Introduction
to
Data-Driven Dependency Parsing

Venue: European Summer School in Logic Language and Information 2007

Instructors:

- Ryan McDonald, Google Inc. (ryanmcd@google.com)
- Joakim Nivre, Växjö University and Uppsala University (nivre@msi.vxu.edu)

Webpage: http://dp.esslli07.googlepages.com/

Desription: Syntactic dependency representations of sentences have a long history in theoretical linguistics. Recently, they have found renewed interest in the computational parsing community due to their efficient computational properties and their ability to naturally model non-nested constructions, which is important in freer-word order languages such as Czech, Dutch, and German. This interest has led to a rapid growth in multilingual data sets and new parsing techniques. One modern approach to building dependency parsers, called data-driven dependency parsing, is to learn good and bad parsing decisions solely from labeled data, without the intervention of an underlying grammar. This course will cover: Dependency parsing (history, definitions, motivation, etc.), grammar-driven versus data-driven parsing, brief introduction to learning frameworks, transition-based parsing algorithms, graph-based parsing algorithms, other algorithms, empirical results and applications, and available software.

This reader contains four papers that form the basis of the course:

- Nivre, J. (2005) Dependency Grammar and Dependency Parsing. MSI report 05133. Vxj University: School of Mathematics and Systems Engineerin
- Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S. and Marsi, E. (2007) MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, 13(2), 95-135.
- McDonald, R., Pereira, F., Crammer, K., and Lerman, K. (2007) Global Inference and Learning Algorithms for Multi-Lingual Dependency Parsing. Unpublished manuscript.
- McDonald, R., and Nivre, J. (2007) Characterizing the Errors of Data-Driven Dependency Parsing Models. Proceedings of the Conference on Empirical Methods in Natural Language Processing and Natural Language Learning.

This reader is to serve as a reference guide for participants. Lecture slides will be available on the official course webpage.

Dependency Grammar and Dependency Parsing

Joakim Nivre

1 Introduction

Despite a long and venerable tradition in descriptive linguistics, dependency grammar has until recently played a fairly marginal role both in theoretical linguistics and in natural language processing. The increasing interest in dependency-based representations in natural language parsing in recent years appears to be motivated both by the potential usefulness of bilexical relations in disambiguation and by the gains in efficiency that result from the more constrained parsing problem for these representations.

In this paper, we will review the state of the art in dependency-based parsing, starting with the theoretical foundations of dependency grammar and moving on to consider both grammar-driven and data-driven methods for dependency parsing. We will limit our attention to systems for dependency parsing in a narrow sense, i.e. systems where the analysis assigned to an input sentence takes the form of a dependency structure. This means that we will not discuss systems that exploit dependency relations for the construction of another type of representation, such as the head-driven parsing models of Collins (1997, 1999). Moreover, we will restrict ourselves to systems for full parsing, which means that we will not deal with systems that produce a partial or underspecified representation of dependency structure, such as Constraint Grammar parsers (Karlsson, 1990; Karlsson et al., 1995).

2 Dependency Grammar

Although its roots may be traced back to Pāṇini's grammar of Sanskrit several centuries before the Common Era (Kruijff, 2002) and to medieval theories of grammar (Covington, 1984), dependency grammar has largely developed as a form for syntactic representation used by traditional grammarians, especially in Europe, and particularly in Classical and Slavic domains (Mel'čuk, 1988). This grammatical tradition can be said to culminate with the seminal work of Tesnière (1959), which

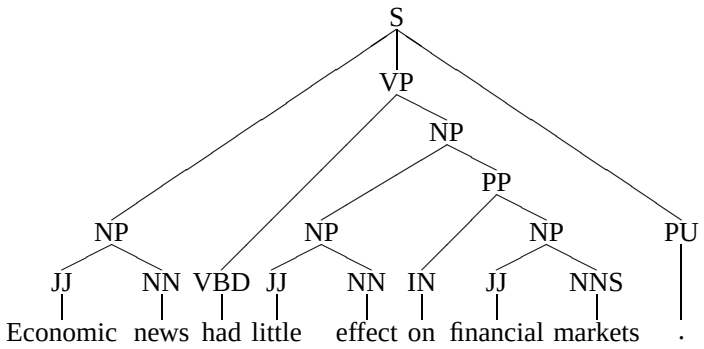


Figure 1: Constituent structure for English sentence from the Penn Treebank

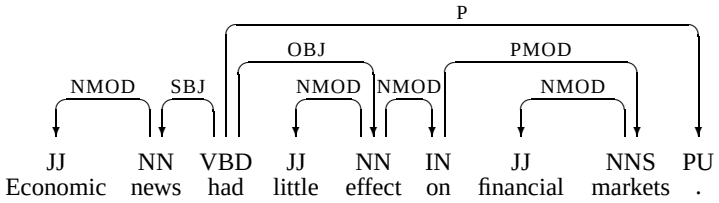


Figure 2: Dependency structure for English sentence from the Penn Treebank

is usually taken as the starting point of the modern theoretical tradition of dependency grammar.

This tradition comprises a large and fairly diverse family of grammatical theories and formalisms that share certain basic assumptions about syntactic structure, in particular the assumption that syntactic structure consists of *lexical* elements linked by binary asymmetrical relations called *dependencies*. Thus, the common formal property of dependency structures, as compared to representations based on constituency is the lack of phrasal nodes. This can be seen by comparing the constituency representation of an English sentence in Figure 1, taken from the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993, 1994), to the corresponding dependency representation in Figure 2.

Among the more well-known theories of dependency grammar, besides the theory of structural syntax developed by Tesnière (1959), we find Word Gram-

mar (WG) (Hudson, 1984, 1990), Functional Generative Description (FGD) (Sgall et al., 1986), Dependency Unification Grammar (DUG) (Hellwig, 1986, 2003), Meaning-Text Theory (MTT) (Mel'čuk, 1988), and Lexicase (Starosta, 1988). In addition, constraint-based theories of dependency grammar have a strong tradition, represented by Constraint Dependency Grammar (CDG) (Maruyama, 1990; Harper and Helzerman, 1995; Menzel and Schröder, 1998) and its descendant Weighted Constraint Dependency Grammar (WCDG) (Schröder, 2002), Functional Dependency Grammar (FDG) (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998), largely developed from Constraint Grammar (CG) (Karlsson, 1990; Karlsson et al., 1995), and finally Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001), subsequently evolved into Extensible Dependency Grammar (XDG) (Debusmann et al., 2004). A synthesis of dependency grammar and categorial grammar is found in the framework of Dependency Grammar Logic (DGL) (Kruijff, 2001).

We will make no attempt at reviewing all these theories here. Instead, we will try to characterize their common core of assumptions, centered upon the notion of dependency, and discuss major points of divergence, such as the issue of projective versus non-projective representations.

2.1 The Notion of Dependency

The fundamental notion of *dependency* is based on the idea that the syntactic structure of a sentence consists of binary asymmetrical relations between the words of the sentence. The idea is expressed in the following way in the opening chapters of Tesnière (1959):

> La phrase est un *ensemble organisé* dont les éléments constituants sont les *mots*. [1.2] Tout mot qui fait partie d'une phrase cesse par lui-même d'être isolé comme dans le dictionnaire. Entre lui et ses voisins, l'esprit aperçoit des *connexions*, dont l'ensemble forme la charpente de la phrase. [1.3] Les connexions structurales établissent entre les mots des rapports de *dépendance*. Chaque connexion unit en principe un terme *supérieur* à un terme *inférieur*. [2.1] Le terme supérieur reçoit le nom de *régissant*. Le terme inférieur reçoit le nom de *subordonné*. Ainsi dans la phrase *Alfred parle* [...], *parle* est le régissant et *Alfred* le subordonné. [2.2] (Tesnière, 1959, 11–13, emphasis in the original)[1]

[1] English translation (by the author): 'The sentence is an *organized whole*, the constituent elements of which are *words*. [1.2] 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

In the terminology used in this paper, a dependency relation holds between a *head* and a *dependent*. Alternative terms in the literature are *governor* and *regent* for *head* (cf. Tesnière's *régissant*) and *modifier* for *dependent* (cf. Tesnière's *subordonné*).

