%
Z          |   192 |    30 |  16% |    53 |  28% |    15 |   8%
F          |   143 |    27 |  19% |    22 |  15% |    18 |  13%
X          |   140 |    33 |  24% |    41 |  29% |    20 |  14%
D          |    74 |    36 |  49% |    32 |  43% |    27 |  36%
Q          |    42 |    13 |  31% |    15 |  36% |    11 |  26%
-          |     1 |     0 |   0% |     0 |   0% |     0 |   0%
I          |     1 |     1 | 100% |     1 | 100% |     1 | 100%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
Adv             |  378 |     181 |    349 |      47.88 |         51.86
AdvAtr          |    7 |       0 |      0 |       0.00 |           NaN
Ante            |    7 |       0 |      0 |       0.00 |           NaN
Apos            |   10 |       2 |      2 |      20.00 |        100.00
Atr             | 1930 |    1751 |   2130 |      90.73 |         82.21
AtrAdv          |    7 |       0 |      4 |       0.00 |          0.00
AtrAtr          |    3 |       0 |      0 |       0.00 |           NaN
AtrObj          |    1 |       0 |      0 |       0.00 |           NaN
Atv             |   50 |      18 |     26 |      36.00 |         69.23
AuxC            |  102 |      88 |    115 |      86.27 |         76.52
AuxE            |   33 |      15 |     26 |      45.45 |         57.69
```

```
AuxG            |    1 |       0 |      0 |       0.00 |           NaN
AuxM            |   76 |      71 |     76 |      93.42 |         93.42
AuxP            |  767 |     733 |    773 |      95.57 |         94.83
AuxY            |  324 |     233 |    285 |      71.91 |         81.75
Coord           |  171 |     140 |    187 |      81.87 |         74.87
ExD             |   71 |      37 |     44 |      52.11 |         84.09
Obj             |  520 |     297 |    482 |      57.12 |         61.62
Pnom            |   19 |       0 |      4 |       0.00 |          0.00
Pred            |  172 |     147 |    168 |      85.47 |         87.50
PredE           |    1 |       1 |      2 |     100.00 |         50.00
PredP           |    1 |       0 |      0 |       0.00 |           NaN
Ref             |    3 |       1 |      1 |      33.33 |        100.00
Sb              |  336 |     252 |    316 |      75.00 |         79.75


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
Adv             |  378 |     173 |    349 |      45.77 |         49.57
AdvAtr          |    7 |       0 |      0 |       0.00 |           NaN
Ante            |    7 |       0 |      0 |       0.00 |           NaN
Apos            |   10 |       1 |      2 |      10.00 |         50.00
Atr             | 1930 |    1605 |   2130 |      83.16 |         75.35
AtrAdv          |    7 |       0 |      4 |       0.00 |          0.00
AtrAtr          |    3 |       0 |      0 |       0.00 |           NaN
AtrObj          |    1 |       0 |      0 |       0.00 |           NaN
Atv             |   50 |      17 |     26 |      34.00 |         65.38
AuxC            |  102 |      71 |    115 |      69.61 |         61.74
AuxE            |   33 |      10 |     26 |      30.30 |         38.46
AuxG            |    1 |       0 |      0 |       0.00 |           NaN
AuxM            |   76 |      70 |     76 |      92.11 |         92.11
AuxP            |  767 |     462 |    773 |      60.23 |         59.77
AuxY            |  324 |     197 |    285 |      60.80 |         69.12
Coord           |  171 |      86 |    187 |      50.29 |         45.99
ExD             |   71 |      35 |     44 |      49.30 |         79.55
Obj             |  520 |     261 |    482 |      50.19 |         54.15
Pnom            |   19 |       0 |      4 |       0.00 |          0.00
Pred            |  172 |     121 |    168 |      70.35 |         72.02
PredE           |    1 |       1 |      2 |     100.00 |         50.00
PredP           |    1 |       0 |      0 |       0.00 |           NaN
Ref             |    3 |       1 |      1 |      33.33 |        100.00
Sb              |  336 |     228 |    316 |      67.86 |         72.15


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  295 |     247 |    280 |      83.73 |         88.21
left            | 4157 |    4035 |   4233 |      97.07 |         95.32
```

```
right           |  538 |     356 |    477 |      66.17 |         74.63
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  295 |     247 |    280 |      83.73 |         88.21
1               | 3081 |    2897 |   3176 |      94.03 |         91.22
2               |  637 |     437 |    639 |      68.60 |         68.39
3-6             |  626 |     402 |    637 |      64.22 |         63.11
7-...           |  351 |     158 |    258 |      45.01 |         61.24
```

C.2 Bulgarian

```
Labeled   attachment score: 4390 / 5013 * 100 = 87.57 %
Unlabeled attachment score: 4614 / 5013 * 100 = 92.04 %
Label accuracy score:       4547 / 5013 * 100 = 90.70 %

================================================================================

Evaluation of the results in bulgarian.proj.pred
vs. gold standard bulgarian_bultreebank_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 921

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5013 |  4614 |  92% |  4547 |  91% |  4390 |  88%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1566 |  1476 |  94% |  1426 |  91% |  1386 |  89%
V          |   831 |   773 |  93% |   786 |  95% |   766 |  92%
R          |   827 |   701 |  85% |   655 |  79% |   620 |  75%
P          |   469 |   438 |  93% |   435 |  93% |   426 |  91%
A          |   436 |   425 |  97% |   427 |  98% |   423 |  97%
C          |   274 |   232 |  85% |   268 |  98% |   230 |  84%
T          |   223 |   209 |  94% |   204 |  91% |   200 |  90%
D          |   170 |   148 |  87% |   141 |  83% |   134 |  79%
M          |   147 |   142 |  97% |   136 |  93% |   136 |  93%
H          |    70 |    70 | 100% |    69 |  99% |    69 |  99%
-----------+-------+-------+------+-------+------+-------+-------
```

```
The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5013 |   399 |   8% |   466 |   9% |   242 |   5%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1566 |    90 |   6% |   140 |   9% |    50 |   3%
V          |   831 |    58 |   7% |    45 |   5% |    38 |   5%
R          |   827 |   126 |  15% |   172 |  21% |    91 |  11%
P          |   469 |    31 |   7% |    34 |   7% |    22 |   5%
A          |   436 |    11 |   3% |     9 |   2% |     7 |   2%
C          |   274 |    42 |  15% |     6 |   2% |     4 |   1%
T          |   223 |    14 |   6% |    19 |   9% |    10 |   4%
D          |   170 |    22 |  13% |    29 |  17% |    15 |   9%
M          |   147 |     5 |   3% |    11 |   7% |     5 |   3%
H          |    70 |     0 |   0% |     1 |   1% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ROOT            |  398 |     387 |    398 |      97.24 |         97.24
adjunct         |  347 |     237 |    319 |      68.30 |         74.29
clitic          |   84 |      77 |     77 |      91.67 |        100.00
comp            |  483 |     449 |    481 |      92.96 |         93.35
conj            |  184 |     183 |    183 |      99.46 |        100.00
conjarg         |  208 |     186 |    199 |      89.42 |         93.47
indobj          |  128 |      86 |    124 |      67.19 |         69.35
marked          |   91 |      90 |     95 |      98.90 |         94.74
mod             | 1299 |    1232 |   1370 |      94.84 |         89.93
obj             |  218 |     172 |    217 |      78.90 |         79.26
pragadjunct     |   64 |      41 |     48 |      64.06 |         85.42
prepcomp        |  831 |     821 |    828 |      98.80 |         99.15
subj            |  405 |     348 |    408 |      85.93 |         85.29
xadjunct        |   55 |      48 |     56 |      87.27 |         85.71
xcomp           |  131 |     121 |    127 |      92.37 |         95.28
xmod            |   71 |      56 |     64 |      78.87 |         87.50
xprepcomp       |    4 |       4 |      5 |     100.00 |         80.00
xsubj           |   12 |       9 |     14 |      75.00 |         64.29


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ROOT            |  398 |     387 |    398 |      97.24 |         97.24
```

```
adjunct         |  347 |     227 |    319 |      65.42 |         71.16
clitic          |   84 |      76 |     77 |      90.48 |         98.70
comp            |  483 |     444 |    481 |      91.93 |         92.31
conj            |  184 |     153 |    183 |      83.15 |         83.61
conjarg         |  208 |     150 |    199 |      72.12 |         75.38
indobj          |  128 |      84 |    124 |      65.62 |         67.74
marked          |   91 |      90 |     95 |      98.90 |         94.74
mod             | 1299 |    1188 |   1370 |      91.45 |         86.72
obj             |  218 |     172 |    217 |      78.90 |         79.26
pragadjunct     |   64 |      37 |     48 |      57.81 |         77.08
prepcomp        |  831 |     821 |    828 |      98.80 |         99.15
subj            |  405 |     339 |    408 |      83.70 |         83.09
xadjunct        |   55 |      43 |     56 |      78.18 |         76.79
xcomp           |  131 |     119 |    127 |      90.84 |         93.70
xmod            |   71 |      47 |     64 |      66.20 |         73.44
xprepcomp       |    4 |       4 |      5 |     100.00 |         80.00
xsubj           |   12 |       9 |     14 |      75.00 |         64.29


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  398 |     387 |    398 |      97.24 |         97.24
left            | 3196 |    3156 |   3200 |      98.75 |         98.62
right           | 1419 |    1376 |   1415 |      96.97 |         97.24
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  398 |     387 |    398 |      97.24 |         97.24
1               | 2630 |    2566 |   2664 |      97.57 |         96.32
2               |  926 |     862 |    914 |      93.09 |         94.31
3-6             |  796 |     687 |    795 |      86.31 |         86.42
7-...           |  263 |     197 |    242 |      74.90 |         81.40
```

C.3 Chinese

```
Labeled   attachment score: 4269 / 4970 * 100 = 85.90 %
Unlabeled attachment score: 4526 / 4970 * 100 = 91.07 %
Label accuracy score:       4385 / 4970 * 100 = 88.23 %

================================================================================

Evaluation of the results in chinese.proj.pred
```

```
vs. gold standard chinese_sinica_test.conll:

Legend: ’.S’ - the beginning of a sentence, ’.E’ - the end of a sentence

Number of non-scoring tokens: 42

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4970 |  4526 |  91% |  4385 |  88% |  4269 |  86%
-----------+-------+-------+------+-------+------+-------+-------
N          |  2021 |  1829 |  90% |  1786 |  88% |  1746 |  86%
V          |  1210 |  1082 |  89% |  1080 |  89% |  1064 |  88%
D          |   505 |   489 |  97% |   504 | 100% |   488 |  97%
DE         |   407 |   389 |  96% |   309 |  76% |   301 |  74%
P          |   251 |   231 |  92% |   201 |  80% |   195 |  78%
C          |   170 |   155 |  91% |   151 |  89% |   148 |  87%
DM         |   148 |   128 |  86% |   122 |  82% |   116 |  78%
Ne         |   129 |   114 |  88% |   117 |  91% |   105 |  81%
Ng         |    81 |    66 |  81% |    68 |  84% |    63 |  78%
A          |    32 |    29 |  91% |    31 |  97% |    29 |  91%
T          |    16 |    14 |  88% |    16 | 100% |    14 |  88%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4970 |   444 |   9% |   585 |  12% |   328 |   7%
-----------+-------+-------+------+-------+------+-------+-------
N          |  2021 |   192 |  10% |   235 |  12% |   152 |   8%
V          |  1210 |   128 |  11% |   130 |  11% |   112 |   9%
D          |   505 |    16 |   3% |     1 |   0% |     0 |   0%
DE         |   407 |    18 |   4% |    98 |  24% |    10 |   2%
P          |   251 |    20 |   8% |    50 |  20% |    14 |   6%
C          |   170 |    15 |   9% |    19 |  11% |    12 |   7%
DM         |   148 |    20 |  14% |    26 |  18% |    14 |   9%
Ne         |   129 |    15 |  12% |    12 |   9% |     3 |   2%
Ng         |    81 |    15 |  19% |    13 |  16% |    10 |  12%
A          |    32 |     3 |   9% |     1 |   3% |     1 |   3%
T          |    16 |     2 |  12% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
```

```
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
DUMMY           |  320 |     289 |    325 |      90.31 |         88.92
DUMMY1          |   78 |      73 |     75 |      93.59 |         97.33
DUMMY2          |   96 |      87 |     98 |      90.62 |         88.78
Head            |   29 |      13 |     25 |      44.83 |         52.00
ROOT            |  864 |     812 |    864 |      93.98 |         93.98
addition        |   18 |      17 |     17 |      94.44 |        100.00
agent           |  128 |     108 |    147 |      84.38 |         73.47
apposition      |   42 |      37 |     41 |      88.10 |         90.24
aspect          |   39 |      39 |     39 |     100.00 |        100.00
benefactor      |    8 |       6 |      6 |      75.00 |        100.00
causer          |   10 |       3 |      4 |      30.00 |         75.00
companion       |   15 |      14 |     18 |      93.33 |         77.78
comparison      |   10 |       9 |     12 |      90.00 |         75.00
complement      |   63 |      44 |     62 |      69.84 |         70.97
concession      |    7 |       7 |      7 |     100.00 |        100.00
condition       |   16 |       7 |     15 |      43.75 |         46.67
contrast        |   22 |      20 |     22 |      90.91 |         90.91
conversion      |    2 |       1 |      1 |      50.00 |        100.00
degree          |   67 |      67 |     68 |     100.00 |         98.53
deixis          |    5 |       5 |      5 |     100.00 |        100.00
deontics        |   54 |      54 |     54 |     100.00 |        100.00
duration        |    4 |       1 |      3 |      25.00 |         33.33
epistemics      |   42 |      42 |     42 |     100.00 |        100.00
evaluation      |   89 |      89 |     89 |     100.00 |        100.00
experiencer     |   18 |      17 |     17 |      94.44 |        100.00
frequency       |    7 |       2 |      4 |      28.57 |         50.00
goal            |  354 |     301 |    351 |      85.03 |         85.75
head            |  333 |     297 |    340 |      89.19 |         87.35
hypothesis      |   12 |      12 |     12 |     100.00 |        100.00
instrument      |    6 |       4 |      5 |      66.67 |         80.00
location        |   77 |      53 |     68 |      68.83 |         77.94
manner          |   96 |      72 |     79 |      75.00 |         91.14
negation        |   43 |      43 |     43 |     100.00 |        100.00
nominal         |   31 |      27 |     28 |      87.10 |         96.43
particle        |   16 |      16 |     16 |     100.00 |        100.00
possessor       |   36 |       5 |      9 |      13.89 |         55.56
predication     |   62 |      15 |     27 |      24.19 |         55.56
property        |  849 |     799 |    940 |      94.11 |         85.00
purpose         |    1 |       1 |      1 |     100.00 |        100.00
quantifier      |  220 |     193 |    208 |      87.73 |         92.79
quantity        |   63 |      59 |     63 |      93.65 |         93.65
range           |  178 |     167 |    177 |      93.82 |         94.35
reason          |   16 |      15 |     18 |      93.75 |         83.33
recipient       |    1 |       0 |      0 |       0.00 |           NaN
restriction     |    6 |       6 |      6 |     100.00 |        100.00
result          |   11 |      10 |     14 |      90.91 |         71.43
source          |    0 |       0 |      1 |        NaN |          0.00
standard        |    1 |       1 |      1 |     100.00 |        100.00
target          |   12 |      10 |     12 |      83.33 |         83.33
theme           |  334 |     281 |    339 |      84.13 |         82.89
time            |  144 |     133 |    144 |      92.36 |         92.36
```

```
topic           |   13 |       0 |      5 |       0.00 |          0.00
whatever        |    2 |       2 |      3 |     100.00 |         66.67