Criteria for establishing dependency relations, and for distinguishing the head and the dependent in such relations, are clearly of central importance for dependency grammar. Such criteria have been discussed not only in the dependency grammar tradition, but also within other frameworks where the notion of syntactic head plays an important role, including all constituency-based frameworks that subscribe to some version of $\overline{\text{X}}$ theory (Chomsky, 1970; Jackendoff, 1977). Here are some of the criteria that have been proposed for identifying a syntactic relation between a head H and a dependent D in a construction C (Zwicky, 1985; Hudson, 1990):

1. H determines the syntactic category of C and can often replace C.
2. H determines the semantic category of C; D gives semantic specification.
3. H is obligatory; D may be optional.
4. H selects D and determines whether D is obligatory or optional.
5. The form of D depends on H (agreement or government).
6. The linear position of D is specified with reference to H.

It is clear that this list contains a mix of different criteria, some syntactic and some semantic, and one may ask whether there is a single coherent notion of dependency corresponding to all the different criteria. This has led some theorists, such as Hudson (1990), to suggest that the concept of head has a prototype structure, i.e. that typical instances of this category satisfy all or most of the criteria while more peripheral instances satisfy fewer. Other authors have emphasized the need to distinguish different kinds of dependency relations. According to Mel'čuk (1988), the word forms of a sentence can be linked by three types of dependencies: *morphological*, *syntactic* and *semantic*. According to Nikula (1986), we must distinguish between syntactic dependency in *endocentric* and *exocentric* constructions (Bloomfield, 1933).

---

of which forms the structure of the sentence. [1.3] The structural connections establish *dependency* relations between the words. Each connection in principle unites a *superior* term and an *inferior* term. [2.1] The superior term receives the name *governor*. The inferior term receives the name *subordinate*. Thus, in the sentence *Alfred parle* [...], *parle* is the governor and *Alfred* the subordinate. [2.2]'

Thus, in Figure 2, the NMOD relation holding between the noun *markets* and the adjective *financial* is an endocentric construction, where the head can replace the whole without disrupting the syntactic structure:

Economic news had little effect on [financial] markets. (1)

Endocentric constructions may satisfy all the criteria listed above, although number 4 is usually considered less relevant, since dependents in endocentric constructions are taken to be optional and not selected by their heads. By contrast, the PMOD relation holding between the preposition *on* and the noun *markets* is an exocentric construction, where the head cannot readily replace the whole:

Economic news had little effect on [markets]. (2)

Exocentric constructions, by their definition, fail on criterion number 1, at least with respect to subsitutability of the head for the whole, but they may satisfy the remaining criteria. Considering the rest of the relations exemplified in Figure 2, the SBJ and OBJ relations are clearly exocentric, and the NMOD relation from the noun *news* to the adjective *Economic* clearly endocentric, while the remaining NMOD relations (effect → little, effect → on) have a more unclear status.

The distinction between endocentric and exocentric constructions is also related to the distinction between *head-complement* and *head-modifier* (or *head-adjunct*) relations found in many contemporary syntactic theories, since head-complement relations are exocentric while head-modifier relations are endocentric. Many theories also recognize a third kind of relation, the *head-specifier* relation, typically exemplified by the determiner-noun relation, which is exocentric like the head-complement relation, but where there is no clear selection of the dependent element by the head.

The distinction between complements and modifiers is often defined in terms of *valency*, which is a central notion in the theoretical tradition of dependency grammar. Although the exact characterization of this notion differs from one theoretical framework to the other, valency is usually related to the semantic predicate-argument structure associated with certain classes of lexemes, in particular verbs but sometimes also nouns and adjectives. The idea is that the verb imposes requirements on its syntactic dependents that reflect its interpretation as a semantic predicate. Dependents that correspond to arguments of the predicate can be obligatory or optional in surface syntax but can only occur once with each predicate instance. By contrast, dependents that do not correspond to arguments can have more than one occurrence with a single predicate instance and tend to be optional. The *valency frame* of the verb is normally taken to include argument dependents, but some theories also allow obligatory non-arguments to be included (Sgall et al., 1986).

The notion of valency will not play a central role in the present paper, but we will sometimes use the terms *valency-bound* and *valency-free* to make a rough distinction between dependents that are more or less closely related to the semantic interpretation of the head. Returning to Figure 2, the subject and the object would normally be treated as valency-bound dependents of the verb *had*, while the adjectival modifiers of the nouns *news* and *markets* would be considered valency-free. The prepositional modification of the noun *effect* may or may not be treated as valency-bound, depending on whether the entity undergoing the effect is supposed to be an argument of the noun *effect* or not.

While head-complement and head-modifier structures have a fairly straightforward analysis in dependency grammar, there are also many constructions that have a relatively unclear status. This group includes constructions that involve grammatical function words, such as articles, complementizers and auxiliary verbs, but also structures involving prepositional phrases. For these constructions, there is no general consensus in the tradition of dependency grammar as to whether they should be analyzed as head-dependent relations at all and, if so, what should be regarded as the head and what should be regarded as the dependent. For example, some theories regard auxiliary verbs as heads taking lexical verbs as dependents; other theories make the opposite assumption; and yet other theories assume that verb chains are connected by relations that are not dependencies in the usual sense.

Another kind of construction that is problematic for dependency grammar (as for most theoretical traditions) is *coordination*. According to Bloomfield (1933), coordination is an endocentric construction, since it contains not only one but several heads that can replace the whole construction syntactically. However, this characterization raises the question of whether coordination can be analyzed in terms of binary asymmetrical relations holding between a head and a dependent. Again, this question has been answered in different ways by different theories within the dependency grammar tradition.

In conclusion, the theoretical tradition of dependency grammar is united by the assumption that an essential part of the syntactic structure of sentences resides in binary asymmetrical relations holding between lexical elements. Moreover, there is a core of syntactic constructions for which the analysis given by different frameworks agree in all important respects. However, there are also important differences with respect to whether dependency analysis is assumed to exhaust syntactic analysis, and with respect to the analysis of certain types of syntactic constructions. We will now turn to a discussion of some of the more important points of divergence in this tradition.

2.2 Varieties of Dependency Grammar

Perhaps the most fundamental open question in the tradition of dependency grammar is whether the notion of dependency is assumed to be not only *necessary* but also *sufficient* for the analysis of syntactic structure in natural language. This assumption is not made in the theory of Tesnière (1959), which is based on the three complementary concepts of *connection* (connexion), *junction* (jonction) and *transfer* (translation), where connection corresponds to dependency (cf. the quotation on page 3) but where junction and transfer are other kinds of relations that can hold between the words of a sentence. More precisely, junction is the relation that holds between coordinated items that are dependents of the same head or heads of the same dependent, while transfer is the relation that holds between a function word or other element that changes the syntactic category of a lexical element so that it can enter into different dependency relations. An example of the latter is the relation holding between the preposition *de* and *Pierre* in the construction *le livre de Pierre* (Pierre's book), where the preposition *de* allows the proper name *Pierre* to modify a noun, a dependency relation otherwise reserved for adjectives. Another way in which theories may depart from a pure dependency analysis is to allow a restricted form of constituency analysis, so that dependencies can hold between strings of words rather than single words. This possibility is exploited, to different degrees, in the frameworks of Hellwig (1986, 2003), Mel'čuk (1988) and Hudson (1990), notably in connection with coordination.