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
DUMMY           |  320 |     288 |    325 |      90.00 |         88.62
DUMMY1          |   78 |      73 |     75 |      93.59 |         97.33
DUMMY2          |   96 |      87 |     98 |      90.62 |         88.78
Head            |   29 |      13 |     25 |      44.83 |         52.00
ROOT            |  864 |     812 |    864 |      93.98 |         93.98
addition        |   18 |      17 |     17 |      94.44 |        100.00
agent           |  128 |     106 |    147 |      82.81 |         72.11
apposition      |   42 |      37 |     41 |      88.10 |         90.24
aspect          |   39 |      39 |     39 |     100.00 |        100.00
benefactor      |    8 |       6 |      6 |      75.00 |        100.00
causer          |   10 |       3 |      4 |      30.00 |         75.00
companion       |   15 |      14 |     18 |      93.33 |         77.78
comparison      |   10 |       8 |     12 |      80.00 |         66.67
complement      |   63 |      42 |     62 |      66.67 |         67.74
concession      |    7 |       7 |      7 |     100.00 |        100.00
condition       |   16 |       7 |     15 |      43.75 |         46.67
contrast        |   22 |      19 |     22 |      86.36 |         86.36
conversion      |    2 |       1 |      1 |      50.00 |        100.00
degree          |   67 |      66 |     68 |      98.51 |         97.06
deixis          |    5 |       5 |      5 |     100.00 |        100.00
deontics        |   54 |      54 |     54 |     100.00 |        100.00
duration        |    4 |       1 |      3 |      25.00 |         33.33
epistemics      |   42 |      39 |     42 |      92.86 |         92.86
evaluation      |   89 |      84 |     89 |      94.38 |         94.38
experiencer     |   18 |      17 |     17 |      94.44 |        100.00
frequency       |    7 |       2 |      4 |      28.57 |         50.00
goal            |  354 |     295 |    351 |      83.33 |         84.05
head            |  333 |     295 |    340 |      88.59 |         86.76
hypothesis      |   12 |      11 |     12 |      91.67 |         91.67
instrument      |    6 |       4 |      5 |      66.67 |         80.00
location        |   77 |      51 |     68 |      66.23 |         75.00
manner          |   96 |      72 |     79 |      75.00 |         91.14
negation        |   43 |      42 |     43 |      97.67 |         97.67
nominal         |   31 |      26 |     28 |      83.87 |         92.86
particle        |   16 |      14 |     16 |      87.50 |         87.50
possessor       |   36 |       5 |      9 |      13.89 |         55.56
predication     |   62 |      14 |     27 |      22.58 |         51.85
property        |  849 |     746 |    940 |      87.87 |         79.36
purpose         |    1 |       1 |      1 |     100.00 |        100.00
quantifier      |  220 |     175 |    208 |      79.55 |         84.13
quantity        |   63 |      56 |     63 |      88.89 |         88.89
range           |  178 |     165 |    177 |      92.70 |         93.22
reason          |   16 |      15 |     18 |      93.75 |         83.33
recipient       |    1 |       0 |      0 |       0.00 |           NaN
```

```
restriction     |    6 |       6 |      6 |     100.00 |        100.00
result          |   11 |      10 |     14 |      90.91 |         71.43
source          |    0 |       0 |      1 |        NaN |          0.00
standard        |    1 |       1 |      1 |     100.00 |        100.00
target          |   12 |       9 |     12 |      75.00 |         75.00
theme           |  334 |     277 |    339 |      82.93 |         81.71
time            |  144 |     130 |    144 |      90.28 |         90.28
topic           |   13 |       0 |      5 |       0.00 |          0.00
whatever        |    2 |       2 |      3 |     100.00 |         66.67


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  864 |     812 |    864 |      93.98 |         93.98
left            | 1172 |    1069 |   1166 |      91.21 |         91.68
right           | 2934 |    2833 |   2940 |      96.56 |         96.36
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  864 |     812 |    864 |      93.98 |         93.98
1               | 2332 |    2247 |   2375 |      96.36 |         94.61
2               |  792 |     715 |    797 |      90.28 |         89.71
3-6             |  843 |     728 |    815 |      86.36 |         89.33
7-...           |  139 |     108 |    119 |      77.70 |         90.76
```

C.4 Czech

```
Labeled   attachment score: 4009 / 5000 * 100 = 80.18 %
Unlabeled attachment score: 4365 / 5000 * 100 = 87.30 %
Label accuracy score:       4336 / 5000 * 100 = 86.72 %

================================================================================

Evaluation of the results in czech.nonproj.pred
vs. gold standard czech_pdt_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 853

The overall accuracy and its distribution over CPOSTAGs
```

```
-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5000 |  4365 |  87% |  4336 |  87% |  4009 |  80%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1748 |  1573 |  90% |  1431 |  82% |  1364 |  78%
V          |   708 |   591 |  83% |   562 |  79% |   540 |  76%
A          |   692 |   668 |  97% |   667 |  96% |   658 |  95%
R          |   598 |   473 |  79% |   593 |  99% |   470 |  79%
P          |   404 |   386 |  96% |   353 |  87% |   346 |  86%
D          |   336 |   274 |  82% |   291 |  87% |   257 |  76%
J          |   321 |   242 |  75% |   292 |  91% |   238 |  74%
C          |   159 |   129 |  81% |   118 |  74% |   109 |  69%
T          |    34 |    29 |  85% |    29 |  85% |    27 |  79%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5000 |   635 |  13% |   664 |  13% |   308 |   6%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1748 |   175 |  10% |   317 |  18% |   108 |   6%
V          |   708 |   117 |  17% |   146 |  21% |    95 |  13%
A          |   692 |    24 |   3% |    25 |   4% |    15 |   2%
R          |   598 |   125 |  21% |     5 |   1% |     2 |   0%
P          |   404 |    18 |   4% |    51 |  13% |    11 |   3%
D          |   336 |    62 |  18% |    45 |  13% |    28 |   8%
J          |   321 |    79 |  25% |    29 |   9% |    25 |   8%
C          |   159 |    30 |  19% |    41 |  26% |    21 |  13%
T          |    34 |     5 |  15% |     5 |  15% |     3 |   9%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
Adv             |  576 |     474 |    567 |      82.29 |         83.60
AdvAtr          |    6 |       0 |      0 |       0.00 |           NaN
Adv_Ap          |    6 |       0 |      3 |       0.00 |          0.00
Adv_Co          |   39 |      13 |     32 |      33.33 |         40.62
Adv_Pa          |    7 |       1 |      2 |      14.29 |         50.00
Apos            |    3 |       2 |      4 |      66.67 |         50.00
Apos_Co         |    1 |       1 |      1 |     100.00 |        100.00
Atr             | 1514 |    1441 |   1563 |      95.18 |         92.19
AtrAdv          |    9 |       0 |      0 |       0.00 |           NaN
AtrAtr          |    4 |       0 |      0 |       0.00 |           NaN
```

```
Atr_Ap          |    0 |       0 |      4 |        NaN |          0.00
Atr_Co          |  134 |      81 |    134 |      60.45 |         60.45
Atr_Pa          |    2 |       0 |      0 |       0.00 |           NaN
Atv             |   10 |       6 |      8 |      60.00 |         75.00
AtvV            |    2 |       0 |      1 |       0.00 |          0.00
Atv_Co          |    2 |       0 |      0 |       0.00 |           NaN
AuxC            |  101 |     100 |    103 |      99.01 |         97.09
AuxO            |    1 |       0 |      2 |       0.00 |          0.00
AuxP            |  610 |     608 |    614 |      99.67 |         99.02
AuxR            |   23 |      12 |     17 |      52.17 |         70.59
AuxT            |   63 |      58 |     74 |      92.06 |         78.38
AuxV            |   82 |      76 |     82 |      92.68 |         92.68
AuxY            |   52 |      39 |     44 |      75.00 |         88.64
AuxZ            |  106 |      93 |    108 |      87.74 |         86.11
Coord           |  157 |     146 |    161 |      92.99 |         90.68
Coord_Ap        |    1 |       0 |      2 |       0.00 |          0.00
Coord_Co        |   11 |       6 |     10 |      54.55 |         60.00
ExD             |   59 |      44 |     56 |      74.58 |         78.57
ExD_Ap          |   17 |       4 |      6 |      23.53 |         66.67
ExD_Co          |   64 |      31 |     74 |      48.44 |         41.89
ExD_Pa          |   14 |      10 |     13 |      71.43 |         76.92
Obj             |  426 |     362 |    434 |      84.98 |         83.41
Obj_Ap          |    7 |       5 |      8 |      71.43 |         62.50
Obj_Co          |   48 |      14 |     54 |      29.17 |         25.93
Obj_Pa          |    2 |       0 |      0 |       0.00 |           NaN
Pnom            |   71 |      56 |     66 |      78.87 |         84.85
Pnom_Co         |    4 |       2 |      5 |      50.00 |         40.00
Pred            |  242 |     219 |    243 |      90.50 |         90.12
Pred_Co         |  113 |      80 |    105 |      70.80 |         76.19
Pred_Pa         |    7 |       3 |      5 |      42.86 |         60.00
Sb              |  360 |     326 |    346 |      90.56 |         94.22
Sb_Ap           |   14 |       1 |      4 |       7.14 |         25.00
Sb_Co           |   30 |      22 |     45 |      73.33 |         48.89


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
Adv             |  576 |     450 |    567 |      78.12 |         79.37
AdvAtr          |    6 |       0 |      0 |       0.00 |           NaN
Adv_Ap          |    6 |       0 |      3 |       0.00 |          0.00
Adv_Co          |   39 |      13 |     32 |      33.33 |         40.62
Adv_Pa          |    7 |       1 |      2 |      14.29 |         50.00
Apos            |    3 |       0 |      4 |       0.00 |          0.00
Apos_Co         |    1 |       1 |      1 |     100.00 |        100.00
Atr             | 1514 |    1397 |   1563 |      92.27 |         89.38
AtrAdv          |    9 |       0 |      0 |       0.00 |           NaN
AtrAtr          |    4 |       0 |      0 |       0.00 |           NaN
Atr_Ap          |    0 |       0 |      4 |        NaN |          0.00
Atr_Co          |  134 |      73 |    134 |      54.48 |         54.48
Atr_Pa          |    2 |       0 |      0 |       0.00 |           NaN
```

```
Atv             |   10 |       6 |      8 |      60.00 |         75.00
AtvV            |    2 |       0 |      1 |       0.00 |          0.00
Atv_Co          |    2 |       0 |      0 |       0.00 |           NaN
AuxC            |  101 |      92 |    103 |      91.09 |         89.32
AuxO            |    1 |       0 |      2 |       0.00 |          0.00
AuxP            |  610 |     483 |    614 |      79.18 |         78.66
AuxR            |   23 |      12 |     17 |      52.17 |         70.59
AuxT            |   63 |      57 |     74 |      90.48 |         77.03
AuxV            |   82 |      75 |     82 |      91.46 |         91.46
AuxY            |   52 |      37 |     44 |      71.15 |         84.09
AuxZ            |  106 |      79 |    108 |      74.53 |         73.15
Coord           |  157 |     106 |    161 |      67.52 |         65.84
Coord_Ap        |    1 |       0 |      2 |       0.00 |          0.00
Coord_Co        |   11 |       4 |     10 |      36.36 |         40.00
ExD             |   59 |      44 |     56 |      74.58 |         78.57
ExD_Ap          |   17 |       4 |      6 |      23.53 |         66.67
ExD_Co          |   64 |      24 |     74 |      37.50 |         32.43
ExD_Pa          |   14 |       9 |     13 |      64.29 |         69.23
Obj             |  426 |     353 |    434 |      82.86 |         81.34
Obj_Ap          |    7 |       5 |      8 |      71.43 |         62.50
Obj_Co          |   48 |      13 |     54 |      27.08 |         24.07
Obj_Pa          |    2 |       0 |      0 |       0.00 |           NaN
Pnom            |   71 |      55 |     66 |      77.46 |         83.33
Pnom_Co         |    4 |       2 |      5 |      50.00 |         40.00
Pred            |  242 |     218 |    243 |      90.08 |         89.71
Pred_Co         |  113 |      72 |    105 |      63.72 |         68.57
Pred_Pa         |    7 |       2 |      5 |      28.57 |         40.00
Sb              |  360 |     300 |    346 |      83.33 |         86.71
Sb_Ap           |   14 |       1 |      4 |       7.14 |         25.00
Sb_Co           |   30 |      21 |     45 |      70.00 |         46.67


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  354 |     310 |    349 |      87.57 |         88.83
left            | 2556 |    2422 |   2567 |      94.76 |         94.35
right           | 2090 |    1944 |   2084 |      93.01 |         93.28
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  354 |     310 |    349 |      87.57 |         88.83
1               | 2471 |    2364 |   2539 |      95.67 |         93.11
2               |  988 |     899 |   1010 |      90.99 |         89.01
3-6             |  921 |     752 |    864 |      81.65 |         87.04
7-...           |  266 |     186 |    238 |      69.92 |         78.15
```

C.5 Danish

```
Labeled   attachment score: 4248 / 5010 * 100 = 84.79 %
Unlabeled attachment score: 4538 / 5010 * 100 = 90.58 %
Label accuracy score:       4470 / 5010 * 100 = 89.22 %

================================================================================

Evaluation of the results in danish.nonproj.pred
vs. gold standard danish_ddt_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 842

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5010 |  4538 |  91% |  4470 |  89% |  4248 |  85%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1386 |  1291 |  93% |  1258 |  91% |  1234 |  89%
V          |   954 |   898 |  94% |   884 |  93% |   864 |  91%
P          |   624 |   599 |  96% |   581 |  93% |   576 |  92%
SP         |   602 |   477 |  79% |   455 |  76% |   388 |  64%
A          |   518 |   471 |  91% |   462 |  89% |   444 |  86%
RG         |   394 |   356 |  90% |   351 |  89% |   326 |  83%
C          |   324 |   267 |  82% |   306 |  94% |   257 |  79%
U          |   172 |   155 |  90% |   154 |  90% |   145 |  84%
X          |    32 |    20 |  62% |    18 |  56% |    13 |  41%
I          |     4 |     4 | 100% |     1 |  25% |     1 |  25%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5010 |   472 |   9% |   540 |  11% |   250 |   5%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1386 |    95 |   7% |   128 |   9% |    71 |   5%
V          |   954 |    56 |   6% |    70 |   7% |    36 |   4%
P          |   624 |    25 |   4% |    43 |   7% |    20 |   3%
SP         |   602 |   125 |  21% |   147 |  24% |    58 |  10%
A          |   518 |    47 |   9% |    56 |  11% |    29 |   6%
RG         |   394 |    38 |  10% |    43 |  11% |    13 |   3%
C          |   324 |    57 |  18% |    18 |   6% |     8 |   2%
U          |   172 |    17 |  10% |    18 |  10% |     8 |   5%
X          |    32 |    12 |  38% |    14 |  44% |     7 |  22%
```

```
I          |     4 |     0 |   0% |     3 |  75% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------



Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
<dobj>          |    2 |       0 |      0 |       0.00 |           NaN
<mod>           |    8 |       0 |      1 |       0.00 |          0.00
<pred>          |    2 |       0 |      1 |       0.00 |          0.00
<subj:pobj>     |    1 |       0 |      0 |       0.00 |           NaN
<subj>          |    2 |       0 |      0 |       0.00 |           NaN
<xpl>           |    0 |       0 |      1 |        NaN |          0.00
ROOT            |  323 |     307 |    322 |      95.05 |         95.34
aobj            |    6 |       0 |      2 |       0.00 |          0.00
appa            |   15 |       8 |     14 |      53.33 |         57.14
appr            |   13 |       5 |      8 |      38.46 |         62.50
avobj           |   15 |       5 |      5 |      33.33 |        100.00
conj            |  239 |     208 |    242 |      87.03 |         85.95
coord           |  156 |     155 |    158 |      99.36 |         98.10
dobj            |  321 |     294 |    350 |      91.59 |         84.00
err             |    1 |       0 |      3 |       0.00 |          0.00
expl            |   23 |      20 |     22 |      86.96 |         90.91
iobj            |   10 |       5 |      6 |      50.00 |         83.33
list            |   13 |       2 |      8 |      15.38 |         25.00
lobj            |   63 |      44 |     59 |      69.84 |         74.58
mod             | 1028 |     885 |   1003 |      86.09 |         88.24
modp            |    2 |       0 |      4 |       0.00 |          0.00
name            |    1 |       0 |      5 |       0.00 |          0.00
namef           |   90 |      84 |     92 |      93.33 |         91.30
namel           |    5 |       2 |      5 |      40.00 |         40.00
nobj            |  989 |     961 |   1005 |      97.17 |         95.62
numm            |    2 |       1 |      1 |      50.00 |        100.00
obl             |    1 |       1 |      3 |     100.00 |         33.33
part            |    8 |       5 |      7 |      62.50 |         71.43
pobj            |  277 |     230 |    310 |      83.03 |         74.19
possd           |  112 |     102 |    109 |      91.07 |         93.58
pred            |  151 |     113 |    142 |      74.83 |         79.58
qobj            |   37 |      30 |     33 |      81.08 |         90.91
rel             |   68 |      62 |     76 |      91.18 |         81.58
subj            |  558 |     514 |    558 |      92.11 |         92.11
title           |    9 |       8 |     11 |      88.89 |         72.73
tobj            |   14 |       4 |      5 |      28.57 |         80.00
vobj            |  424 |     410 |    427 |      96.70 |         96.02
voc             |    2 |       0 |      1 |       0.00 |          0.00
xpl             |   14 |       3 |      8 |      21.43 |         37.50
xtop            |    5 |       2 |      3 |      40.00 |         66.67


Precision and recall of DEPREL + ATTACHMENT
```

```
----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
<dobj>          |    2 |       0 |      0 |       0.00 |            NaN
<mod>           |    8 |       0 |      1 |       0.00 |           0.00
<pred>          |    2 |       0 |      1 |       0.00 |           0.00
<subj:pobj>     |    1 |       0 |      0 |       0.00 |            NaN
<subj>          |    2 |       0 |      0 |       0.00 |            NaN
<xpl>           |    0 |       0 |      1 |        NaN |           0.00
ROOT            |  323 |     307 |    322 |      95.05 |          95.34
aobj            |    6 |       0 |      2 |       0.00 |           0.00
appa            |   15 |       8 |     14 |      53.33 |          57.14
appr            |   13 |       5 |      8 |      38.46 |          62.50
avobj           |   15 |       5 |      5 |      33.33 |         100.00
conj            |  239 |     201 |    242 |      84.10 |          83.06
coord           |  156 |     114 |    158 |      73.08 |          72.15
dobj            |  321 |     287 |    350 |      89.41 |          82.00
err             |    1 |       0 |      3 |       0.00 |           0.00
expl            |   23 |      20 |     22 |      86.96 |          90.91
iobj            |   10 |       5 |      6 |      50.00 |          83.33
list            |   13 |       2 |      8 |      15.38 |          25.00
lobj            |   63 |      44 |     59 |      69.84 |          74.58
mod             | 1028 |     768 |   1003 |      74.71 |          76.57
modp            |    2 |       0 |      4 |       0.00 |           0.00
name            |    1 |       0 |      5 |       0.00 |           0.00
namef           |   90 |      83 |     92 |      92.22 |          90.22
namel           |    5 |       2 |      5 |      40.00 |          40.00
nobj            |  989 |     950 |   1005 |      96.06 |          94.53
numm            |    2 |       1 |      1 |      50.00 |         100.00
obl             |    1 |       1 |      3 |     100.00 |          33.33
part            |    8 |       5 |      7 |      62.50 |          71.43
pobj            |  277 |     216 |    310 |      77.98 |          69.68
possd           |  112 |      95 |    109 |      84.82 |          87.16
pred            |  151 |     112 |    142 |      74.17 |          78.87
qobj            |   37 |      30 |     33 |      81.08 |          90.91
rel             |   68 |      51 |     76 |      75.00 |          67.11
subj            |  558 |     513 |    558 |      91.94 |          91.94
title           |    9 |       8 |     11 |      88.89 |          72.73
tobj            |   14 |       4 |      5 |      28.57 |          80.00
vobj            |  424 |     406 |    427 |      95.75 |          95.08
voc             |    2 |       0 |      1 |       0.00 |           0.00
xpl             |   14 |       3 |      8 |      21.43 |          37.50
xtop            |    5 |       2 |      3 |      40.00 |          66.67


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  323 |     307 |    322 |      95.05 |          95.34
left            | 3707 |    3649 |   3721 |      98.44 |          98.07
right           |  980 |     910 |    967 |      92.86 |          94.11
```

```
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  323 |     307 |    322 |      95.05 |         95.34
1               | 2893 |    2807 |   2958 |      97.03 |         94.90
2               |  858 |     777 |    846 |      90.56 |         91.84
3-6             |  705 |     575 |    674 |      81.56 |         85.31
7-...           |  231 |     168 |    210 |      72.73 |         80.00
```

C.6 Dutch

```
Labeled   attachment score: 3958 / 4998 * 100 = 79.19 %
Unlabeled attachment score: 4177 / 4998 * 100 = 83.57 %
Label accuracy score:       4193 / 4998 * 100 = 83.89 %

================================================================================

Evaluation of the results in dutch.nonproj.pred
vs. gold standard dutch_alpino_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 587

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4998 |  4177 |  84% |  4193 |  84% |  3958 |  79%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1374 |  1143 |  83% |  1105 |  80% |  1074 |  78%
V          |   849 |   738 |  87% |   727 |  86% |   720 |  85%
Prep       |   674 |   543 |  81% |   563 |  84% |   487 |  72%
Art        |   615 |   583 |  95% |   608 |  99% |   582 |  95%
Adj        |   350 |   326 |  93% |   326 |  93% |   317 |  91%
Conj       |   306 |   178 |  58% |   182 |  59% |   158 |  52%
Adv        |   287 |   233 |  81% |   252 |  88% |   219 |  76%
Pron       |   256 |   231 |  90% |   217 |  85% |   209 |  82%
MWU        |   141 |   104 |  74% |   121 |  86% |   103 |  73%
Num        |   129 |    83 |  64% |    78 |  60% |    75 |  58%
Punc       |    12 |    12 | 100% |    12 | 100% |    12 | 100%
Misc       |     4 |     3 |  75% |     2 |  50% |     2 |  50%
Int        |     1 |     0 |   0% |     0 |   0% |     0 |   0%
```

```
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4998 |   821 |  16% |   805 |  16% |   586 |  12%
-----------+-------+-------+------+-------+------+-------+-------
N          |  1374 |   231 |  17% |   269 |  20% |   200 |  15%
V          |   849 |   111 |  13% |   122 |  14% |   104 |  12%
Prep       |   674 |   131 |  19% |   111 |  16% |    55 |   8%
Art        |   615 |    32 |   5% |     7 |   1% |     6 |   1%
Adj        |   350 |    24 |   7% |    24 |   7% |    15 |   4%
Conj       |   306 |   128 |  42% |   124 |  41% |   104 |  34%
Adv        |   287 |    54 |  19% |    35 |  12% |    21 |   7%
Pron       |   256 |    25 |  10% |    39 |  15% |    17 |   7%
MWU        |   141 |    37 |  26% |    20 |  14% |    19 |  13%
Num        |   129 |    46 |  36% |    51 |  40% |    43 |  33%
Punc       |    12 |     0 |   0% |     0 |   0% |     0 |   0%
Misc       |     4 |     1 |  25% |     2 |  50% |     1 |  25%
Int        |     1 |     1 | 100% |     1 | 100% |     1 | 100%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ROOT            |  514 |     366 |    424 |      71.21 |         86.32
app             |   75 |      28 |     77 |      37.33 |         36.36
body            |  153 |     124 |    150 |      81.05 |         82.67
cnj             |  535 |     427 |    526 |      79.81 |         81.18
crd             |    4 |       2 |      6 |      50.00 |         33.33
det             |  761 |     743 |    778 |      97.63 |         95.50
hdf             |    3 |       3 |      4 |     100.00 |         75.00
ld              |   23 |      12 |     22 |      52.17 |         54.55
me              |    3 |       1 |      2 |      33.33 |         50.00
mod             | 1255 |    1137 |   1339 |      90.60 |         84.91
obcomp          |    9 |       8 |      8 |      88.89 |        100.00
obj1            |  823 |     716 |    864 |      87.00 |         82.87
obj2            |    8 |       2 |      5 |      25.00 |         40.00
pc              |  101 |      48 |     68 |      47.52 |         70.59
pobj1           |    2 |       2 |      3 |     100.00 |         66.67
predc           |   91 |      56 |     78 |      61.54 |         71.79
predm           |    8 |       2 |      6 |      25.00 |         33.33
punct           |   17 |      12 |     12 |      70.59 |        100.00
se              |    3 |       3 |      4 |     100.00 |         75.00
su              |  306 |     240 |    303 |      78.43 |         79.21
sup             |    4 |       2 |      4 |      50.00 |         50.00
```

```
svp             |   42 |      32 |     37 |      76.19 |         86.49
vc              |  258 |     227 |    278 |      87.98 |         81.65


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ROOT            |  514 |     366 |    424 |      71.21 |         86.32
app             |   75 |      27 |     77 |      36.00 |         35.06
body            |  153 |     123 |    150 |      80.39 |         82.00
cnj             |  535 |     413 |    526 |      77.20 |         78.52
crd             |    4 |       2 |      6 |      50.00 |         33.33
det             |  761 |     717 |    778 |      94.22 |         92.16
hdf             |    3 |       3 |      4 |     100.00 |         75.00
ld              |   23 |      12 |     22 |      52.17 |         54.55
me              |    3 |       1 |      2 |      33.33 |         50.00
mod             | 1255 |     981 |   1339 |      78.17 |         73.26
obcomp          |    9 |       7 |      8 |      77.78 |         87.50
obj1            |  823 |     699 |    864 |      84.93 |         80.90
obj2            |    8 |       2 |      5 |      25.00 |         40.00
pc              |  101 |      45 |     68 |      44.55 |         66.18
pobj1           |    2 |       2 |      3 |     100.00 |         66.67
predc           |   91 |      55 |     78 |      60.44 |         70.51
predm           |    8 |       1 |      6 |      12.50 |         16.67
punct           |   17 |      12 |     12 |      70.59 |        100.00
se              |    3 |       3 |      4 |     100.00 |         75.00
su              |  306 |     230 |    303 |      75.16 |         75.91
sup             |    4 |       2 |      4 |      50.00 |         50.00
svp             |   42 |      32 |     37 |      76.19 |         86.49
vc              |  258 |     223 |    278 |      86.43 |         80.22


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  514 |     366 |    424 |      71.21 |         86.32
left            | 2336 |    2163 |   2411 |      92.59 |         89.71
right           | 2148 |    1974 |   2163 |      91.90 |         91.26
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  514 |     366 |    424 |      71.21 |         86.32
1               | 2366 |    2231 |   2445 |      94.29 |         91.25
2               |  903 |     792 |    922 |      87.71 |         85.90
```

```
3-6            |  888 |     701 |    895 |      78.94 |         78.32
7-...          |  327 |     226 |    312 |      69.11 |         72.44
```

C.7 English

```
Labeled   attachment score: 44552 / 49847 * 100 = 89.38 %
Unlabeled attachment score: 45578 / 49847 * 100 = 91.44 %
Label accuracy score:       46984 / 49847 * 100 = 94.26 %

================================================================================

Evaluation of the results in /scratch/ryantm/mine/parsers/old_parsers/conll_lab/eng_out/new.out
vs. gold standard ../data/english/ptb/test/conll_english.txt:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 6837

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      | 49847 | 45578 |  91% | 46984 |  94% | 44552 |  89%
-----------+-------+-------+------+-------+------+-------+-------
NN         |  7536 |  6974 |  93% |  6995 |  93% |  6790 |  90%
IN         |  5934 |  5039 |  85% |  5645 |  95% |  4884 |  82%
NNP        |  5500 |  5215 |  95% |  5270 |  96% |  5126 |  93%
DT         |  4834 |  4692 |  97% |  4790 |  99% |  4672 |  97%
JJ         |  3663 |  3439 |  94% |  3424 |  93% |  3322 |  91%
NNS        |  3561 |  3264 |  92% |  3293 |  92% |  3193 |  90%
RB         |  1991 |  1661 |  83% |  1668 |  84% |  1517 |  76%
CD         |  1943 |  1792 |  92% |  1862 |  96% |  1776 |  91%
VBD        |  1814 |  1695 |  93% |  1722 |  95% |  1689 |  93%
VB         |  1549 |  1446 |  93% |  1454 |  94% |  1412 |  91%
CC         |  1291 |  1092 |  85% |  1284 |  99% |  1091 |  85%
TO         |  1240 |  1151 |  93% |  1209 |  98% |  1147 |  92%
VBZ        |  1239 |  1126 |  91% |  1140 |  92% |  1110 |  90%
VBN        |  1190 |  1092 |  92% |  1086 |  91% |  1046 |  88%
PRP        |  1050 |  1032 |  98% |  1035 |  99% |  1027 |  98%
VBG        |   856 |   722 |  84% |   749 |  88% |   693 |  81%
VBP        |   811 |   727 |  90% |   731 |  90% |   710 |  88%
MD         |   583 |   530 |  91% |   547 |  94% |   525 |  90%
''         |   531 |   426 |  80% |   531 | 100% |   426 |  80%
PRP$       |   511 |   494 |  97% |   511 | 100% |   494 |  97%
POS        |   504 |   484 |  96% |   496 |  98% |   483 |  96%
$          |   376 |   342 |  91% |   343 |  91% |   336 |  89%
WDT        |   276 |   234 |  85% |   268 |  97% |   233 |  84%
JJR        |   190 |   151 |  79% |   147 |  77% |   138 |  73%
```

```
WRB        |   132 |    96 |  73% |   123 |  93% |    93 |  70%
RP         |   130 |   129 |  99% |   105 |  81% |   105 |  81%
JJS        |   128 |   118 |  92% |   115 |  90% |   112 |  88%
NNPS       |   118 |   112 |  95% |   112 |  95% |   111 |  94%
WP         |   112 |    95 |  85% |   105 |  94% |    93 |  83%
RBR        |   107 |    85 |  79% |    86 |  80% |    76 |  71%
EX         |    58 |    56 |  97% |    58 | 100% |    56 |  97%
RBS        |    27 |    24 |  89% |    26 |  96% |    24 |  89%
PDT        |    21 |    17 |  81% |    21 | 100% |    17 |  81%
WP$        |    21 |    17 |  81% |    20 |  95% |    17 |  81%
UH         |    10 |     5 |  50% |     8 |  80% |     5 |  50%
FW         |     4 |     2 |  50% |     1 |  25% |     1 |  25%
LS         |     4 |     2 |  50% |     4 | 100% |     2 |  50%
,          |     2 |     0 |   0% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      | 49847 |  4269 |   9% |  2863 |   6% |  1837 |   4%
-----------+-------+-------+------+-------+------+-------+-------
NN         |  7536 |   562 |   7% |   541 |   7% |   357 |   5%
IN         |  5934 |   895 |  15% |   289 |   5% |   134 |   2%
NNP        |  5500 |   285 |   5% |   230 |   4% |   141 |   3%
DT         |  4834 |   142 |   3% |    44 |   1% |    24 |   0%
JJ         |  3663 |   224 |   6% |   239 |   7% |   122 |   3%
NNS        |  3561 |   297 |   8% |   268 |   8% |   197 |   6%
RB         |  1991 |   330 |  17% |   323 |  16% |   179 |   9%
CD         |  1943 |   151 |   8% |    81 |   4% |    65 |   3%
VBD        |  1814 |   119 |   7% |    92 |   5% |    86 |   5%
VB         |  1549 |   103 |   7% |    95 |   6% |    61 |   4%
CC         |  1291 |   199 |  15% |     7 |   1% |     6 |   0%
TO         |  1240 |    89 |   7% |    31 |   2% |    27 |   2%
VBZ        |  1239 |   113 |   9% |    99 |   8% |    83 |   7%
VBN        |  1190 |    98 |   8% |   104 |   9% |    58 |   5%
PRP        |  1050 |    18 |   2% |    15 |   1% |    10 |   1%
VBG        |   856 |   134 |  16% |   107 |  12% |    78 |   9%
VBP        |   811 |    84 |  10% |    80 |  10% |    63 |   8%
MD         |   583 |    53 |   9% |    36 |   6% |    31 |   5%
‘‘         |   531 |   105 |  20% |     0 |   0% |     0 |   0%
PRP$       |   511 |    17 |   3% |     0 |   0% |     0 |   0%
POS        |   504 |    20 |   4% |     8 |   2% |     7 |   1%
$          |   376 |    34 |   9% |    33 |   9% |    27 |   7%
WDT        |   276 |    42 |  15% |     8 |   3% |     7 |   3%
JJR        |   190 |    39 |  21% |    43 |  23% |    30 |  16%
WRB        |   132 |    36 |  27% |     9 |   7% |     6 |   5%
RP         |   130 |     1 |   1% |    25 |  19% |     1 |   1%
JJS        |   128 |    10 |   8% |    13 |  10% |     7 |   5%
NNPS       |   118 |     6 |   5% |     6 |   5% |     5 |   4%
```