A second dividing line is that between mono-stratal and multi-stratal frameworks, i.e. between theories that rely on a single syntactic representation and theories that posit several layers of representation. In fact, most theories of dependency grammar, in contrast to frameworks for dependency parsing that will be discussed in Section 3, are multi-stratal, at least if semantic representations are considered to be a stratum of the theory. Thus, in FGD (Sgall et al., 1986) there is both an *analytical* layer, which can be characterized as a surface syntactic representation, and a *tectogrammatical* layer, which can be regarded as a deep syntactic (or shallow semantic) representation. In a similar fashion, MTT (Mel'čuk, 1988) recognizes both *surface syntactic* and *deep syntactic* representations (in addition to representations of deep phonetics, surface morphology, deep morphology and semantics). By contrast, Tesnière (1959) uses a single level of syntactic representation, the so-called *stemma*, which on the other hand includes junction and transfer in addition to syntactic connection.[2] The framework of XDG (Debusmann et al., 2004) can be seen as a compromise in that it allows multiple layers of dependency-based linguistic representations but requires that all layers, or *dimensions* as they are called in

[2] Tesnière's representations also include *anaphors*, which are described as supplementary semantic connections without corresponding syntactic connections.

XDG, share the same set of nodes. This is in contrast to theories like FGD, where e.g. function words are present in the analytical layer but not in the tectogrammatical layer.

The different requirements of XDG and FGD point to another issue, namely what can constitute a node in a dependency structure. Although most theories agree that dependency relations hold between *lexical* elements, rather than *phrases*, they can make different assumptions about the nature of these elements. The most straightforward view is that the nodes of the dependency structure are simply the word forms occurring in the sentence, which is the view adopted in most parsing systems based on dependency grammar. But it is also possible to construct dependency structures involving more abstract entities, such as lemmas or lexemes, especially in deeper syntactic representations. Another variation is that the elements may involve several word forms, constituting a *dissociate nucleus* (nucléus dissocié) in the terminology of Tesnière (1959), or alternatively correspond to smaller units than word forms, as in the morphological dependencies of Mel'čuk (1988).

A fourth dimension of variation concerns the inventory of specific dependency types posited by different theories, i.e. functional categories like SBJ, OBJ and NMOD that are used to label dependency arcs in the representation in Figure 2. Broadly speaking, most theories either assume a set of more surface-oriented grammatical functions, such as *subject*, *object*, *adverbial*, etc., with a more or less elaborate subclassification, or a set of more semantically oriented role types, such as *agent*, *patient*, *goal*, etc., belonging to the tradition of *case roles* or *thematic roles* (Fillmore, 1968; Jackendoff, 1972; Dowty, 1989).[3] Multi-stratal theories often combine the two relation types. Thus, FGD (Sgall et al., 1986) uses grammatical functions in the analytical layer and semantic roles in the tectogrammatical layer. An alternative scheme of representation, which is found in MTT (Mel'čuk, 1988), is to use numerical indices for valency-bound dependents to reflect a canonical ordering of arguments (argument 1, 2, 3, etc.) and to use descriptive labels only for valency-free dependents. Finally, it is also possible to use unlabeled dependency structures, although this is more common in practical parsing systems than in linguistic theories.

There are several open issues in dependency grammar that have to do with formal properties of representations. Since a dependency representation consists of lexical elements linked by binary asymmetrical relations, it can be defined as a *labeled directed graph*, where the set of nodes (or vertices) is the set of lexical elements (as defined by the particular framework), and the set of labeled arcs

[3]The notion of a semantic role can be traced back to Pāṇini's *kānaka* theory (Misra, 1966), which is sometimes also seen as the earliest manifestation of dependency grammar. The notion of a grammatical function also has a long history that extends at least to the work of Appolonius Dyscolus in the second century of the Common Era (Robins, 1967).

represent dependency relations from heads to dependents. In order to provide a complete syntactic analysis of a sentence, the graph must also be *connected* so that every node is related to at least one other node (Mel'čuk, 1988). Again, we refer to Figure 2 as an illustration of this representation, where the nodes are the word tokens of the sentence (annotated with parts-of-speech) and the arcs are labeled with grammatical functions.[4]

Given this general characterization, we may then impose various additional conditions on these graphs. Two basic constraints that are assumed in most versions of dependency grammar are the *single-head* constraint, i.e. the assumption that each node has at most one head, and the *acyclicity* constraint, i.e. the assumption that the graph should not contain cycles. These two constraints, together with connectedness, imply that the graph should be a rooted tree, with a single root node that is not a dependent of any other node. For example, the representation in Figure 2 is a rooted tree with the verb *had* as the root node. Although these constraints are assumed in most versions of dependency grammar, there are also frameworks that allow multiple heads as well as cyclic graphs, such as WG (Hudson, 1984, 1990). Another issue that arises for multi-stratal theories is whether each level of representation has its own set of nodes, as in most theories, or whether they only define different arc sets on top of the same set of nodes, as in XDG (Debusmann et al., 2004).

However, the most important and hotly debated issues concerning formal representations have to do with the relation between dependency structure and word order. According to Tesnière (1959), dependency relations belong to the *structural order* (l'ordre structural), which is different from the *linear order* (l'ordre linéaire) of a spoken or written string of words, and *structural syntax* is based on the relations that exist between these two dimensions. Most versions of dependency grammar follow Tesnière in assuming that the nodes of a dependency structure are not linearly ordered in themselves but only in relation to a particular surface realization of this structure. A notable exception to this generalization is FGD, where the representations of both the analytical layer and the tectogrammatical layer are linearly ordered in order to capture aspects of information structure (Sgall et al., 1986). In addition, there are frameworks, such as TDG (Duchier and Debusmann, 2001), where the linear order is represented by means of a linearly ordered dependency structure, the Linear Precedence (LP) tree, while the proper dependency representation, the Immediate Dominance (ID) tree, is unordered.

[4]There seems to be no general consensus in the literature on dependency grammar as to whether the arcs representing dependency relations should be drawn pointing from heads to dependents or vice versa (or indeed with arrowheads at all). We have chosen to adopt the former alternative, both because it seems to be the most common representation in the literature and because it is consistent with standard practice in graph theory.

However, whether dependency relations introduce a linear ordering or not, there may be constraints relating dependency structures to linear realizations. The most well-known example is the constraint of *projectivity*, first discussed by Lecerf (1960), Hays (1964) and Marcus (1965), which is related to the contiguity constraint for constituent representations. A dependency graph satisfies the constraint of projectivity with respect to a particular linear order of the nodes if, for every arc $h \rightarrow d$ and node w, w occurs between h and d in the linear order only if w is dominated by h (where *dominates* is the reflexive and transitive closure of the arc relation). For example, the representation in Figure 2 is an example of a *projective* dependency graph, given the linear order imposed by the word order of the sentence.

The distinction between *projective* and *non-projective* dependency grammar often made in the literature thus refers to the issue of whether this constraint is assumed or not. Broadly speaking, we can say that whereas most practical systems for dependency parsing do assume projectivity, most dependency-based linguistic theories do not. More precisely, most theoretical formulations of dependency grammar regard projectivity as the norm but also recognize the need for non-projective representations of certain linguistic constructions, e.g. long-distance dependencies (Mel'čuk, 1988; Hudson, 1990). It is also often assumed that the constraint of projectivity is too rigid for the description of languages with free or flexible word order.

Some multi-stratal theories allow non-projective representations in some layers but not in others. For example, FGD assumes that tectogrammatical representations are projective while analytical representations are not (Sgall et al., 1986). Similarly, TDG (Duchier and Debusmann, 2001) assume projectivity for LP trees but not for ID trees. Sometimes a weaker condition called *planarity* is assumed, which allows a node w to occur between a head h and a dependent d without being dominated by h only if w is a root (Sleator and Temperley, 1993).[5] Further relaxations of these constraints are discussed in Kahane et al. (1998) and Yli-Jyrä (2003).

The points of divergence considered up till now have all been concerned with aspects of representation. However, as mentioned at the end of the previous section, there are also a number of points concerning the substantive linguistic analysis where different frameworks of dependency grammar make different assumptions, in the same way as theories differ also within other traditions. We will limit ourselves to a brief discussion of two such points.

The first point concerns the issue of *syntactic* versus *semantic* heads. As noted in Section 2.1, the criteria for identifying heads and dependents invoke both syn-

[5] This constraint is related to but not equivalent to the standard notion of planarity in graph theory (see, e.g., Grimaldi, 2004).

tactic and semantic properties. In many cases, these criteria give the same result, but in others they diverge. A typical example is found in so-called case marking prepositions, exemplified in the following sentence:

I believe in the system. (3)

According to syntactic criteria, it is natural to treat the preposition *in* as a dependent of the verb *believe* and as the head of the noun *system*. According to semantic criteria, it is more natural to regard *system* as a direct dependent of *believe* and to treat *in* as a dependent of *system* (corresponding to a case marking affix in some other languages).[6] Most versions of dependency grammar treat the preposition as the head of the noun, but there are also theories that make the opposite assumption. Similar considerations apply to many constructions involving one function word and one content word, such as determiner-noun and complementizer-verb constructions. An elegant solution to this problem is provided by the theory of Tesnière (1959), according to which the function word and the content word form a *dissociate nucleus* (nucléus dissocié), united by a relation of *transfer* (translation). In multi-stratal theories, it is possible to treat the function word as the head only in more surface-oriented layers. For example, to return to example (3), FGD would assume that the preposition takes the noun as a dependent in the analytical layer, but in the tectogrammatical layer the preposition would be absent and the noun would instead depend directly on the verb.

The second point concerns the analysis of coordination, which presents problems for any syntactic theory but which seems to be especially hard to reconcile with the idea that syntactic constructions should be analyzed in terms of binary head-dependent relations. Consider the following example:

They operate ships and banks. (4)

It seems clear that the phrase *ships and banks* functions as a direct object of the verb *operate*, but it is not immediately clear how this phrase can be given an internal analysis that is compatible with the basic assumptions of dependency analysis, since the two nouns *ships* and *banks* seem to be equally good candidates for being heads. One alternative is to treat the conjunction as the head, as shown in Figure 3 (*top*), an analysis that may be motivated on semantic grounds and is adopted in FGD. Another alternative, advocated by Mel'čuk (1988), is to treat the conjunction as the head only of the second conjunct and analyze the conjunction as a dependent of the first conjunct, as shown in Figure 3 (*bottom*). The arguments for this analysis are essentially the same as the arguments for an asymmetric right-branching

[6] A third alternative is to treat both *in* and *system* as dependents of *believe*, since it is the verb that selects the preposition and takes the noun as an argument.

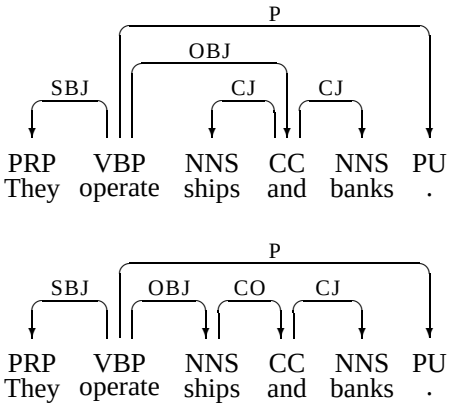


Figure 3: Two analyses of coordination

analysis in constituency-based frameworks. A third option is to give up a pure dependency analysis and allow a limited form of phrase structure, as in WG (Hudson, 1990). A fourth and final variant is the analysis of Tesnière (1959), according to which both *ships* and *banks* are dependents of the verb, while the conjunction marks a relation of *junction* (jonction) between the two nouns.

3 Parsing with Dependency Representations

So far, we have reviewed the theoretical tradition of dependency grammar, focusing on the common core of assumptions as well as major points of divergence, rather than on individual instantiations of this tradition. We will now turn to what is the main topic of this paper, namely the computational implementation of syntactic analysis based on dependency representations, i.e. representations involving lexical nodes, connected by dependency arcs, possibly labeled with dependency types.

Such implementations may be intimately tied to the development of a particular theory, such as the PLAIN system based on DUG (Hellwig, 1980, 2003) or the FDG parsing system (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998). On the whole, however, the connections between theoretical frameworks and computational systems are often rather indirect for dependency-based analysis, probably more so than for theories and parsers based on constituency analysis. This may be due to the relatively lower degree of formalization of dependency grammar theories in general, and this is also part of the reason why the topic of this section

is described as parsing with dependency *representations*, rather than parsing with dependency *grammar*.

In discussing dependency-based systems for syntactic parsing, we will follow Carroll (2000) and distinguish two broad types of strategy, the *grammar-driven approach* and the *data-driven approach*, although these approaches are not mutually exclusive. We will conclude the paper with a brief discussion of some of the potential advantages of using dependency representations in syntactic parsing.

3.1 Grammar-Driven Dependency Parsing

The earliest work on parsing with dependency representations was intimately tied to formalizations of dependency grammar that were very close to context-free grammar, such as the proposals of Hays (1964) and Gaifman (1965). In the formulation of Gaifman (1965) a *dependency system* contains three sets of rules:[7]

1. L_{I}: Rules of the form $X(Y_1 \cdots Y_i * Y_{i+1} \cdots Y_n)$, where i may equal 0 and/or n, which say that the category X may occur with categories $Y_1, \ldots, Y_n$ as dependents, in the order given (with X in position $*$).

2. L_{II}: Rules giving for every category X the list of words belonging to it (where each word may belong to more than one category).

3. L_{III}: A rule giving the list of all categories the occurrence of which may govern a sentence.

A sentence consisting of words $w_1, \ldots, w_n$ is analyzed by assigning to it a sequence of categories $X_1, \ldots, X_n$ and a relation of dependency d between words such that the following conditions hold (where d^* is the transitive closure of d):

1. For no w_i, $d^*(w_i, w_i)$.

2. For every w_i, there is at most one w_j such that $d(w_i, w_j)$.

3. If $d^*(w_i, w_j)$ and w_k is between w_i and w_j, then $d^*(w_k, w_j)$.

4. The whole set of word occurrences is connected by d.

5. If $w_1, \ldots, w_i$ are left dependents and $w_{i+1}, \ldots, w_n$ right dependents of some word, and $X_1, \ldots, X_i, X_{i+1}, \ldots, X_n$ are the categories of $w_1, \ldots, w_i, w_{i+1}$, $\ldots, w_n$, then $X(X_1 \cdots X_i * X_{i+1} \cdots X_n)$ is a rule of L_{I}.

6. The word occurrence w_i that governs the sentence belongs to a category listed in L_{III}.

[7] The formulation of Hays (1964) is slightly different but equivalent in all respects.

Gaifman remarks that 1–4 are general structure requirements that can be made on any relation defined on a finite linearly ordered set whether it is a set of categories or not, while 5–6 are requirements which relate the relation to the specific grammar given by the three sets of rules L_{I}–L_{III}. Referring back to the discussion of graph conditions in Section 2.2, we may first of all note that Gaifman defines dependency relations to hold from dependent to head, rather than the other way round which is more common in the recent literature. Secondly, we see that condition 2 corresponds to the *single-head* constraint and condition 3 to the *projectivity* constraint. Conditions 1, 2 and 4 jointly entail that the graph is a rooted tree, which is presupposed in condition 6. Finally, it should be pointed out that this kind of dependency system only gives an unlabeled dependency analysis, since there are no dependency types used to label dependency relations.

Gaifman (1965) proves several equivalence results relating his dependency systems to context-free grammars. In particular, he shows that the two systems are weakly equivalent, i.e. that they both characterize the class of context-free languages. However, he also shows that whereas any dependency system can be converted to a strongly equivalent context-free grammar (modulo a specific mapping between dependency trees and context-free parse trees), the inverse construction is only possible for a restricted subset of context-free grammar (roughly grammars where all productions are lexicalized).