```
WP         |   112 |    17 |  15% |     7 |   6% |     5 |   4%
RBR        |   107 |    22 |  21% |    21 |  20% |    12 |  11%
EX         |    58 |     2 |   3% |     0 |   0% |     0 |   0%
RBS        |    27 |     3 |  11% |     1 |   4% |     1 |   4%
PDT        |    21 |     4 |  19% |     0 |   0% |     0 |   0%
WP$        |    21 |     4 |  19% |     1 |   5% |     1 |   5%
UH         |    10 |     5 |  50% |     2 |  20% |     2 |  20%
FW         |     4 |     2 |  50% |     3 |  75% |     2 |  50%
LS         |     4 |     2 |  50% |     0 |   0% |     0 |   0%
,          |     2 |     2 | 100% |     2 | 100% |     2 | 100%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ADJP            |  732 |     519 |    636 |      70.90 |         81.60
ADVP            | 1166 |    1003 |   1178 |      86.02 |         85.14
CONJP           |   21 |       7 |     12 |      33.33 |         58.33
DEP             | 19010 |   18572 |  19178 |      97.70 |         96.84
FRAG            |   19 |       2 |     11 |      10.53 |         18.18
INTJ            |   10 |       8 |     11 |      80.00 |         72.73
LST             |    4 |       4 |      5 |     100.00 |         80.00
NAC             |   30 |      18 |     22 |      60.00 |         81.82
NP              | 7225 |    6779 |   7196 |      93.83 |         94.21
NP-OBJ          | 1974 |    1768 |   2006 |      89.56 |         88.14
NP-PRD          |  344 |     290 |    365 |      84.30 |         79.45
NP-SBJ          | 4097 |    3848 |   4039 |      93.92 |         95.27
NX              |   44 |       4 |      9 |       9.09 |         44.44
PP              | 5429 |    5318 |   5532 |      97.96 |         96.13
PRN             |  140 |      90 |    119 |      64.29 |         75.63
PRT             |  159 |     105 |    133 |      66.04 |         78.95
QP              |  187 |     156 |    185 |      83.42 |         84.32
ROOT            | 2410 |    2282 |   2411 |      94.69 |         94.65
RRC             |    0 |       0 |      3 |        NaN |          0.00
S               | 2773 |    2523 |   2806 |      90.98 |         89.91
SBAR            | 1757 |    1537 |   1685 |      87.48 |         91.22
SBARQ           |    2 |       0 |      0 |       0.00 |           NaN
SINV            |   11 |       2 |      3 |      18.18 |         66.67
SQ              |    1 |       0 |      2 |       0.00 |          0.00
UCP             |   28 |       3 |     13 |      10.71 |         23.08
VP              | 2231 |    2115 |   2248 |      94.80 |         94.08
WHADJP          |    0 |       0 |      2 |        NaN |          0.00
WHADVP          |    8 |       5 |      6 |      62.50 |         83.33
WHNP            |   30 |      25 |     28 |      83.33 |         89.29
WHPP            |    0 |       0 |      2 |        NaN |          0.00
X               |    5 |       1 |      1 |      20.00 |        100.00


Precision and recall of DEPREL + ATTACHMENT
```

```
----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ADJP            |  732 |     498 |    636 |      68.03 |         78.30
ADVP            | 1166 |     900 |   1178 |      77.19 |         76.40
CONJP           |   21 |       5 |     12 |      23.81 |         41.67
DEP             | 19010 |   17803 |  19178 |      93.65 |         92.83
FRAG            |   19 |       2 |     11 |      10.53 |         18.18
INTJ            |   10 |       5 |     11 |      50.00 |         45.45
LST             |    4 |       2 |      5 |      50.00 |         40.00
NAC             |   30 |      18 |     22 |      60.00 |         81.82
NP              | 7225 |    6447 |   7196 |      89.23 |         89.59
NP-OBJ          | 1974 |    1744 |   2006 |      88.35 |         86.94
NP-PRD          |  344 |     290 |    365 |      84.30 |         79.45
NP-SBJ          | 4097 |    3811 |   4039 |      93.02 |         94.36
NX              |   44 |       4 |      9 |       9.09 |         44.44
PP              | 5429 |    4587 |   5532 |      84.49 |         82.92
PRN             |  140 |      73 |    119 |      52.14 |         61.34
PRT             |  159 |     105 |    133 |      66.04 |         78.95
QP              |  187 |     154 |    185 |      82.35 |         83.24
ROOT            | 2410 |    2282 |   2411 |      94.69 |         94.65
RRC             |    0 |       0 |      3 |        NaN |          0.00
S               | 2773 |    2386 |   2806 |      86.04 |         85.03
SBAR            | 1757 |    1333 |   1685 |      75.87 |         79.11
SBARQ           |    2 |       0 |      0 |       0.00 |           NaN
SINV            |   11 |       1 |      3 |       9.09 |         33.33
SQ              |    1 |       0 |      2 |       0.00 |          0.00
UCP             |   28 |       3 |     13 |      10.71 |         23.08
VP              | 2231 |    2068 |   2248 |      92.69 |         91.99
WHADJP          |    0 |       0 |      2 |        NaN |          0.00
WHADVP          |    8 |       5 |      6 |      62.50 |         83.33
WHNP            |   30 |      25 |     28 |      83.33 |         89.29
WHPP            |    0 |       0 |      2 |        NaN |          0.00
X               |    5 |       1 |      1 |      20.00 |        100.00


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         | 2410 |    2282 |   2409 |      94.69 |         94.73
left            | 22207 |   21394 |  22194 |      96.34 |         96.40
right           | 25230 |   24436 |  25244 |      96.85 |         96.80
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         | 2410 |    2282 |   2409 |      94.69 |         94.73
```

```
1               | 24338 |   23454 |  24425 |      96.37 |         96.02
2               | 10379 |    9716 |  10475 |      93.61 |         92.75
3-6             | 9224 |    8210 |   9198 |      89.01 |         89.26
7-...           | 3496 |    2863 |   3340 |      81.89 |         85.72
```

C.8 German

```
Labeled   attachment score: 4374 / 5008 * 100 = 87.34 %
Unlabeled attachment score: 4526 / 5008 * 100 = 90.38 %
Label accuracy score:       4613 / 5008 * 100 = 92.11 %

================================================================================

Evaluation of the results in german.nonproj.pred
vs. gold standard german_tiger_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 686

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5008 |  4526 |  90% |  4613 |  92% |  4374 |  87%
-----------+-------+-------+------+-------+------+-------+-------
NN         |  1201 |  1093 |  91% |  1067 |  89% |  1041 |  87%
ART        |   593 |   584 |  98% |   592 | 100% |   583 |  98%
APPR       |   463 |   368 |  79% |   375 |  81% |   340 |  73%
NE         |   343 |   323 |  94% |   318 |  93% |   311 |  91%
ADJA       |   321 |   311 |  97% |   318 |  99% |   310 |  97%
ADV        |   274 |   212 |  77% |   264 |  96% |   209 |  76%
VVFIN      |   227 |   212 |  93% |   210 |  93% |   207 |  91%
VAFIN      |   157 |   142 |  90% |   142 |  90% |   137 |  87%
KON        |   141 |   114 |  81% |   139 |  99% |   113 |  80%
$(         |   123 |   106 |  86% |   122 |  99% |   106 |  86%
ADJD       |   116 |   103 |  89% |   108 |  93% |   101 |  87%
APPRART    |   109 |    82 |  75% |    88 |  81% |    78 |  72%
VVPP       |   103 |    99 |  96% |    95 |  92% |    94 |  91%
PPER       |    90 |    88 |  98% |    82 |  91% |    80 |  89%
VVINF      |    89 |    82 |  92% |    79 |  89% |    77 |  87%
CARD       |    86 |    77 |  90% |    80 |  93% |    76 |  88%
VMFIN      |    62 |    56 |  90% |    57 |  92% |    55 |  89%
PPOSAT     |    49 |    48 |  98% |    48 |  98% |    48 |  98%
KOUS       |    48 |    47 |  98% |    47 |  98% |    47 |  98%
PTKNEG     |    45 |    33 |  73% |    44 |  98% |    33 |  73%
PIAT       |    40 |    39 |  98% |    39 |  98% |    39 |  98%
PRF        |    37 |    36 |  97% |    35 |  95% |    34 |  92%
```

```
PRELS      |    34 |    33 |  97% |    33 |  97% |    32 |  94%
PTKVZ      |    31 |    31 | 100% |    31 | 100% |    31 | 100%
PIS        |    31 |    26 |  84% |    23 |  74% |    23 |  74%
PROAV      |    29 |    26 |  90% |    27 |  93% |    24 |  83%
PTKZU      |    26 |    26 | 100% |    26 | 100% |    26 | 100%
PDS        |    24 |    21 |  88% |    20 |  83% |    18 |  75%
VAINF      |    16 |    16 | 100% |    16 | 100% |    16 | 100%
PDAT       |    16 |    16 | 100% |    16 | 100% |    16 | 100%
VVIZU      |    11 |     9 |  82% |     7 |  64% |     7 |  64%
PWAV       |    11 |    10 |  91% |    11 | 100% |    10 |  91%
FM         |     9 |     8 |  89% |     4 |  44% |     4 |  44%
KOKOM      |     9 |     8 |  89% |     9 | 100% |     8 |  89%
PWS        |     8 |     5 |  62% |     6 |  75% |     5 |  62%
VAPP       |     7 |     7 | 100% |     7 | 100% |     7 | 100%
TRUNC      |     5 |     5 | 100% |     4 |  80% |     4 |  80%
KOUI       |     5 |     5 | 100% |     5 | 100% |     5 | 100%
XY         |     4 |     4 | 100% |     4 | 100% |     4 | 100%
PTKANT     |     3 |     3 | 100% |     3 | 100% |     3 | 100%
VVIMP      |     2 |     2 | 100% |     2 | 100% |     2 | 100%
PTKA       |     2 |     2 | 100% |     2 | 100% |     2 | 100%
VMINF      |     2 |     2 | 100% |     2 | 100% |     2 | 100%
APPO       |     2 |     2 | 100% |     2 | 100% |     2 | 100%
APZR       |     1 |     1 | 100% |     1 | 100% |     1 | 100%
PRELAT     |     1 |     1 | 100% |     1 | 100% |     1 | 100%
NNE        |     1 |     1 | 100% |     1 | 100% |     1 | 100%
VAIMP      |     1 |     1 | 100% |     1 | 100% |     1 | 100%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5008 |   482 |  10% |   395 |   8% |   243 |   5%
-----------+-------+-------+------+-------+------+-------+-------
NN         |  1201 |   108 |   9% |   134 |  11% |    82 |   7%
ART        |   593 |     9 |   2% |     1 |   0% |     0 |   0%
APPR       |   463 |    95 |  21% |    88 |  19% |    60 |  13%
NE         |   343 |    20 |   6% |    25 |   7% |    13 |   4%
ADJA       |   321 |    10 |   3% |     3 |   1% |     2 |   1%
ADV        |   274 |    62 |  23% |    10 |   4% |     7 |   3%
VVFIN      |   227 |    15 |   7% |    17 |   7% |    12 |   5%
VAFIN      |   157 |    15 |  10% |    15 |  10% |    10 |   6%
KON        |   141 |    27 |  19% |     2 |   1% |     1 |   1%
$(         |   123 |    17 |  14% |     1 |   1% |     1 |   1%
ADJD       |   116 |    13 |  11% |     8 |   7% |     6 |   5%
APPRART    |   109 |    27 |  25% |    21 |  19% |    17 |  16%
VVPP       |   103 |     4 |   4% |     8 |   8% |     3 |   3%
PPER       |    90 |     2 |   2% |     8 |   9% |     0 |   0%
VVINF      |    89 |     7 |   8% |    10 |  11% |     5 |   6%
CARD       |    86 |     9 |  10% |     6 |   7% |     5 |   6%
```

```
VMFIN      |    62 |     6 |  10% |     5 |   8% |     4 |   6%
PPOSAT     |    49 |     1 |   2% |     1 |   2% |     1 |   2%
KOUS       |    48 |     1 |   2% |     1 |   2% |     1 |   2%
PTKNEG     |    45 |    12 |  27% |     1 |   2% |     1 |   2%
PIAT       |    40 |     1 |   2% |     1 |   2% |     1 |   2%
PRF        |    37 |     1 |   3% |     2 |   5% |     0 |   0%
PRELS      |    34 |     1 |   3% |     1 |   3% |     0 |   0%
PTKVZ      |    31 |     0 |   0% |     0 |   0% |     0 |   0%
PIS        |    31 |     5 |  16% |     8 |  26% |     5 |  16%
PROAV      |    29 |     3 |  10% |     2 |   7% |     0 |   0%
PTKZU      |    26 |     0 |   0% |     0 |   0% |     0 |   0%
PDS        |    24 |     3 |  12% |     4 |  17% |     1 |   4%
VAINF      |    16 |     0 |   0% |     0 |   0% |     0 |   0%
PDAT       |    16 |     0 |   0% |     0 |   0% |     0 |   0%
VVIZU      |    11 |     2 |  18% |     4 |  36% |     2 |  18%
PWAV       |    11 |     1 |   9% |     0 |   0% |     0 |   0%
FM         |     9 |     1 |  11% |     5 |  56% |     1 |  11%
KOKOM      |     9 |     1 |  11% |     0 |   0% |     0 |   0%
PWS        |     8 |     3 |  38% |     2 |  25% |     2 |  25%
VAPP       |     7 |     0 |   0% |     0 |   0% |     0 |   0%
TRUNC      |     5 |     0 |   0% |     1 |  20% |     0 |   0%
KOUI       |     5 |     0 |   0% |     0 |   0% |     0 |   0%
XY         |     4 |     0 |   0% |     0 |   0% |     0 |   0%
PTKANT     |     3 |     0 |   0% |     0 |   0% |     0 |   0%
VVIMP      |     2 |     0 |   0% |     0 |   0% |     0 |   0%
PTKA       |     2 |     0 |   0% |     0 |   0% |     0 |   0%
VMINF      |     2 |     0 |   0% |     0 |   0% |     0 |   0%
APPO       |     2 |     0 |   0% |     0 |   0% |     0 |   0%
APZR       |     1 |     0 |   0% |     0 |   0% |     0 |   0%
PRELAT     |     1 |     0 |   0% |     0 |   0% |     0 |   0%
NNE        |     1 |     0 |   0% |     0 |   0% |     0 |   0%
VAIMP      |     1 |     0 |   0% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
AC              |    4 |       4 |      4 |     100.00 |        100.00
AG              |  150 |     129 |    152 |      86.00 |         84.87
AMS             |    0 |       0 |      1 |        NaN |          0.00
APP             |   21 |      11 |     20 |      52.38 |         55.00
CC              |    6 |       4 |      8 |      66.67 |         50.00
CD              |  129 |     129 |    132 |     100.00 |         97.73
CJ              |  172 |     148 |    174 |      86.05 |         85.06
CM              |    9 |       9 |      9 |     100.00 |        100.00
CP              |   52 |      52 |     53 |     100.00 |         98.11
CVC             |    2 |       1 |      2 |      50.00 |         50.00
DA              |   22 |      11 |     17 |      50.00 |         64.71
DH              |    3 |       2 |      3 |      66.67 |         66.67
DM              |    2 |       2 |      2 |     100.00 |        100.00
```

```
EP              |   11 |       9 |     11 |      81.82 |         81.82
JU              |   12 |      10 |     10 |      83.33 |        100.00
MNR             |  153 |     114 |    160 |      74.51 |         71.25
MO              |  772 |     690 |    755 |      89.38 |         91.39
NG              |   44 |      43 |     44 |      97.73 |         97.73
NK              | 1721 |    1692 |   1724 |      98.31 |         98.14
NMC             |   24 |      23 |     24 |      95.83 |         95.83
OA              |  206 |     175 |    220 |      84.95 |         79.55
OA2             |    0 |       0 |      1 |        NaN |          0.00
OC              |  220 |     210 |    236 |      95.45 |         88.98
OP              |   46 |      22 |     26 |      47.83 |         84.62
PAR             |   10 |       4 |      4 |      40.00 |        100.00
PD              |   62 |      46 |     61 |      74.19 |         75.41
PG              |   22 |      21 |     29 |      95.45 |         72.41
PH              |    1 |       0 |      0 |       0.00 |           NaN
PM              |   27 |      27 |     27 |     100.00 |        100.00
PNC             |   67 |      65 |     67 |      97.01 |         97.01
PUNC            |  122 |     121 |    121 |      99.18 |        100.00
RC              |   36 |      36 |     39 |     100.00 |         92.31
RE              |   13 |       7 |      9 |      53.85 |         77.78
ROOT            |  357 |     346 |    357 |      96.92 |         96.92
RS              |    2 |       1 |      1 |      50.00 |        100.00
SB              |  425 |     376 |    423 |      88.47 |         88.89
SBP             |   10 |       7 |     10 |      70.00 |         70.00
SVP             |   31 |      31 |     31 |     100.00 |        100.00
UC              |    4 |       2 |      5 |      50.00 |         40.00
VO              |    1 |       1 |      1 |     100.00 |        100.00
_               |   37 |      32 |     35 |      86.49 |         91.43


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
AC              |    4 |       4 |      4 |     100.00 |        100.00
AG              |  150 |     123 |    152 |      82.00 |         80.92
AMS             |    0 |       0 |      1 |        NaN |          0.00
APP             |   21 |       8 |     20 |      38.10 |         40.00
CC              |    6 |       4 |      8 |      66.67 |         50.00
CD              |  129 |     103 |    132 |      79.84 |         78.03
CJ              |  172 |     128 |    174 |      74.42 |         73.56
CM              |    9 |       8 |      9 |      88.89 |         88.89
CP              |   52 |      52 |     53 |     100.00 |         98.11
CVC             |    2 |       1 |      2 |      50.00 |         50.00
DA              |   22 |       9 |     17 |      40.91 |         52.94
DH              |    3 |       2 |      3 |      66.67 |         66.67
DM              |    2 |       2 |      2 |     100.00 |        100.00
EP              |   11 |       9 |     11 |      81.82 |         81.82
JU              |   12 |      10 |     10 |      83.33 |        100.00
MNR             |  153 |      98 |    160 |      64.05 |         61.25
MO              |  772 |     598 |    755 |      77.46 |         79.21
NG              |   44 |      32 |     44 |      72.73 |         72.73
```

```
NK              | 1721 |    1666 |   1724 |      96.80 |         96.64
NMC             |   24 |      23 |     24 |      95.83 |         95.83
OA              |  206 |     170 |    220 |      82.52 |         77.27
OA2             |    0 |       0 |      1 |        NaN |          0.00
OC              |  220 |     206 |    236 |      93.64 |         87.29
OP              |   46 |      22 |     26 |      47.83 |         84.62
PAR             |   10 |       4 |      4 |      40.00 |        100.00
PD              |   62 |      45 |     61 |      72.58 |         73.77
PG              |   22 |      19 |     29 |      86.36 |         65.52
PH              |    1 |       0 |      0 |       0.00 |           NaN
PM              |   27 |      27 |     27 |     100.00 |        100.00
PNC             |   67 |      65 |     67 |      97.01 |         97.01
PUNC            |  122 |     105 |    121 |      86.07 |         86.78
RC              |   36 |      30 |     39 |      83.33 |         76.92
RE              |   13 |       7 |      9 |      53.85 |         77.78
ROOT            |  357 |     346 |    357 |      96.92 |         96.92
RS              |    2 |       1 |      1 |      50.00 |        100.00
SB              |  425 |     374 |    423 |      88.00 |         88.42
SBP             |   10 |       7 |     10 |      70.00 |         70.00
SVP             |   31 |      31 |     31 |     100.00 |        100.00
UC              |    4 |       2 |      5 |      50.00 |         40.00
VO              |    1 |       1 |      1 |     100.00 |        100.00
_               |   37 |      32 |     35 |      86.49 |         91.43


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  357 |     346 |    357 |      96.92 |         96.92
left            | 2526 |    2439 |   2557 |      96.56 |         95.39
right           | 2125 |    2006 |   2094 |      94.40 |         95.80
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  357 |     346 |    357 |      96.92 |         96.92
1               | 2122 |    2038 |   2137 |      96.04 |         95.37
2               |  895 |     823 |    888 |      91.96 |         92.68
3-6             | 1055 |     948 |   1069 |      89.86 |         88.68
7-...           |  579 |     501 |    557 |      86.53 |         89.95
```

C.9 Japanese

```
Labeled   attachment score: 4538 / 5003 * 100 = 90.71 %
```

```
Unlabeled attachment score: 4645 / 5003 * 100 = 92.84 %
Label accuracy score:       4690 / 5003 * 100 = 93.74 %

================================================================================

Evaluation of the results in japanese.nonproj.pred
vs. gold standard japanese_verbmobil_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 708

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5003 |  4645 |  93% |  4690 |  94% |  4538 |  91%
-----------+-------+-------+------+-------+------+-------+-------
P          |  1045 |   958 |  92% |   965 |  92% |   912 |  87%
N          |  1043 |   971 |  93% |   968 |  93% |   939 |  90%
PS         |   505 |   443 |  88% |   460 |  91% |   442 |  88%
V          |   407 |   377 |  93% |   381 |  94% |   374 |  92%
PV         |   353 |   345 |  98% |   350 |  99% |   345 |  98%
CD         |   297 |   279 |  94% |   278 |  94% |   271 |  91%
ADV        |   252 |   218 |  87% |   237 |  94% |   214 |  85%
ITJ        |   219 |   217 |  99% |   217 |  99% |   217 |  99%
ADJ        |   218 |   208 |  95% |   211 |  97% |   205 |  94%
NAME       |   203 |   201 |  99% |   199 |  98% |   199 |  98%
CNJ        |   138 |   116 |  84% |   117 |  85% |   116 |  84%
VS         |    99 |    94 |  95% |    95 |  96% |    93 |  94%
VAUX       |    87 |    86 |  99% |    86 |  99% |    85 |  98%
NT         |    56 |    54 |  96% |    53 |  95% |    53 |  95%
VADJ       |    41 |    41 | 100% |    41 | 100% |    41 | 100%
UNIT       |    21 |    21 | 100% |    16 |  76% |    16 |  76%
GR         |    11 |    11 | 100% |    11 | 100% |    11 | 100%
--         |     6 |     3 |  50% |     3 |  50% |     3 |  50%
xxx        |     2 |     2 | 100% |     2 | 100% |     2 | 100%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5003 |   358 |   7% |   313 |   6% |   206 |   4%
-----------+-------+-------+------+-------+------+-------+-------
P          |  1045 |    87 |   8% |    80 |   8% |    34 |   3%
N          |  1043 |    72 |   7% |    75 |   7% |    43 |   4%
PS         |   505 |    62 |  12% |    45 |   9% |    44 |   9%
```

```
V          |   407 |    30 |   7% |    26 |   6% |    23 |   6%
PV         |   353 |     8 |   2% |     3 |   1% |     3 |   1%
CD         |   297 |    18 |   6% |    19 |   6% |    11 |   4%
ADV        |   252 |    34 |  13% |    15 |   6% |    11 |   4%
ITJ        |   219 |     2 |   1% |     2 |   1% |     2 |   1%
ADJ        |   218 |    10 |   5% |     7 |   3% |     4 |   2%
NAME       |   203 |     2 |   1% |     4 |   2% |     2 |   1%
CNJ        |   138 |    22 |  16% |    21 |  15% |    21 |  15%
VS         |    99 |     5 |   5% |     4 |   4% |     3 |   3%
VAUX       |    87 |     1 |   1% |     1 |   1% |     0 |   0%
NT         |    56 |     2 |   4% |     3 |   5% |     2 |   4%
VADJ       |    41 |     0 |   0% |     0 |   0% |     0 |   0%
UNIT       |    21 |     0 |   0% |     5 |  24% |     0 |   0%
GR         |    11 |     0 |   0% |     0 |   0% |     0 |   0%
--         |     6 |     3 |  50% |     3 |  50% |     3 |  50%
xxx        |     2 |     0 |   0% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
-               |   56 |      40 |     43 |      71.43 |         93.02
ADJ             |  854 |     706 |    801 |      82.67 |         88.14
COMP            | 2406 |    2367 |   2440 |      98.38 |         97.01
HD              |  110 |     102 |    114 |      92.73 |         89.47
MRK             |  436 |     435 |    436 |      99.77 |         99.77
ROOT            |  937 |     865 |    962 |      92.32 |         89.92
SBJ             |  204 |     175 |    207 |      85.78 |         84.54


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
-               |   56 |      40 |     43 |      71.43 |         93.02
ADJ             |  854 |     595 |    801 |      69.67 |         74.28
COMP            | 2406 |    2354 |   2440 |      97.84 |         96.48
HD              |  110 |      99 |    114 |      90.00 |         86.84
MRK             |  436 |     425 |    436 |      97.48 |         97.48
ROOT            |  937 |     865 |    962 |      92.32 |         89.92
SBJ             |  204 |     160 |    207 |      78.43 |         77.29


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  937 |     865 |    962 |      92.32 |         89.92
```

```
left            |  417 |     409 |    413 |      98.08 |         99.03
right           | 3649 |    3549 |   3628 |      97.26 |         97.82
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  937 |     865 |    962 |      92.32 |         89.92
1               | 3228 |    3200 |   3273 |      99.13 |         97.77
2               |  270 |     224 |    265 |      82.96 |         84.53
3-6             |  356 |     291 |    357 |      81.74 |         81.51
7-...           |  212 |     117 |    146 |      55.19 |         80.14
```

C.10 Portuguese

```
Labeled   attachment score: 4349 / 5009 * 100 = 86.82 %
Unlabeled attachment score: 4576 / 5009 * 100 = 91.36 %
Label accuracy score:       4531 / 5009 * 100 = 90.46 %

================================================================================

Evaluation of the results in portuguese.nonproj.pred
vs. gold standard portuguese_bosque_test.conll:

Legend: ’.S’ - the beginning of a sentence, ’.E’ - the end of a sentence

Number of non-scoring tokens: 858

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5009 |  4576 |  91% |  4531 |  90% |  4349 |  87%
-----------+-------+-------+------+-------+------+-------+-------
n          |  1143 |  1071 |  94% |  1071 |  94% |  1044 |  91%
prp        |   898 |   753 |  84% |   716 |  80% |   662 |  74%
art        |   780 |   777 | 100% |   778 | 100% |   777 | 100%
v          |   732 |   643 |  88% |   629 |  86% |   595 |  81%
prop       |   332 |   307 |  92% |   305 |  92% |   289 |  87%
pron       |   321 |   313 |  98% |   298 |  93% |   297 |  93%
adv        |   248 |   201 |  81% |   203 |  82% |   186 |  75%
adj        |   230 |   226 |  98% |   220 |  96% |   219 |  95%
conj       |   182 |   152 |  84% |   176 |  97% |   150 |  82%
num        |   134 |   126 |  94% |   128 |  96% |   124 |  93%
pp         |     9 |     7 |  78% |     7 |  78% |     6 |  67%
```

```
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5009 |   433 |   9% |   478 |  10% |   251 |   5%
-----------+-------+-------+------+-------+------+-------+-------
n          |  1143 |    72 |   6% |    72 |   6% |    45 |   4%
prp        |   898 |   145 |  16% |   182 |  20% |    91 |  10%
art        |   780 |     3 |   0% |     2 |   0% |     2 |   0%
v          |   732 |    89 |  12% |   103 |  14% |    55 |   8%
prop       |   332 |    25 |   8% |    27 |   8% |     9 |   3%
pron       |   321 |     8 |   2% |    23 |   7% |     7 |   2%
adv        |   248 |    47 |  19% |    45 |  18% |    30 |  12%
adj        |   230 |     4 |   2% |    10 |   4% |     3 |   1%
conj       |   182 |    30 |  16% |     6 |   3% |     4 |   2%
num        |   134 |     8 |   6% |     6 |   4% |     4 |   3%
pp         |     9 |     2 |  22% |     2 |  22% |     1 |  11%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
>A              |   42 |      31 |     47 |      73.81 |         65.96
>N              | 1043 |    1032 |   1037 |      98.95 |         99.52
>P              |   10 |       1 |      2 |      10.00 |         50.00
?               |    9 |       1 |      1 |      11.11 |        100.00
A<              |   29 |      26 |     39 |      89.66 |         66.67
ACC             |  316 |     288 |    327 |      91.14 |         88.07
ACC>-PASS       |    2 |       0 |      0 |       0.00 |           NaN
ADVL            |  451 |     349 |    459 |      77.38 |         76.03
ADVO            |    8 |       1 |      1 |      12.50 |        100.00
ADVS            |   20 |       5 |     10 |      25.00 |         50.00
APP             |   25 |      18 |     29 |      72.00 |         62.07
AUX             |    8 |       7 |     13 |      87.50 |         53.85
AUX<            |    2 |       0 |      0 |       0.00 |           NaN
CJT             |  166 |     139 |    157 |      83.73 |         88.54
CJT&ADVL        |    2 |       0 |      0 |       0.00 |           NaN
CO              |  127 |     126 |    126 |      99.21 |        100.00
COM             |    2 |       0 |      0 |       0.00 |           NaN
DAT             |    5 |       5 |      5 |     100.00 |        100.00
EXC             |    1 |       0 |      0 |       0.00 |           NaN
FOC             |    4 |       4 |      4 |     100.00 |        100.00
KOMP<           |    3 |       1 |      2 |      33.33 |         50.00
MV              |   83 |      76 |     87 |      91.57 |         87.36
N<              |  712 |     675 |    732 |      94.80 |         92.21
```

```
N<PRED          |  139 |      97 |    142 |      69.78 |         68.31
OC              |    8 |       3 |      9 |      37.50 |         33.33
P               |    2 |       0 |      1 |       0.00 |          0.00
P<              |  884 |     871 |    884 |      98.53 |         98.53
PASS            |   17 |      17 |     20 |     100.00 |         85.00
PCJT            |    1 |       0 |      1 |       0.00 |          0.00
PIV             |   82 |      48 |     74 |      58.54 |         64.86
PMV             |    2 |       0 |      0 |       0.00 |           NaN
PRED            |   11 |       5 |      5 |      45.45 |        100.00
PRT-AUX         |    1 |       0 |      0 |       0.00 |           NaN
PRT-AUX<        |   10 |      10 |     14 |     100.00 |         71.43
QUE             |    6 |       0 |      3 |       0.00 |          0.00
S<              |    5 |       0 |      0 |       0.00 |           NaN
SC              |   77 |      62 |     79 |      80.52 |         78.48
STA             |  249 |     233 |    254 |      93.57 |         91.73
SUB             |   50 |      48 |     54 |      96.00 |         88.89
SUBJ            |  352 |     318 |    354 |      90.34 |         89.83
TOP             |    1 |       0 |      0 |       0.00 |           NaN
UTT             |   41 |      34 |     37 |      82.93 |         91.89
VOC             |    1 |       0 |      0 |       0.00 |           NaN