These results have been invoked to explain the relative lack of interest in dependency grammar within natural language processing for the subsequent twenty-five years or so, based on the erroneous conclusion that dependency grammar is only a restricted variant of context-free grammar (Järvinen and Tapanainen, 1998).[8] This conclusion is erroneous simply because the results only concern the specific version of dependency grammar formalized by Hays and Gaifman, which for one thing is restricted to projective dependency structures. However, it also worth emphasizing that with the increasing importance of problems like robustness and disambiguation, issues of (limited) generative capacity have lost some of their significance in the context of natural language parsing. Nevertheless, it seems largely true to say that, except for isolated studies of dependency grammar as an alternative to context-free grammar as the basis for transformational grammar (Robinson, 1970), dependency grammar has played a marginal role both in syntactic theory and in natural language parsing until fairly recently.

The close relation to context-free grammar in the formalization of dependency grammar by Hays and Gaifman means that essentially the same parsing methods

[8] A similar development seems to have taken place with respect to categorial grammar after the weak equivalence of a restricted type of categorial grammar with context-free grammar was proven by Bar-Hillel et al. (1960).

can be used for both types of system. Hence, the parsing algorithm outlined in Hays (1964) is a bottom-up dynamic programming algorithm very similar to the CKY algorithm proposed for context-free parsing at about the same time (Kasami, 1965; Younger, 1967). The use of dynamic programming algorithms that are closely connected to context-free parsing algorithms such as CKY and Earley's algorithm (Earley, 1970) is a prominent trend also in more recent grammar-driven approaches to dependency parsing. One example is the link grammar parser of Sleator and Temperley (1991, 1993), which uses a dynamic programming algorithm implemented as a top-down recursive algorithm with memoization to achieve parsing in $O(n^3)$ time. Link grammar is not considered an instance of dependency grammar by its creators, and it departs from the traditional view of dependency by using undirected links, but the representations used in link grammar parsing are similar to dependency representations in that they consist of words linked by binary relations. Other examples include a modification of the CKY algorithm (Holan et al., 1997) and an Earley-type algorithm with left-corner filtering in the prediction step (Lombardo and Lesmo, 1996; Barbero et al., 1998).

A common property of all frameworks that implement dependency parsing as a form of lexicalized context-free parsing is that they are restricted to the derivation of projective dependency structures, although some of the frameworks allow post-processing that may introduce non-projective structures (Sleator and Temperley, 1991, 1993). Many of these frameworks can be subsumed under the notion of *bilexical grammar* introduced by Eisner (2000). In Eisner's formulation, a bilexical grammar consists of two elements:

1. A vocabulary V of terminal symbols (words), containing a distinguished symbol ROOT.

2. For each word $w \in V$, a pair of deterministic finite-state automata l_w and r_w. Each automaton accepts some regular subset of V^*.

The language $L(G)$ defined by a bilexical dependency grammar G is defined as follows:

1. A *dependency tree* is a rooted tree whose nodes are labeled with words from V, and where the root node is labeled with the special symbol ROOT. The children of a node are ordered with respect to each other and the node itself, so that the node has both *left children* that precede it and *right children* that follow it.

2. A dependency tree is *grammatical* according to G iff for every word token w that appears in the tree, l_w accepts the (possibly empty) sequence of w's

left children (from right to left), and r_w accepts the sequence of w's right children (from left to right).

3. A string $x \in V^*$ is generated by G with analysis y if y is a grammatical dependency tree according to G and listing the node labels of y in infix order yields the string x followed by ROOT; x is called the *yield* of y.

4. The language $L(G)$ is the set of all strings generated by G.

The general parsing algorithm proposed by Eisner for bilexical grammar is again a dynamic programming algorithm, which proceeds by linking *spans* (strings where roots occur either leftmost or rightmost or both) instead of *constituents*, thereby reducing the time complexity from $O(n^5)$ to $O(n^3)$. More precisely, the running time is $O(n^3g^3t)$, where g is an upper bound on the number of possible senses (lexical entries) of a single word, and t is an upper bound on the number of states of a single automaton.

Eisner shows how the framework of bilexical grammar, and the cubic-time parsing algorithm, can be modified to capture a number of different frameworks and approaches such as Milward's (mono)lexical dependency grammar (Milward, 1994), Alshawi's head automata (Alshawi, 1996), Sleator and Temperley's link grammar (Sleator and Temperley, 1991, 1993), and Eisner's own probabilistic dependency models that will be discussed below in Section 3.2 (Eisner, 1996b,a).

The second main tradition in grammar-driven dependency parsing is based on the notion of *eliminative* parsing, where sentences are analyzed by successively eliminating representations that violate constraints until only valid representations remain. One of the first parsing systems based on this idea is the CG framework (Karlsson, 1990; Karlsson et al., 1995), which uses underspecified dependency structures represented as syntactic tags and disambiguated by a set of constraints intended to exclude ill-formed analyses. In CDG (Maruyama, 1990), this idea is extended to complete dependency structures by generalizing the notion of tag to pairs consisting of a syntactic label and an identifier of the head node. This kind of representation is fundamental for many different approaches to dependency parsing, since it provides a way to reduce the parsing problem to a tagging or classification problem. Typical representatives of this tradition are the extended CDG framework of Harper and Helzerman (1995) and the FDG system (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998), where the latter is a development of CG that combines eliminative parsing with a non-projective dependency grammar inspired by Tesnière (1959).

In the eliminative approach, parsing is viewed as a constraint satisfaction problem, where any analysis satisfying all the constraints of the grammar is a valid

analysis. Constraint satisfaction in general is NP complete, which means that special care must be taken to ensure reasonable efficiency in practice. Early versions of this approach used procedures based on local consistency (Maruyama, 1990; Harper et al., 1995), which attain polynomial worst case complexity by only considering local information in the application of constraints. In the more recently developed TDG framework (Duchier, 1999, 2003), the problem is confronted head-on by using constraint programming to solve the satisfaction problem defined by the grammar for a given input string. The TDG framework also introduces several levels of representation (cf. Section 2.2), arguing that constraints can be simplified by isolating different aspects of the grammar such as Immediate Dominance (ID) and Linear Precedence (LP) and have constraints that relate different levels to each other (Duchier and Debusmann, 2001; Debusmann, 2001). This view is taken to its logical extension in the most recent version of the framework called Extensible Dependency Grammar (XDG), where any number of levels, or dimensions, can be defined in the grammatical framework (Debusmann et al., 2004)

From the point of view of parsing unrestricted text, parsing as constraint satisfaction can be problematic in two ways. First, for a given input string, there may be no analysis satisfying all constraints, which leads to a robustness problem. Secondly, there may be more than one analysis, which leads to a problem of disambiguation. Menzel and Schröder (1998) extends the CDG framework of Maruyama (1990) with *graded*, or *weighted*, constraints, by assigning a weight w ($0.0 \leq w \leq 1.0$) to each constraint indicating how serious the violation of this constraint is (where 0.0 is the most serious). In this extended framework, later developed into WCDG (Schröder, 2002), the best analysis for a given input string is the analysis that minimizes the total weight of violated constraints. While early implementations of this system used an eliminative approach to parsing (Menzel and Schröder, 1998), the more recent versions instead use a transformation-based approach, which successively tries to improve the analysis by transforming one solution into another guided by the observed constraint violations in the current solution. One advantage of this heuristic approximation strategy is that it can be combined with arbitrarily complex constraints, whereas standard eliminative procedures usually require constraints to be binary for efficiency reasons (Foth et al., 2004).

So far, we have distinguished two main trends in grammar-driven dependency parsing. The first is based on a formalization of dependency grammar that is closely related to context-free grammar, and therefore usually restricted to projective dependency structures, using standard techniques from context-free parsing to obtain good efficiency in the presence of massive ambiguity, in particular dynamic programming or memoization. The second is based on a formalization of dependency grammar in terms of constraints, not necessarily limited to projective structures,

where parsing is naturally viewed as a constraint satisfaction problem which can be addressed using eliminative parsing methods, although the exact parsing problem is often intractable.