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
>A              |   42 |      31 |     47 |      73.81 |         65.96
>N              | 1043 |    1031 |   1037 |      98.85 |         99.42
>P              |   10 |       1 |      2 |      10.00 |         50.00
?               |    9 |       1 |      1 |      11.11 |        100.00
A<              |   29 |      26 |     39 |      89.66 |         66.67
ACC             |  316 |     284 |    327 |      89.87 |         86.85
ACC>-PASS       |    2 |       0 |      0 |       0.00 |           NaN
ADVL            |  451 |     308 |    459 |      68.29 |         67.10
ADVO            |    8 |       1 |      1 |      12.50 |        100.00
ADVS            |   20 |       4 |     10 |      20.00 |         40.00
APP             |   25 |      11 |     29 |      44.00 |         37.93
AUX             |    8 |       7 |     13 |      87.50 |         53.85
AUX<            |    2 |       0 |      0 |       0.00 |           NaN
CJT             |  166 |     111 |    157 |      66.87 |         70.70
CJT&ADVL        |    2 |       0 |      0 |       0.00 |           NaN
CO              |  127 |     100 |    126 |      78.74 |         79.37
COM             |    2 |       0 |      0 |       0.00 |           NaN
DAT             |    5 |       5 |      5 |     100.00 |        100.00
EXC             |    1 |       0 |      0 |       0.00 |           NaN
FOC             |    4 |       2 |      4 |      50.00 |         50.00
KOMP<           |    3 |       0 |      2 |       0.00 |          0.00
MV              |   83 |      76 |     87 |      91.57 |         87.36
N<              |  712 |     640 |    732 |      89.89 |         87.43
N<PRED          |  139 |      72 |    142 |      51.80 |         50.70
OC              |    8 |       3 |      9 |      37.50 |         33.33
P               |    2 |       0 |      1 |       0.00 |          0.00
```

```
P<              |  884 |     871 |    884 |      98.53 |         98.53
PASS            |   17 |      16 |     20 |      94.12 |         80.00
PCJT            |    1 |       0 |      1 |       0.00 |          0.00
PIV             |   82 |      48 |     74 |      58.54 |         64.86
PMV             |    2 |       0 |      0 |       0.00 |           NaN
PRED            |   11 |       4 |      5 |      36.36 |         80.00
PRT-AUX         |    1 |       0 |      0 |       0.00 |           NaN
PRT-AUX<        |   10 |      10 |     14 |     100.00 |         71.43
QUE             |    6 |       0 |      3 |       0.00 |          0.00
S<              |    5 |       0 |      0 |       0.00 |           NaN
SC              |   77 |      61 |     79 |      79.22 |         77.22
STA             |  249 |     232 |    254 |      93.17 |         91.34
SUB             |   50 |      48 |     54 |      96.00 |         88.89
SUBJ            |  352 |     311 |    354 |      88.35 |         87.85
TOP             |    1 |       0 |      0 |       0.00 |           NaN
UTT             |   41 |      34 |     37 |      82.93 |         91.89
VOC             |    1 |       0 |      0 |       0.00 |           NaN


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  288 |     271 |    288 |      94.10 |         94.10
left            | 3006 |    2966 |   3003 |      98.67 |         98.77
right           | 1715 |    1680 |   1718 |      97.96 |         97.79
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  288 |     271 |    288 |      94.10 |         94.10
1               | 2658 |    2610 |   2696 |      98.19 |         96.81
2               | 1117 |    1059 |   1121 |      94.81 |         94.47
3-6             |  623 |     512 |    630 |      82.18 |         81.27
7-...           |  323 |     223 |    274 |      69.04 |         81.39
```

C.11 Slovene

```
Labeled   attachment score: 3675 / 5004 * 100 = 73.44 %
Unlabeled attachment score: 4162 / 5004 * 100 = 83.17 %
Label accuracy score:       4129 / 5004 * 100 = 82.51 %

================================================================================

Evaluation of the results in slovene.nonproj.pred
```

```
vs. gold standard slovene_sdt_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 1386

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5004 |  4162 |  83% |  4129 |  83% |  3675 |  73%
-----------+-------+-------+------+-------+------+-------+-------
Verb       |  1483 |  1256 |  85% |  1229 |  83% |  1138 |  77%
Noun       |   994 |   875 |  88% |   740 |  74% |   693 |  70%
Pronoun    |   583 |   517 |  89% |   437 |  75% |   413 |  71%
Conjunction |   448 |   283 |  63% |   408 |  91% |   279 |  62%
Adposition |   419 |   327 |  78% |   415 |  99% |   324 |  77%
Adverb     |   399 |   334 |  84% |   339 |  85% |   303 |  76%
Adjective  |   395 |   347 |  88% |   346 |  88% |   330 |  84%
Particle   |   214 |   168 |  79% |   164 |  77% |   149 |  70%
Numeral    |    45 |    38 |  84% |    29 |  64% |    29 |  64%
Interjection |    15 |     8 |  53% |    13 |  87% |     8 |  53%
Abbreviation |     9 |     9 | 100% |     9 | 100% |     9 | 100%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5004 |   842 |  17% |   875 |  17% |   388 |   8%
-----------+-------+-------+------+-------+------+-------+-------
Verb       |  1483 |   227 |  15% |   254 |  17% |   136 |   9%
Noun       |   994 |   119 |  12% |   254 |  26% |    72 |   7%
Pronoun    |   583 |    66 |  11% |   146 |  25% |    42 |   7%
Conjunction |   448 |   165 |  37% |    40 |   9% |    36 |   8%
Adposition |   419 |    92 |  22% |     4 |   1% |     1 |   0%
Adverb     |   399 |    65 |  16% |    60 |  15% |    29 |   7%
Adjective  |   395 |    48 |  12% |    49 |  12% |    32 |   8%
Particle   |   214 |    46 |  21% |    50 |  23% |    31 |  14%
Numeral    |    45 |     7 |  16% |    16 |  36% |     7 |  16%
Interjection |    15 |     7 |  47% |     2 |  13% |     2 |  13%
Abbreviation |     9 |     0 |   0% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
```

```
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
Adv             |  680 |     552 |    727 |      81.18 |         75.93
AdvAtr          |   11 |       0 |      2 |       0.00 |          0.00
Atr             |  760 |     658 |    799 |      86.58 |         82.35
AtrAdv          |    9 |       0 |      7 |       0.00 |          0.00
AtrAtr          |    4 |       0 |      0 |       0.00 |           NaN
AtrObj          |    1 |       0 |      0 |       0.00 |           NaN
Atv             |   57 |      27 |     39 |      47.37 |         69.23
AtvV            |   27 |       6 |      7 |      22.22 |         85.71
AuxC            |  234 |     225 |    233 |      96.15 |         96.57
AuxG            |    0 |       0 |      1 |        NaN |          0.00
AuxP            |  412 |     410 |    411 |      99.51 |         99.76
AuxR            |    2 |       0 |      1 |       0.00 |          0.00
AuxT            |   47 |      37 |     61 |      78.72 |         60.66
AuxV            |  631 |     616 |    645 |      97.62 |         95.50
AuxX            |    0 |       0 |      1 |        NaN |          0.00
AuxY            |  154 |      91 |    125 |      59.09 |         72.80
AuxZ            |  110 |      91 |    122 |      82.73 |         74.59
Coord           |  174 |     161 |    177 |      92.53 |         90.96
ExD             |  156 |      75 |    141 |      48.08 |         53.19
Obj             |  501 |     367 |    517 |      73.25 |         70.99
ObjAtr          |    3 |       0 |      0 |       0.00 |           NaN
Pnom            |  158 |     114 |    172 |      72.15 |         66.28
Pred            |  496 |     430 |    504 |      86.69 |         85.32
Sb              |  377 |     269 |    312 |      71.35 |         86.22


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
Adv             |  680 |     510 |    727 |      75.00 |         70.15
AdvAtr          |   11 |       0 |      2 |       0.00 |          0.00
Atr             |  760 |     626 |    799 |      82.37 |         78.35
AtrAdv          |    9 |       0 |      7 |       0.00 |          0.00
AtrAtr          |    4 |       0 |      0 |       0.00 |           NaN
AtrObj          |    1 |       0 |      0 |       0.00 |           NaN
Atv             |   57 |      27 |     39 |      47.37 |         69.23
AtvV            |   27 |       5 |      7 |      18.52 |         71.43
AuxC            |  234 |     161 |    233 |      68.80 |         69.10
AuxG            |    0 |       0 |      1 |        NaN |          0.00
AuxP            |  412 |     320 |    411 |      77.67 |         77.86
AuxR            |    2 |       0 |      1 |       0.00 |          0.00
AuxT            |   47 |      36 |     61 |      76.60 |         59.02
AuxV            |  631 |     602 |    645 |      95.40 |         93.33
AuxX            |    0 |       0 |      1 |        NaN |          0.00
AuxY            |  154 |      82 |    125 |      53.25 |         65.60
AuxZ            |  110 |      82 |    122 |      74.55 |         67.21
Coord           |  174 |      99 |    177 |      56.90 |         55.93
ExD             |  156 |      65 |    141 |      41.67 |         46.10
Obj             |  501 |     336 |    517 |      67.07 |         64.99
```

```
ObjAtr          |    3 |       0 |      0 |       0.00 |           NaN
Pnom            |  158 |     109 |    172 |      68.99 |         63.37
Pred            |  496 |     368 |    504 |      74.19 |         73.02
Sb              |  377 |     247 |    312 |      65.52 |         79.17


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  392 |     312 |    378 |      79.59 |         82.54
left            | 2339 |    2140 |   2327 |      91.49 |         91.96
right           | 2273 |    2085 |   2299 |      91.73 |         90.69
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  392 |     312 |    378 |      79.59 |         82.54
1               | 2336 |    2197 |   2375 |      94.05 |         92.51
2               | 1063 |     952 |   1113 |      89.56 |         85.53
3-6             |  976 |     755 |    931 |      77.36 |         81.10
7-...           |  237 |     120 |    207 |      50.63 |         57.97
```

C.12 Spanish

```
Labeled   attachment score: 4105 / 4991 * 100 = 82.25 %
Unlabeled attachment score: 4295 / 4991 * 100 = 86.05 %
Label accuracy score:       4512 / 4991 * 100 = 90.40 %

================================================================================

Evaluation of the results in spanish.proj.pred
vs. gold standard spanish_cast3lb_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 703

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4991 |  4295 |  86% |  4512 |  90% |  4105 |  82%
```

```
-----------+-------+-------+------+-------+------+-------+-------
n          |  1310 |  1208 |  92% |  1225 |  94% |  1159 |  88%
d          |   856 |   844 |  99% |   855 | 100% |   844 |  99%
s          |   815 |   653 |  80% |   665 |  82% |   595 |  73%
v          |   695 |   495 |  71% |   586 |  84% |   476 |  68%
a          |   415 |   372 |  90% |   396 |  95% |   364 |  88%
c          |   350 |   250 |  71% |   327 |  93% |   245 |  70%
p          |   277 |   247 |  89% |   231 |  83% |   218 |  79%
r          |   224 |   184 |  82% |   183 |  82% |   166 |  74%
w          |    20 |    20 | 100% |    19 |  95% |    19 |  95%
z          |    15 |    12 |  80% |    15 | 100% |    12 |  80%
Z          |    13 |    10 |  77% |     9 |  69% |     7 |  54%
i          |     1 |     0 |   0% |     1 | 100% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  4991 |   696 |  14% |   479 |  10% |   289 |   6%
-----------+-------+-------+------+-------+------+-------+-------
n          |  1310 |   102 |   8% |    85 |   6% |    36 |   3%
d          |   856 |    12 |   1% |     1 |   0% |     1 |   0%
s          |   815 |   162 |  20% |   150 |  18% |    92 |  11%
v          |   695 |   200 |  29% |   109 |  16% |    90 |  13%
a          |   415 |    43 |  10% |    19 |   5% |    11 |   3%
c          |   350 |   100 |  29% |    23 |   7% |    18 |   5%
p          |   277 |    30 |  11% |    46 |  17% |    17 |   6%
r          |   224 |    40 |  18% |    41 |  18% |    23 |  10%
w          |    20 |     0 |   0% |     1 |   5% |     0 |   0%
z          |    15 |     3 |  20% |     0 |   0% |     0 |   0%
Z          |    13 |     3 |  23% |     4 |  31% |     1 |   8%
i          |     1 |     1 | 100% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ATR             |   85 |      72 |     86 |      84.71 |         83.72
CAG             |   12 |      10 |     15 |      83.33 |         66.67
CC              |  383 |     289 |    417 |      75.46 |         69.30
CD              |  289 |     205 |    265 |      70.93 |         77.36
CD.Q            |   15 |      13 |     14 |      86.67 |         92.86
CI              |   47 |      32 |     45 |      68.09 |         71.11
CPRED           |    2 |       0 |      0 |       0.00 |           NaN
CPRED.CD        |    6 |       3 |      4 |      50.00 |         75.00
CPRED.SUJ       |    6 |       3 |      3 |      50.00 |        100.00
```

```
CREG            |   45 |      15 |     23 |      33.33 |         65.22
ET              |   17 |      11 |     15 |      64.71 |         73.33
IMPERS          |    7 |       3 |      4 |      42.86 |         75.00
MOD             |   15 |       9 |      9 |      60.00 |        100.00
NEG             |   43 |      41 |     42 |      95.35 |         97.62
PASS            |   17 |      16 |     20 |      94.12 |         80.00
ROOT            |  197 |     172 |    199 |      87.31 |         86.43
SUJ             |  340 |     278 |    333 |      81.76 |         83.48
_               | 3465 |    3340 |   3497 |      96.39 |         95.51


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ATR             |   85 |      72 |     86 |      84.71 |         83.72
CAG             |   12 |       9 |     15 |      75.00 |         60.00
CC              |  383 |     264 |    417 |      68.93 |         63.31
CD              |  289 |     190 |    265 |      65.74 |         71.70
CD.Q            |   15 |      13 |     14 |      86.67 |         92.86
CI              |   47 |      31 |     45 |      65.96 |         68.89
CPRED           |    2 |       0 |      0 |       0.00 |           NaN
CPRED.CD        |    6 |       3 |      4 |      50.00 |         75.00
CPRED.SUJ       |    6 |       3 |      3 |      50.00 |        100.00
CREG            |   45 |      15 |     23 |      33.33 |         65.22
ET              |   17 |      10 |     15 |      58.82 |         66.67
IMPERS          |    7 |       3 |      4 |      42.86 |         75.00
MOD             |   15 |       8 |      9 |      53.33 |         88.89
NEG             |   43 |      40 |     42 |      93.02 |         95.24
PASS            |   17 |      16 |     20 |      94.12 |         80.00
ROOT            |  197 |     172 |    199 |      87.31 |         86.43
SUJ             |  340 |     263 |    333 |      77.35 |         78.98
_               | 3465 |    2993 |   3497 |      86.38 |         85.59


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  197 |     172 |    199 |      87.31 |         86.43
left            | 3028 |    2940 |   3030 |      97.09 |         97.03
right           | 1766 |    1677 |   1762 |      94.96 |         95.18
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  197 |     172 |    199 |      87.31 |         86.43
```

```
1              | 2671 |    2576 |   2708 |      96.44 |         95.13
2              |  979 |     886 |    989 |      90.50 |         89.59
3-6            |  736 |     560 |    722 |      76.09 |         77.56
7-...          |  408 |     280 |    373 |      68.63 |         75.07
```

C.13 Swedish

```
Labeled   attachment score: 4145 / 5021 * 100 = 82.55 %
Unlabeled attachment score: 4465 / 5021 * 100 = 88.93 %
Label accuracy score:       4297 / 5021 * 100 = 85.58 %

================================================================================

Evaluation of the results in swedish.proj.pred
vs. gold standard swedish_talbanken05_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 635

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5021 |  4465 |  89% |  4297 |  86% |  4145 |  83%
-----------+-------+-------+------+-------+------+-------+-------
NN         |  1109 |  1008 |  91% |   994 |  90% |   976 |  88%
PR         |   689 |   551 |  80% |   430 |  62% |   399 |  58%
PO         |   620 |   586 |  95% |   582 |  94% |   572 |  92%
VV         |   455 |   397 |  87% |   384 |  84% |   372 |  82%
AB         |   365 |   315 |  86% |   315 |  86% |   282 |  77%
AJ         |   322 |   302 |  94% |   302 |  94% |   297 |  92%
VN         |   221 |   204 |  92% |   197 |  89% |   193 |  87%
++         |   178 |   166 |  93% |   177 |  99% |   166 |  93%
RO         |   145 |   124 |  86% |   112 |  77% |   110 |  76%
ID         |   114 |   113 |  99% |   114 | 100% |   113 |  99%
UK         |   111 |   105 |  95% |   111 | 100% |   105 |  95%
PN         |   106 |    91 |  86% |    87 |  82% |    86 |  81%
EN         |    97 |    94 |  97% |    95 |  98% |    94 |  97%
AV         |    70 |    61 |  87% |    54 |  77% |    53 |  76%
HV         |    60 |    45 |  75% |    44 |  73% |    42 |  70%
QV         |    58 |    50 |  86% |    50 |  86% |    49 |  84%
TP         |    53 |    47 |  89% |    45 |  85% |    45 |  85%
IM         |    45 |    43 |  96% |    45 | 100% |    43 |  96%
AN         |    38 |    33 |  87% |    31 |  82% |    30 |  79%
FV         |    35 |    31 |  89% |    32 |  91% |    31 |  89%
MN         |    35 |    20 |  57% |    19 |  54% |    15 |  43%
SV         |    32 |    30 |  94% |    28 |  88% |    27 |  84%
```