In addition to these two traditions, which both involve fairly complex grammars and parsing algorithms, there is a third tradition which is based on a simpler notion of dependency grammar together with a deterministic parsing strategy (possibly with limited backtracking). As in other parsing paradigms, the study of deterministic parsing can be motivated either by a wish to model human sentence processing or by a desire to make syntactic parsing more efficient (or possibly both). According to Covington (2001), these methods have been known since the 1960's without being presented systematically in the literature. The fundamental parsing strategy comes in different versions but we will concentrate here on the left-to-right (or incremental) version, which is formulated in the following way by Covington (2001):

> Accept words one by one starting at the beginning of the sentence, and try linking each word as head or dependent of every previous word.

This parsing strategy is compatible with many different grammar formulations. All that is required is that a grammar G defines a boolean function f_G that, for any two words w_1 and w_2, returns **true** if w_1 can be the head of w_2 according to G (and **false**) otherwise.[9] Covington (2001) demonstrates how this parsing strategy can be used to produce dependency structures satisfying different conditions such as *uniqueness* (single head) and *projectivity* simply by imposing different constraints on the linking operation. Covington has also shown in previous work how this parsing strategy can be adapted to suit languages with free, flexible or rigid word order (Covington, 1990a,b, 1994). The time complexity of Covington's algorithm is $O(n^2)$ in the deterministic version.

The parsing algorithm proposed by Nivre (2003), which will be discussed in Section 3.2, can be derived as a special case of Covington's algorithm, although we will not give this formulation here, and the very first experiments with this algorithm used a simple grammar of the kind presupposed by Covington to perform unlabeled dependency parsing (Nivre, 2003). A similar approach can be found in Obrebski (2003), although this system is nondeterministic and derives a compact representation of all permissible dependency trees in the form of a directed acyclic graph. Yet another framework that shows affinities with the deterministic grammar-driven approach is that of Kromann (2004), although it is based on a

[9] In this formulation, the parsing strategy is limited to unlabeled dependency graphs. In principle, it is possible to perform labeled dependency parsing by returning a set of permissible dependency types instead of **true**, but this makes the process nondeterministic in general.

more sophisticated notion of grammar called Discontinuous Grammar. Parsing in this framework is essentially deterministic but subject to repair mechanisms that are associated with local cost functions derived from the grammar.

Before we close the discussion of grammar-driven dependency parsing, we should also mention the work of Oflazer (2003), which is an extended finite-state approach to dependency parsing similar to the cascaded partial parsers used for constituency-based parsing by Abney (1996) and Roche (1997). Oflazer's system allows violable constraints for robust parsing and uses total link length to rank alternative analyses, as proposed by Lin (1996).

3.2 Data-Driven Dependency Parsing

As for natural language parsing in general, the first attempts at data-driven dependency parsing were also grammar-driven in that they relied on a formal dependency grammar and used corpus data to induce a probabilistic model for disambiguation. Thus, Carroll and Charniak (1992) essentially use a PCFG model, where the context-free grammar is restricted to be equivalent to a Hays/Gaifman type dependency grammar. They report experiments trying to induce such a probabilistic grammar using unsupervised learning on an artificially created corpus but with relatively poor results.

A more successful and more influential approach was developed by Eisner (1996a,b), who defined several probabilistic models for dependency parsing and evaluated them using supervised learning with data from the Wall Street Journal section of the Penn Treebank. In later work, Eisner (2000) has shown how these models can be subsumed under the general notion of a *bilexical grammar* (BG), which means that parsing can be performed efficiently as discussed in Section 3.1. Eisner (2000) defines the notion of a *weighted bilexical grammar* (WBG) in terms of BG as follows (cf. Section 3.1):

1. A *weighted DFA* A is a deterministic finite automaton that associates a real-valued *weight* with each arc and each final state. Each accepting path through A is assigned a weight, namely the sum of all arc weights on the path and the weight of the final state. Each string x accepted by A is assigned the weight of its accepting path.

2. A WBG G is a BG in which all the automata l_w and r_w are weighted DFAs. The weight of a dependency tree y under G is defined as the sum, over all word tokens w in y, of the weight with which l_w accepts w's sequence of left children plus the weight with which r_w accepts w's sequence of right children.

Eisner (1996b) presents three different probabilistic models for dependency parsing, which can be reconstructed as different weighting schemes within the framework of WBG. However, the first two models (models A and B) require that we distinguish between an underlying string $x \in V^*$, described by the WBG, and a surface string x', which results from a possibly nondeterministic, possibly weighted finite-state transduction R on x. The surface string x' is then grammatical with analysis (y, p) if y is a grammatical dependency tree whose yield x is transduced to x' along an accepting path p in R. To avoid the distinction between underlying strings and surface strings, we will restrict our attention to model C, which was found to perform significantly better than the other two models in the experiments reported in Eisner (1996a).

First of all, it should be pointed out that all the models in Eisner (1996b) involve part-of-speech tags, in addition to word tokens and (unlabeled) dependency relations, and define the joint probability of the words, tags and dependency links. Model C is defined as follows:

$$P(tw(1), \ldots, tw(n), links) = \prod_{i=1}^{n} P(lc(i) \,|\, tw(i)) P(rc(i) \,|\, tw(i)) \tag{5}$$

where $tw(i)$ is the ith tagged word, and $lc(i)$ and $rc(i)$ are the left and right children of the ith word, respectively. The probability of generating each child is conditioned on the tagged head word and the tag of the preceding child (left children being generated from right to left):

$$P(lc(i) \,|\, tw(i)) = \prod_{j=1}^{m} P(tw(lc_j(i)) \,|\, t(lc_{j-1}(i)), tw(i)) \tag{6}$$

$$P(rc(i) \,|\, tw(i)) = \prod_{j=1}^{m} P(tw(rc_j(i)) \,|\, t(rc_{j-1}(i)), tw(i)) \tag{7}$$

where $lc_j(i)$ is the jth left child of the ith word and $t(lc_{j-1}(i))$ is the tag of the preceding left child (and analogously $rc_j(i)$ and $t(rc_{j-1}(i))$ for right children). This model can be implemented in the WBG framework by letting the automata l_w and r_w have weights on their arcs corresponding to the log of the probability of generating a particular left or right child given the tag of the preceding child. In this way, the weight assigned to a dependency tree T will be the log of $P(tw(1), \ldots, tw(n), links)$ as defined above (Eisner, 2000).

Eisner's work on data-driven dependency parsing has been influential in two ways. First, it showed that generative probabilistic modeling and supervised learning could be applied to dependency representations to achieve a parsing accuracy comparable to the best results reported for constituency-based parsing at the time,

although the evalutation metrics used in the two cases are not strictly comparable. Secondly, it showed how these models could be coupled with efficient parsing techniques that exploit the special properties of dependency structures. The importance of the second aspect can be seen in recent work by McDonald et al. (2005), applying discriminative estimation methods to probabilistic dependency parsing. Thanks to the more efficient parsing methods offered by Eisner's methods for bilexical parsing, training can be performed without pruning the search space, which is impossible for efficiency reasons when using lexicalized constituency representations with comparable lexical dependencies.

Collins et al. (1999) apply the generative probabilistic parsing models of Collins (1997, 1999) to dependency parsing, using data from the Prague Dependency Treebank. This requires preprocessing to transform dependency structures into flat phrase structures for the training phase and postprocessing to extract dependency structures from the phrase structures produced by the parser. The parser of Charniak (2000) has been adapted and applied to the Prague Dependency Treebank in a similar fashion, although this work remains unpublished.

Samuelsson (2000) proposes a probabilistic model for dependency grammar that goes beyond the models considered so far by incorporating labeled dependencies and allowing non-projective dependency structures. In this model, dependency representations are generated by two stochastic processes: one top-down process generating the tree structure y and one bottom-up process generating the surface string x given the tree structure, defining the joint probability as $P(x, y) = P(y)P(x|y)$. Samuelsson suggests that the model can be implemented using a bottom-up dynamic programming approach, but the model has unfortunately never been implemented and evaluated.

Another probabilistic approach to dependency parsing that incorporates labeled dependencies is the stochastic CDG parser of Wang and Harper (2004), which extends the CDG model with a generative probabilistic model. Parsing is performed in two steps, which may be tightly or loosely integrated, where the first step assigns to each word a set of SuperARVs, representing constraints on possible heads and dependents, and where the second step determines actual dependency links given the SuperARV assignment. Although the basic model and parsing algorithm is limited to projective structures, the system allows non-projective structures for certain *wh*-constructions. The system has been evaluated on data from the Wall Street Journal section of the Penn Treebank and achieves state-of-the-art performance using a dependency-based evaluation metric (Wang and Harper, 2004).

The first step in the parsing model of Wang and Harper (2004) can be seen as a kind of supertagging, which has turned out to be a crucial element in many recent approaches to statistical parsing, e.g. in LTAG (Joshi and Sarkar, 2003; Bangalore, 2003) and CCG (Clark and Curran, 2004; Curran and Clark, 2004). A similar

two-step process is used in the statistical dependency parser of Bangalore (2003), which uses elementary LTAG trees as supertags in order to derive a dependency structure in the second step. Supertagging is performed using a standard HMM trigram tagger, while dependency structures are derived using a heuristic deterministic algorithm that runs in linear time. Another data-driven dependency parser based on supertagging is Nasr and Rambow (2004), where supertags are derived from a lexicalized extended context-free grammar and the most probable analysis is derived using a modified version of the CKY algorithm.

Most of the systems described in this section are based on a formal dependency grammar in combination with a generative probabilistic model, which means that parsing conceptually consists in the derivation of all analyses that are permissible according to the grammar and the selection of the most probable analysis according to the generative model. This is in contrast to recent work based on purely discriminative models of inductive learning in combination with a deterministic parsing strategy, methods that do not involve a formal grammar.

The deterministic discriminative approach was first proposed by Kudo and Matsumoto (2000, 2002) and Yamada and Matsumoto (2003), using support vector machines (Vapnik, 1995) to train classifiers that predict the next action of a deterministic parser constructing unlabeled dependency structures. The parsing algorithm used in these systems implements a form of shift-reduce parsing with three possible parse actions that apply to two neighboring words referred to as *target nodes*:[10]

1. A *Shift* action adds no dependency construction between the target words w_i and w_{i+1} but simply moves the point of focus to the right, making w_{i+1} and w_{i+2} the new target words.

2. A *Right* action constructs a dependency relation between the target words, adding the left node w_i as a child of the right node w_{i+1} and reducing the target words to w_{i+1}, making w_{i-1} and w_{i+1} the new target words.

3. A *Left* action constructs a dependency relation between the target words, adding the right node w_{i+1} as a child of the left node w_i and reducing the target words to w_i, making w_{i-1} and w_i the new target words.

The parser processes the input from left to right repeatedly as long as new dependencies are added, which means that up to $n - 1$ passes over the input may be required to construct a complete dependency tree, giving a worst case time complexity of $O(n^2)$, although the worst case seldom occurs in practice. The features

[10]The parsers described in Kudo and Matsumoto (2000, 2002) are applied to Japanese, which is assumed to be strictly head-final, which means that only the actions *Shift* and *Right* are required.

used to predict the next parse action are the word forms and part-of-speech tags of the target words, of their left and right children, and of their left and right string context (in the reduced string). Yamada and Matsumoto (2003) evaluate the system using the standard data set from the Wall Street Journal section of the Penn Treebank and shows that deterministic discriminative dependency parsing can achieve an accuracy that is close to the state-of-the-art with respect to dependency accuracy. Further improvements with this model are reported in Isozaki et al. (2004).

The framework of inductive dependency parsing, first presented in Nivre et al. (2004) and more fully described in Nivre (2005), has many properties in common with the system of Yamada and Matsumoto (2003), but there are three differences. The first and most important difference is that the system of Nivre et al. (2004) constructs labeled dependency representations, i.e. representations where dependency arcs are labeled with dependency types. This also means that dependency type information can be exploited in the feature model used to predict the next parse action. The second difference is that the algorithm proposed in Nivre (2003) is a head-driven arc-eager algorithm that constructs a complete dependency tree in a single pass over the data. The third and final difference is that Nivre et al. (2004) use memory-based learning to induce classifiers for predicting the next parsing action based on conditional features, whereas Yamada and Matsumoto (2003) use support vector machines. However, as pointed out by Kudo and Matsumoto (2002), in a deterministic discriminative parser the learning method is largely independent of the rest of the system.

4 The Case for Dependency Parsing

As noted several times already, dependency-based syntactic representations have played a fairly marginal role in the history of linguistic theory as well as that of natural language processing. Saying that there is increasing interest in dependency-based approaches to syntactic parsing may therefore not be saying very much, but it is hard to deny that the notion of dependency has become more prominent in the literature on syntactic parsing during the last decade or so.

In conclusion, it therefore seems appropriate to ask what are the potential benefits of using dependency-based representations in syntactic parsing, as opposed to the more traditional representations based on constituency. According to Covington (2001), dependency parsing offers three *prima facie* advantages:

- Dependency links are close to the semantic relationships needed for the next stage of interpretation; it is not necessary to "read off" head-modifier or head-complement relations from a tree that does not show them directly.

- The dependency tree contains one node per word. Because the parser's job is only to connect existing nodes, not to postulate new ones, the task of parsing is in some sense more straightforward. [...]

- Dependency parsing lends itself to word-at-a-time operation, i.e., parsing by accepting and attaching words one at a time rather than by waiting for complete phrases. [...]

To this it is sometimes added that dependency-based parsing allows a more adequate treatment of languages with variable word order, where discontinuous syntactic constructions are more common than in languages like English (Mel'čuk, 1988; Covington, 1990b). However, this argument is only plausible if the formal framework allows non-projective dependency structures, which is not the case for most dependency parsers that exist today.

For us, the first two advantages identified by Covington seem to be the most important. Having a more constrained representation, where the number of nodes is fixed by the input string itself, should enable conceptually simpler and computationally more efficient methods for parsing. At the same time, it is clear that a more constrained representation is a less expressive representation and that dependency representations are necessarily underspecified with respect to certain aspects of syntactic structure. For example, as pointed out by Mel'čuk (1988), it is impossible to distinguish in a pure dependency representation between an element modifying the head of a phrase and the same element modifying the entire phrase. However, this is precisely the kind of ambiguity that is notoriously hard to disambiguate correctly in syntactic parsing anyway, so it might be argued that this kind of underspecification is actually beneficial. And as long as the syntactic representation encodes enough of the structural relations that are relevant for semantic interpretation, then we are only happy if we can constrain the problem of deriving these representations.

In general, there is a tradeoff between the expressivity of syntactic representations and the complexity of syntactic parsing, and we believe that dependency representations provide a good compromise in this respect. They are less expressive than most constituency-based representations, but they compensate for this by providing a relatively direct encoding of predicate-argument structure, which is relevant for semantic interpretation, and by being composed of bilexical relations, which are beneficial for disambiguation. In this way, dependency structures are sufficiently expressive to be useful in natural language processing systems and at the same time sufficiently restricted to allow full parsing with high accuracy and efficiency. At least, this seems like a reasonable working hypothesis.

References

Abney, S. (1996). Partial parsing via finite-state cascades. *Journal of Natural Language Engineering* **2**: 337–344.

Alshawi, H. (1996). Head automata and bilingual tiling: Translation with minimal representations. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 167–176.

Bangalore, S. (2003). Localizing dependencies and supertagging. In Bod, R., Scha, R. and Sima'an, K. (eds), *Data-Oriented Parsing*, CSLI Publications, University of Chicago Press, pp. 283–298.