```
BV         |    21 |    17 |  81% |    15 |  71% |    15 |  71%
MV         |    13 |    11 |  85% |    10 |  77% |    10 |  77%
GV         |     8 |     7 |  88% |     7 |  88% |     7 |  88%
KV         |     8 |     7 |  88% |     7 |  88% |     7 |  88%
PU         |     4 |     3 |  75% |     4 | 100% |     3 |  75%
WV         |     4 |     2 |  50% |     2 |  50% |     2 |  50%
XX         |     3 |     1 |  33% |     3 | 100% |     1 |  33%
IK         |     1 |     1 | 100% |     0 |   0% |     0 |   0%
IG         |     1 |     0 |   0% |     1 | 100% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5021 |   556 |  11% |   724 |  14% |   404 |   8%
-----------+-------+-------+------+-------+------+-------+-------
NN         |  1109 |   101 |   9% |   115 |  10% |    83 |   7%
PR         |   689 |   138 |  20% |   259 |  38% |   107 |  16%
PO         |   620 |    34 |   5% |    38 |   6% |    24 |   4%
VV         |   455 |    58 |  13% |    71 |  16% |    46 |  10%
AB         |   365 |    50 |  14% |    50 |  14% |    17 |   5%
AJ         |   322 |    20 |   6% |    20 |   6% |    15 |   5%
VN         |   221 |    17 |   8% |    24 |  11% |    13 |   6%
++         |   178 |    12 |   7% |     1 |   1% |     1 |   1%
RO         |   145 |    21 |  14% |    33 |  23% |    19 |  13%
ID         |   114 |     1 |   1% |     0 |   0% |     0 |   0%
UK         |   111 |     6 |   5% |     0 |   0% |     0 |   0%
PN         |   106 |    15 |  14% |    19 |  18% |    14 |  13%
EN         |    97 |     3 |   3% |     2 |   2% |     2 |   2%
AV         |    70 |     9 |  13% |    16 |  23% |     8 |  11%
HV         |    60 |    15 |  25% |    16 |  27% |    13 |  22%
QV         |    58 |     8 |  14% |     8 |  14% |     7 |  12%
TP         |    53 |     6 |  11% |     8 |  15% |     6 |  11%
IM         |    45 |     2 |   4% |     0 |   0% |     0 |   0%
AN         |    38 |     5 |  13% |     7 |  18% |     4 |  11%
FV         |    35 |     4 |  11% |     3 |   9% |     3 |   9%
MN         |    35 |    15 |  43% |    16 |  46% |    11 |  31%
SV         |    32 |     2 |   6% |     4 |  12% |     1 |   3%
BV         |    21 |     4 |  19% |     6 |  29% |     4 |  19%
MV         |    13 |     2 |  15% |     3 |  23% |     2 |  15%
GV         |     8 |     1 |  12% |     1 |  12% |     1 |  12%
KV         |     8 |     1 |  12% |     1 |  12% |     1 |  12%
PU         |     4 |     1 |  25% |     0 |   0% |     0 |   0%
WV         |     4 |     2 |  50% |     2 |  50% |     2 |  50%
XX         |     3 |     2 |  67% |     0 |   0% |     0 |   0%
IK         |     1 |     0 |   0% |     1 | 100% |     0 |   0%
IG         |     1 |     1 | 100% |     0 |   0% |     0 |   0%
-----------+-------+-------+------+-------+------+-------+-------
```

```
Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
++              |  181 |     180 |    181 |      99.45 |         99.45
+A              |   51 |      49 |     54 |      96.08 |         90.74
+F              |   49 |      24 |     46 |      48.98 |         52.17
AA              |  266 |     155 |    234 |      58.27 |         66.24
AG              |    6 |       5 |      9 |      83.33 |         55.56
AN              |   29 |      11 |     25 |      37.93 |         44.00
AT              |  234 |     230 |    236 |      98.29 |         97.46
BS              |    1 |       0 |      2 |       0.00 |          0.00
C+              |   12 |       9 |     10 |      75.00 |         90.00
CA              |   41 |      35 |     39 |      85.37 |         89.74
CC              |  218 |     203 |    237 |      93.12 |         85.65
DB              |    3 |       2 |      3 |      66.67 |         66.67
DT              |  554 |     533 |    582 |      96.21 |         91.58
EF              |    2 |       0 |      1 |       0.00 |          0.00
ES              |   15 |       7 |      8 |      46.67 |         87.50
ET              |  341 |     265 |    348 |      77.71 |         76.15
FS              |   16 |      11 |     11 |      68.75 |        100.00
FV              |    2 |       0 |      0 |       0.00 |           NaN
HD              |  129 |     114 |    122 |      88.37 |         93.44
IG              |    1 |       1 |      1 |     100.00 |        100.00
IK              |    1 |       0 |      1 |       0.00 |          0.00
IM              |   45 |      45 |     45 |     100.00 |        100.00
IO              |   12 |       7 |      9 |      58.33 |         77.78
IV              |   11 |      11 |     11 |     100.00 |        100.00
KA              |   15 |       7 |      9 |      46.67 |         77.78
MA              |    6 |       5 |      8 |      83.33 |         62.50
MS              |   17 |       8 |     27 |      47.06 |         29.63
NA              |   42 |      42 |     43 |     100.00 |         97.67
OA              |  160 |     118 |    211 |      73.75 |         55.92
OO              |  284 |     243 |    294 |      85.56 |         82.65
PA              |  677 |     642 |    688 |      94.83 |         93.31
PL              |   48 |      32 |     39 |      66.67 |         82.05
PT              |   19 |       9 |     11 |      47.37 |         81.82
RA              |  134 |      64 |    107 |      47.76 |         59.81
ROOT            |  389 |     359 |    389 |      92.29 |         92.29
SP              |   89 |      75 |     90 |      84.27 |         83.33
SS              |  507 |     464 |    508 |      91.52 |         91.34
TA              |  139 |      77 |    112 |      55.40 |         68.75
UK              |  111 |     111 |    111 |     100.00 |        100.00
VA              |    8 |       7 |      7 |      87.50 |        100.00
VG              |  135 |     130 |    140 |      96.30 |         92.86
VO              |    0 |       0 |      2 |        NaN |          0.00
XA              |    0 |       0 |      1 |        NaN |          0.00
XT              |    3 |       3 |      3 |     100.00 |        100.00
XX              |   18 |       4 |      6 |      22.22 |         66.67
```

```
Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
++              |  181 |     169 |    181 |      93.37 |         93.37
+A              |   51 |      45 |     54 |      88.24 |         83.33
+F              |   49 |      18 |     46 |      36.73 |         39.13
AA              |  266 |     145 |    234 |      54.51 |         61.97
AG              |    6 |       5 |      9 |      83.33 |         55.56
AN              |   29 |       8 |     25 |      27.59 |         32.00
AT              |  234 |     230 |    236 |      98.29 |         97.46
BS              |    1 |       0 |      2 |       0.00 |          0.00
C+              |   12 |       8 |     10 |      66.67 |         80.00
CA              |   41 |      25 |     39 |      60.98 |         64.10
CC              |  218 |     179 |    237 |      82.11 |         75.53
DB              |    3 |       2 |      3 |      66.67 |         66.67
DT              |  554 |     523 |    582 |      94.40 |         89.86
EF              |    2 |       0 |      1 |       0.00 |          0.00
ES              |   15 |       7 |      8 |      46.67 |         87.50
ET              |  341 |     245 |    348 |      71.85 |         70.40
FS              |   16 |      11 |     11 |      68.75 |        100.00
FV              |    2 |       0 |      0 |       0.00 |           NaN
HD              |  129 |     113 |    122 |      87.60 |         92.62
IG              |    1 |       0 |      1 |       0.00 |          0.00
IK              |    1 |       0 |      1 |       0.00 |          0.00
IM              |   45 |      43 |     45 |      95.56 |         95.56
IO              |   12 |       7 |      9 |      58.33 |         77.78
IV              |   11 |      11 |     11 |     100.00 |        100.00
KA              |   15 |       7 |      9 |      46.67 |         77.78
MA              |    6 |       5 |      8 |      83.33 |         62.50
MS              |   17 |       8 |     27 |      47.06 |         29.63
NA              |   42 |      35 |     43 |      83.33 |         81.40
OA              |  160 |     113 |    211 |      70.62 |         53.55
OO              |  284 |     239 |    294 |      84.15 |         81.29
PA              |  677 |     640 |    688 |      94.53 |         93.02
PL              |   48 |      32 |     39 |      66.67 |         82.05
PT              |   19 |       9 |     11 |      47.37 |         81.82
RA              |  134 |      53 |    107 |      39.55 |         49.53
ROOT            |  389 |     359 |    389 |      92.29 |         92.29
SP              |   89 |      75 |     90 |      84.27 |         83.33
SS              |  507 |     459 |    508 |      90.53 |         90.35
TA              |  139 |      73 |    112 |      52.52 |         65.18
UK              |  111 |     105 |    111 |      94.59 |         94.59
VA              |    8 |       6 |      7 |      75.00 |         85.71
VG              |  135 |     128 |    140 |      94.81 |         91.43
VO              |    0 |       0 |      2 |        NaN |          0.00
XA              |    0 |       0 |      1 |        NaN |          0.00
XT              |    3 |       3 |      3 |     100.00 |        100.00
XX              |   18 |       2 |      6 |      11.11 |         33.33


Precision and recall of binned HEAD direction
```

```
----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  389 |     359 |    389 |      92.29 |         92.29
left            | 2745 |    2652 |   2748 |      96.61 |         96.51
right           | 1887 |    1797 |   1884 |      95.23 |         95.38
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  389 |     359 |    389 |      92.29 |         92.29
1               | 2512 |    2389 |   2522 |      95.10 |         94.73
2               | 1107 |    1031 |   1130 |      93.13 |         91.24
3-6             |  803 |     670 |    805 |      83.44 |         83.23
7-...           |  210 |     132 |    175 |      62.86 |         75.43
```

C.14 Turkish

```
Labeled   attachment score: 3173 / 5021 * 100 = 63.19 %
Unlabeled attachment score: 3749 / 5021 * 100 = 74.67 %
Label accuracy score:       3889 / 5021 * 100 = 77.45 %

================================================================================

Evaluation of the results in turkish.proj.pred
vs. gold standard turkish_metu_sabanci_test.conll:

Legend: '.S' - the beginning of a sentence, '.E' - the end of a sentence

Number of non-scoring tokens: 2526

The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5021 |  3749 |  75% |  3889 |  77% |  3173 |  63%
-----------+-------+-------+------+-------+------+-------+-------
Noun       |  2209 |  1565 |  71% |  1491 |  67% |  1186 |  54%
Verb       |   891 |   767 |  86% |   753 |  85% |   722 |  81%
Adj        |   552 |   440 |  80% |   496 |  90% |   407 |  74%
Adv        |   346 |   234 |  68% |   303 |  88% |   215 |  62%
Pron       |   297 |   214 |  72% |   221 |  74% |   168 |  57%
Conj       |   244 |   175 |  72% |   212 |  87% |   163 |  67%
```

```
Det        |   213 |   183 |  86% |   178 |  84% |   155 |  73%
Postp      |   146 |    77 |  53% |   132 |  90% |    71 |  49%
Num        |    58 |    46 |  79% |    51 |  88% |    44 |  76%
Ques       |    40 |    33 |  82% |    37 |  92% |    32 |  80%
Interj     |    25 |    15 |  60% |    15 |  60% |    10 |  40%
-----------+-------+-------+------+-------+------+-------+-------


The overall error rate and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Error      | words | head  |   %  |  dep  |   %  | both  |   %
Rate       |       | err   |      |  err  |      | wrong |
-----------+-------+-------+------+-------+------+-------+-------
total      |  5021 |  1272 |  25% |  1132 |  23% |   556 |  11%
-----------+-------+-------+------+-------+------+-------+-------
Noun       |  2209 |   644 |  29% |   718 |  33% |   339 |  15%
Verb       |   891 |   124 |  14% |   138 |  15% |    93 |  10%
Adj        |   552 |   112 |  20% |    56 |  10% |    23 |   4%
Adv        |   346 |   112 |  32% |    43 |  12% |    24 |   7%
Pron       |   297 |    83 |  28% |    76 |  26% |    30 |  10%
Conj       |   244 |    69 |  28% |    32 |  13% |    20 |   8%
Det        |   213 |    30 |  14% |    35 |  16% |     7 |   3%
Postp      |   146 |    69 |  47% |    14 |  10% |     8 |   5%
Num        |    58 |    12 |  21% |     7 |  12% |     5 |   9%
Ques       |    40 |     7 |  18% |     3 |   7% |     2 |   5%
Interj     |    25 |    10 |  40% |    10 |  40% |     5 |  20%
-----------+-------+-------+------+-------+------+-------+-------



Precision and recall of DEPREL

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ABLATIVE.ADJUNCT |   51 |      36 |     64 |      70.59 |         56.25
APPOSITION      |   32 |       2 |      8 |       6.25 |         25.00
CLASSIFIER      |  191 |     150 |    245 |      78.53 |         61.22
COLLOCATION     |    7 |       0 |      0 |       0.00 |           NaN
COORDINATION    |  249 |     180 |    279 |      72.29 |         64.52
DATIVE.ADJUNCT  |  178 |     126 |    189 |      70.79 |         66.67
DETERMINER      |  180 |     176 |    213 |      97.78 |         82.63
INSTRUMENTAL.ADJUNCT |   19 |       4 |     20 |      21.05 |         20.00
INTENSIFIER     |  103 |      87 |    101 |      84.47 |         86.14
LOCATIVE.ADJUNCT |  124 |     102 |    136 |      82.26 |         75.00
MODIFIER        | 1362 |    1103 |   1301 |      80.98 |         84.78
NEGATIVE.PARTICLE |   17 |      12 |     15 |      70.59 |         80.00
OBJECT          | 1010 |     742 |    974 |      73.47 |         76.18
POSSESSOR       |  135 |     109 |    133 |      80.74 |         81.95
QUESTION.PARTICLE |   43 |      37 |     43 |      86.05 |         86.05
RELATIVIZER     |   13 |       7 |      9 |      53.85 |         77.78
ROOT            |  659 |     597 |    639 |      90.59 |         93.43
S.MODIFIER      |   76 |      43 |     63 |      56.58 |         68.25
```

```
SENTENCE        |   25 |      16 |     20 |      64.00 |         80.00
SUBJECT         |  491 |     340 |    539 |      69.25 |         63.08
VOCATIVE        |   56 |      20 |     30 |      35.71 |         66.67


Precision and recall of DEPREL + ATTACHMENT

----------------+------+---------+--------+------------+---------------
deprel          | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
ABLATIVE.ADJUNCT |   51 |      30 |     64 |      58.82 |         46.88
APPOSITION      |   32 |       1 |      8 |       3.12 |         12.50
CLASSIFIER      |  191 |     148 |    245 |      77.49 |         60.41
COLLOCATION     |    7 |       0 |      0 |       0.00 |           NaN
COORDINATION    |  249 |     129 |    279 |      51.81 |         46.24
DATIVE.ADJUNCT  |  178 |      90 |    189 |      50.56 |         47.62
DETERMINER      |  180 |     154 |    213 |      85.56 |         72.30
INSTRUMENTAL.ADJUNCT |   19 |       3 |     20 |      15.79 |         15.00
INTENSIFIER     |  103 |      81 |    101 |      78.64 |         80.20
LOCATIVE.ADJUNCT |  124 |      59 |    136 |      47.58 |         43.38
MODIFIER        | 1362 |     798 |   1301 |      58.59 |         61.34
NEGATIVE.PARTICLE |   17 |      12 |     15 |      70.59 |         80.00
OBJECT          | 1010 |     606 |    974 |      60.00 |         62.22
POSSESSOR       |  135 |     109 |    133 |      80.74 |         81.95
QUESTION.PARTICLE |   43 |      32 |     43 |      74.42 |         74.42
RELATIVIZER     |   13 |       7 |      9 |      53.85 |         77.78
ROOT            |  659 |     597 |    639 |      90.59 |         93.43
S.MODIFIER      |   76 |      34 |     63 |      44.74 |         53.97
SENTENCE        |   25 |      16 |     20 |      64.00 |         80.00
SUBJECT         |  491 |     254 |    539 |      51.73 |         47.12
VOCATIVE        |   56 |      13 |     30 |      23.21 |         43.33


Precision and recall of binned HEAD direction

----------------+------+---------+--------+------------+---------------
direction       | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  659 |     597 |    639 |      90.59 |         93.43
left            |  288 |     247 |    291 |      85.76 |         84.88
right           | 4074 |    4014 |   4091 |      98.53 |         98.12
self            |    0 |       0 |      0 |        NaN |           NaN


Precision and recall of binned HEAD distance

----------------+------+---------+--------+------------+---------------
distance        | gold | correct | system | recall (%) | precision (%)
----------------+------+---------+--------+------------+---------------
to_root         |  659 |     597 |    639 |      90.59 |         93.43
1               | 2187 |    1989 |   2389 |      90.95 |         83.26
2               |  844 |     556 |    850 |      65.88 |         65.41
3-6             |  897 |     577 |    820 |      64.33 |         70.37
```

```
7-...           |  434 |     232 |    323 |      53.46 |         71.83
```

Bibliography

[1] A. Abeillé, editor. *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, Dordrecht, 2003.

[2] S. Afonso, E. Bick, R. Haber, and D. Santos. "Floresta sintá(c)tica": A treebank for Portuguese. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pages 1698–1703, 2002.

[3] R.K. Ando and T. Zhang. A high-performance semi-supervised learning method for text chunking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[4] N. B. Atalay, K. Oflazer, and B. Say. The annotation process in the Turkish Treebank. In *Proceedings of the 4th International Workshop on Linguistically Interpreteted Corpora (LINC)*, 2003.

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

[6] Dan Bikel. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models*. PhD thesis, University of Pennsylvania, 2004.

[7] D.M. Bikel. Intricacies of Collins parsing model. *Computational Linguistics*, 2004.

[8] J. Blitzer, R. McDonald, and F. Pereira. Doman adaptation with structural correspondence learning. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2006.

[9] A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. The PDT: a 3-level annotation scenario. In Abeillé [1], chapter 7.

[10] B.E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings COLT*, pages 144–152, 1992.

[11] S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. The TIGER treebank. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, 2002.

[12] L. Breiman. Random forests. *Machine Learning*, 1(45), 2001.

[13] S. Buchholz, E. Marsi, A. Dubey, and Y. Krymolowski. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2006.

[14] P. M. Camerini, L. Fratta, and F. Maffioli. The k best spanning arborescences of a network. *Networks*, 10(2):91–110, 1980.

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

[16] E. Charniak. A maximum-entropy-inspired parser. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics (ACL)*, 2000.

[17] E. Charniak and M. Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[18] K. Chen and C. Huang. The sinica corpus.

[19] K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. Sinica Treebank: Design criteria, representational issues and implementation. In Abeillé [1], chapter 13, pages 231–248.

[20] D.M. Chickering, D. Geiger, and D. Heckerman. Learning bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, 1994.

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

[22] M. Civit, M^{a} A. Martí, B. Navarro, N. Bufi, B. Fernández, and R. Marcos. Issues in the syntactic annotation of Cast3LB. In *Proceedings of the 4th International Workshop on Linguistically Interpreteted Corpora (LINC)*, 2003.

[23] M. Civit Torruella and M^{a} A. Martí Antonín. Design principles for a Spanish treebank. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, 2002.

[24] S. Clark and J.R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

[25] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[26] M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[27] M. Collins and J. Brooks. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, 1995.

[28] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[29] M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann. A statistical parser for Czech. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1999.

[30] M. Collins and B. Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

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

[32] S. Corston-Oliver, A. Aue, K. Duh, and E. Ringger. Multilingual dependency parsing using bayes point machines. In *Proceedings of the Joint Conference on Human Language Technology and North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2006.

[33] K. Crammer, O. Dekel, J. Keshat, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. *Journal of Machine Learning Research*, 2006.

[34] K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2003.

[35] K. Crammer, R. McDonald, and F. Pereira. Scalable large-margin online learning for structured classification, 2005. Unpublished.

[36] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel based vector machines. *Journal of Machine Learning Research*, 2001.

[37] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 2003.

[38] A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

[39] H. Daumé and D. Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the International Conference on Machine Learning*, 2005.

[40] Y. Ding and M. Palmer. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[41] S. Džeroski, T. Erjavec, N. Ledinek, P. Pajas, Z. Žabokrtsky, and A. Žele. Towards a Slovene dependency treebank. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC)*, 2006.

[42] J. Early. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie Mellon University, 1968.

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

[44] J. Einarsson. Talbankens skriftspråkskonkordans, 1976.

[45] J. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 1996.

[46] J. Eisner and N. Smith. Parsing with soft and hard constraints on dependency length. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2005.

[47] D. Eppstein. Finding the k smallest spanning trees. In *2nd Scandanavian Workshop on Algorithm Theory*, 1990.

[48] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *Proceedings of the International Conference on Machine Learning*, 2005.

[49] R. Florian, H. Hassan, A. Ittycheriah, H. Jing, N. Kambhatla, X. Luo, N. Nicolov, S. Roukos, and T. Zhang. A statistical method for multilingual entity detection and tracking. In *Proceedings of the Joint Conference on Human Language Technology and North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2004.

[50] G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research: Special Issue on Variable and Feature Selection*, (3):1289–1305, 2003.

[51] K. Foth, W. Menzel, and I. Schröder. A transformation-based parsing technique with anytime properties. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2000.

[52] Y. Freund and R.E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[53] H. Gaifman. Dependency systems and phrase-structure systems. *Information and Control*, 1965.

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

[55] D. Gildea. Corpus variation and parser performance. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2001.

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

[57] J. Hajič, E. Hajičová, P. Pajas, J. Panevova, P. Sgall, and B. Vidova Hladka. The Prague Dependency Treebank 1.0 CDROM, 2001. Linguistics Data Consortium Cat. No. LDC2001T10.

[58] J. Hajič, O. Smrž, P. Zemánek, J. Šnaidauf, and E. Beška. Prague Arabic Dependency Treebank: Development in data and tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pages 110–117, 2004.

[59] K. Hall and V. Nóvák. Corrective modeling for non-projective dependency parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2005.

[60] M. P. Harper and R. A. Helzerman. Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language*, 1995.

[61] D. G. Hays. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525, 1964.

[62] X. He, R. Zemel, and M. Carreira-Perpinan. Multiscale conditional random fields for image labelling. In *Proceedings of Conference on Vision and Pattern Recognition*, 2004.

[63] J. Henderson. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the Joint Conference on Human Language Technology and North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2003.

[64] J. Henderson and E. Brill. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 1999.

[65] H. Hirakawa. Semantic dependency analysis method for Japanese based on optimum tree search algorithm. In *Proceedings of the Pacific Association for Computational Linguistics*, 2001.

[66] L. Huang and D. Chiang. Better k-best parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2005.

[67] R. Hudson. *Word Grammar*. Blackwell, 1984.

[68] R. Hwa, P. Resnik, A. Weinberg, C. Cabezas, and O. Kolak. Bootstrapping parsers via syntactic projection across parallel texts. *Special Issue of the Journal of Natural Language Engineering on Parallel Texts*, 11(3):311–325, 2005.

[69] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.

[70] A.K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? *Natural Language Parsing*, 1985.

[71] S. Kahane, A. Nasr, and O Rambow. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1998.

[72] R. Kassel. *A comparison of approaches to on-line character handwritten digit recognition*. PhD thesis, MIT Spoken Language Systems Group, 1995.

[73] Y. Kawata and J. Bartels. Stylebook for the Japanese Treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen, 2000.

[74] M. Kay. Experiments with a powerful parser. In *Proceedings 2eme Conference Internationale sue le Traitement Automatique des Languages*, 1967.

[75] D. Klein. *The Unsupervised Learning of Natural Language Structure*. PhD thesis, Stanford University, 2004.

[76] K. Knight and D. Marcu. Statistical-based summarization - step one: Sentence compression. In *Proceedings the American Association of Artificial Intelligence*, 2000.

[77] T. Koo and M. Collins. Hidden-variable models for discriminative reranking. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[78] M. T. Kromann. Optimaility parsing and local cost functions in discontinuous grammars. In *Proceedings of Joint Conference on Formal Grammars and the Mathematics of Language*, 2001.

[79] M. T. Kromann. The Danish Dependency Treebank and the underlying linguistic theory. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, 2003.

[80] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, 2001.

[81] M. Lease and E. Charniak. Parsing biomedical literature. In *Proceedings of the International Joint Conference on Natural Language Processing*, 2005.

[82] D. Lin. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*, 1998.

[83] D.M. Magerman. Statistical decision-tree models for parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1995.

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

[85] H. Maruyama. Structural disambiguation with constraint propagation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1990.

[86] A. McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2003.

[87] A. K. McCallum. MALLET: A machine learning for language toolkit, 2002. http://mallet.cs.umass.edu.

[88] D. McCloskly, E. Charniak, and M. Johnson. Effective self-training for parsing. In *Proceedings of the Joint Conference on Human Language Technology and North*

American Chapter of the Association for Computational Linguistics (HLT/NAACL), 2006.

[89] D. McCloskly, E. Charniak, and M. Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2006.

[90] R. McDonald, K. Crammer, and F. Pereira. Flexible text segmentation with structured multilabel classification. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[91] R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[92] R. McDonald, K. Lerman, and F. Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2006.

[93] R. McDonald and F. Pereira. Identifying gene and protein mentions in text using conditional random fields. *BMC Bioinformatics*, 6:Supp1(S6), 2005.

[94] R. McDonald and F. Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of the Annual Meeting of the European American Chapter of the Association for Computational Linguistics (ACL)*, 2006.

[95] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[96] I.A. Meĺčuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, 1988.

[97] R. Moore. A discriminative framework for bilingual word alignment. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[98] K.R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201, 2001.

[99] B. Navarro, M. Civit, M^{a} A. Martí, R. Marcos, and B. Fernández. Syntactic, semantic and pragmatic annotation in Cast3LB. In *Proceedings of the Workshop on Shallow Processing of Large Corpora (SProLaC)*, 2003.

[100] The Nordic Treebank Network. http://w3.msi.vxu.se/˜nivre/research/nt.html.

[101] P. Neuhaus and N. Böker. The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1997.

[102] J. Nilsson, J. Hall, and J. Nivre. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proceedings of the NODALIDA Special Session on Treebanks*, 2005.

[103] J. Nivre. Dependency grammar and dependency parsing. Technical Report MSI report 05133, Vxj University: School of Mathematics and Systems Engineering, 2005.

[104] J. Nivre and J. Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[105] J. Nivre and M. Scholz. Deterministic dependency parsing of english text. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2004.

[106] Joakim Nivre. Penn2malt, 2004. http://w3.msi.vxu.se/˜nivre/research/Penn2Malt.html.

[107] K. Oflazer, B. Say, D. Zeynep Hakkani-Tür, and G. Tür. Building a Turkish treebank. In Abeillé [1], chapter 15.

[108] P. Osenova and K. Simov. BTB-TR05: BulTreeBank stylebook. BulTreeBank version 1.0. Bultreebank project technical report, 2004. Available at: http://www.bultreebank.org/TechRep/BTB-TR05.pdf.

[109] PennBioIE. Mining The Bibliome Project, 2005. http://bioie.ldc.upenn.edu/.

[110] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142, 1996.

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

[112] K. Ribarov. *Automatic building of a dependency tree*. PhD thesis, Charles University, 2004.

[113] S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[114] S. Riezler, T. H. King, R. Crouch, and A. Zaenen. Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for lexical-

functional grammar. In *Proceedings of the Joint Conference on Human Language Technology and North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2003.

[115] B. Roark, M. Saraclar, M. Collins, and M. Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

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

[117] K. Sagae and A. Lavie. Parser combination by reparsing. In *Proceedings of the Joint Conference on Human Language Technology and North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2006.

[118] S. Sarawagi and W. Cohen. Semi-Markov conditional random fields for information extraction. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2004.

[119] P. Sgall, E. Hajičová, and J. Panevová. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel, 1986.

[120] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the Joint Conference on Human Language Technology and North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, pages 213–220, 2003.

[121] L. Shen and A. Joshi. Incremental LTAG parsing. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[122] Y. Shinyama, S. Sekine, K. Sudo, and R. Grishman. Automatic paraphrase acquisition from news articles. In *Proceedings of the Human Language Technology Conference (HLT)*, 2002.

[123] K. Simov and P. Osenova. Practical annotation scheme for an HPSG treebank of Bulgarian. In *Proceedings of the 4th International Workshop on Linguistically Interpreteted Corpora (LINC)*, pages 17–24, 2003.

[124] K. Simov, P. Osenova, A. Simov, and M. Kouylekov. Design and implementation of the Bulgarian HPSG-based treebank. In *Journal of Research on Language and Computation – Special Issue*, pages 495–522. Kluwer Academic Publishers, 2005.

[125] K. Simov, P. Osenova, and M. Slavcheva. BTB-TR03: BulTreeBank morphosyntactic tagset. BTB-TS version 2.0. Bultreebank project technical report, 2004. Available at: http://www.bultreebank.org/TechRep/BTB-TR03.pdf.

[126] K. Simov, G. Popova, and P. Osenova. HPSG-based syntactic treebank of Bulgarian (BulTreeBank). In A. Wilson, P. Rayson, and T. McEnery, editors, *A Rainbow of Corpora: Corpus Linguistics and the Languages of the World*, pages 135–142. Lincom-Europa, Munich, 2002.

[127] D. Sleator and D. Temperley. Parsing English with a link grammar. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 1993.

[128] N. Smith and J. Eisner. Guiding unsupervised grammar induction using contrastive estimation. In *Working Notes of the International Joint Conference on Artificial Intelligence Workshop on Grammatical Inference Applications*, 2005.

[129] O. Smrž, J. Šnaidauf, and P. Zemánek. Prague Dependency Treebank for Arabic: Multi-level annotation of Arabic corpus. In *Proceedings of the International Symposium on Processing of Arabic*, pages 147–155, 2002.

[130] R. Snow, D. Jurafsky, and A. Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2004.

[131] M. Steedman. *The Syntactic Process*. MIT Press, 2000.

[132] C. Sutton, C. Pal, and A. McCallum. Reducing weight undertraining in structured discriminative learning. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*.

[133] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[134] P. Tapanainen and T. Järvinen. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, 1997.

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

[136] B. Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford, 2004.

[137] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2003.

[138] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[139] U. Teleman. *Manual för grammatisk beskrivning av talad och skriven svenska (MAMBA)*, 1974.

[140] L. Tesnière. *Éléments de syntaxe structurale*. Editions Klincksieck, 1959.

[141] E.F. Tjong Kim Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2000.

[142] E.F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2003.

[143] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning*, 2004.

[144] J. Turian and D. Melamed. Constituent parsing by classification. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2005.

[145] J. Turian and D. Melamed. Advances in discriminative parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2006.

[146] J. Turner and E. Charniak. Supervised and unsupervised learning for sentence compression. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[147] L. van der Beek, G. Bouma, J. Daciuk, T. Gaustad, R. Malouf, G. van Noord, R. Prins, and B. Villada. The Alpino dependency treebank. In *Algorithms for Linguistic Processing*, NWO PIONIER progress report 5. 2002.

[148] L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*, 2002.

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

[150] N. Xue, F. Xia, F. Chiou, and M. Palmer. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 2004.

[151] H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2003.

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

[153] D. Zelenko, C. Aone, and A. Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106, 2003.

[154] D. Zeman. *Parsing with a Statistical Dependency Model*. PhD thesis, Univerzita Karlova, Praha, 2004.