Bar-Hillel, Y., Gaifman, C. and Shamir, E. (1960). On categorial and phrase-structure grammars. *Bulletin of the Research Council of Israel* **9F**: 1–16.

Barbero, C., Lesmo, L., Lombardo, V. and Merlo, P. (1998). Integration of syntactic and lexical information in a hierarchical dependency grammar. In Kahane, S. and Polguère, A. (eds), *Proceedings of the Workshop on Processing of Dependency-Based Grammars (ACL-COLING)*, pp. 58–67.

Bloomfield, L. (1933). *Language*. The University of Chicago Press.

Carroll, G. and Charniak, E. (1992). Two experiments on learning probabilistic dependency grammars from corpora, *Technical Report TR-92*, Department of Computer Science, Brown University.

Carroll, J. (2000). Statistical parsing. In Dale, R., Moisl, H. and Somers, H. (eds), *Handbook of Natural Language Processing*, Marcel Dekker, pp. 525–543.

Charniak, E. (2000). A maximum-entropy-inspired parser. *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 132–139.

Chomsky, N. (1970). Remarks on nominalization. In Jacobs, R. and Rosenbaum, P. S. (eds), *Readings in English Transformational Grammar*, Ginn and Co.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 104–111.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 16–23.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Collins, M., Hajič, J., Brill, E., Ramshaw, L. and Tillmann, C. (1999). A statistical parser for Czech. *Proceedings of the 37th Meeting of the Association for Computational Linguistics (ACL)*, pp. 505–512.

Covington, M. A. (1984). *Syntactic Theory in the High Middle Ages*. Cambridge University Press.

Covington, M. A. (1990a). A dependency parser for variable-word-order languages, *Technical Report AI-1990-01*, University of Georgia.

Covington, M. A. (1990b). Parsing discontinuous constituents in dependency grammar. *Computational Linguistics* **16**: 234–236.

Covington, M. A. (1994). Discontinuous dependency parsing of free and fixed word order: Work in progress, *Technical Report AI-1994-02*, University of Georgia.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. *Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102.

Curran, J. R. and Clark, S. (2004). The importance of supertagging for wide-coverage CCG parsing. *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pp. 282–288.

Debusmann, R. (2001). *A declarative grammar formalism for dependency grammar*, Master's thesis, Computational Linguistics, Universität des Saarlandes.

Debusmann, R., Duchier, D. and Kruijff, G.-J. M. (2004). Extensible dependency grammar: A new methodology. *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pp. 78–85.

Dowty, D. (1989). On the semantic content of the notion of 'thematic role'. In Chierchia, G., Partee, B. H. and Turner, R. (eds), *Properties, Types and Meaning. Volume II: Semantic Issues*, Reider, pp. 69–130.

Duchier, D. (1999). Axiomatizing dependency parsing using set constraints. *Proceedings of the Sixth Meeting on Mathematics of Language*, pp. 115–126.

Duchier, D. (2003). Configuration of labeled trees under lexicalized constraints and principles. *Research on Language and Computation* **1**: 307–336.

Duchier, D. and Debusmann, R. (2001). Topological dependency trees: A constraint-based account of linear precedence. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 180–187.

Earley, J. (1970). En efficient context-free parsing algorithm. *Communications of the ACM* **13**: 94–102.

Eisner, J. M. (1996a). An empirical comparison of probability models for dependency grammar, *Technical Report IRCS-96-11*, Institute for Research in Cognitive Science, University of Pennsylvania.

Eisner, J. M. (1996b). Three new probabilistic models for dependency parsing: An exploration. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 340–345.

Eisner, J. M. (2000). Bilexical grammars and their cubic-time parsing algorithms. In Bunt, H. and Nijholt, A. (eds), *Advances in Probabilistic and Other Parsing Technologies*, Kluwer, pp. 29–62.

Fillmore, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (eds), *Universals in Linguistic Theory*, Holt, Rinehart and Winston, pp. 1–88.

Foth, K., Daum, M. and Menzel, W. (2004). A broad-coverage parser for German based on defeasible constraints. *Proceedings of KONVENS 2004*, pp. 45–52.

Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and Control* **8**: 304–337.

Grimaldi, R. P. (2004). *Discrete and Combinatorial Mathematics*. 5th edn, Addison-Wesley.

Harper, M. P. and Helzerman, R. A. (1995). Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language* **9**: 187–234.

Harper, M. P., Helzermann, R. A., Zoltowski, C. B., Yeo, B. L., Chan, Y., Steward, T. and Pellom, B. L. (1995). Implementation issues in the development of the PARSEC parser. *Software: Practice and Experience* **25**: 831–862.

Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language* **40**: 511–525.

Hellwig, P. (1980). PLAIN – a program system for dependency analysis and for simulating natural language inference. In Bolc, L. (ed.), *Representation and Processing of Natural Language*, Hanser, pp. 195–198.

Hellwig, P. (1986). Dependency unification grammar. *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, pp. 195–198.

Hellwig, P. (2003). Dependency unification grammar. In Agel, V., Eichinger, L. M., Eroms, H.-W., Hellwig, P., Heringer, H. J. and Lobin, H. (eds), *Dependency and Valency*, Walter de Gruyter, pp. 593–635.

Holan, T., Kuboň, V. and Plátek, M. (1997). A prototype of a grammar checker for Czech. *Fifth Conference on Applied Natural Language Processing*, pp. 147–154.

Hudson, R. A. (1984). *Word Grammar*. Blackwell.

Hudson, R. A. (1990). *English Word Grammar*. Blackwell.

Isozaki, H., Kazawa, H. and Hirao, T. (2004). A deterministic word dependency analyzer enhanced with preference learning. *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pp. 275–281.

Jackendoff, R. (1972). *Semantic Interpretation in Generative Grammar*. MIT Press.

Jackendoff, R. S. (1977). $\overline{X}$ *Syntax: A Study of Phrase Structure*. MIT Press.

Järvinen, T. and Tapanainen, P. (1998). Towards an implementable dependency grammar. In Kahane, S. and Polguère, A. (eds), *Proceedings of the Workshop on Processing of Dependency-Based Grammars*, pp. 1–10.

Joshi, A. and Sarkar, A. (2003). Tree adjoining grammars and their application to statistical parsing. In Bod, R., Scha, R. and Sima'an, K. (eds), *Data-Oriented Parsing*, CSLI Publications, University of Chicago Press, pp. 253–281.

Kahane, S., Nasr, A. and Rambow, O. (1998). Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pp. 646–652.

Karlsson, F. (1990). Constraint grammar as a framework for parsing running text. In Karlgren, H. (ed.), *Papers presented to the 13th International Conference on Computational Linguistics (COLING)*, pp. 168–173.

Karlsson, F., Voutilainen, A., Heikkilä, J. and Anttila, A. (eds) (1995). *Constraint Grammar: A language-independent system for parsing unrestricted text*. Mouton de Gruyter.

Kasami, T. (1965). An efficient recognition and syntax algorithm for context-free languages, *Technical Report AF-CRL-65-758*, Air Force Cambridge Research Laboratory.

Kromann, M. T. (2004). Optimality parsing and local cost functions in Discontinuous Grammar. *Electronic Notes of Theoretical Computer Science* **53**: 163–179.

Kruijff, G.-J. M. (2001). *A Categorial-Modal Logical Architecture of Informativity: Dependency Grammar Logic and Information Structure*. PhD thesis, Charles University.

Kruijff, G.-J. M. (2002). Formal and computational aspects of dependency grammar: History and development of DG, *Technical report*, ESSLLI-2002.

Kudo, T. and Matsumoto, Y. (2000). Japanese dependency structure analysis based on support vector machines. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC)*, pp. 18–25.

Kudo, T. and Matsumoto, Y. (2002). Japanese dependency analysis using cascaded chunking. *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pp. 63–69.

Lecerf, Y. (1960). Programme des conflits, modèle des conflits. *Bulletin bimestriel de l'ATALA* **1(4): 11–18, 1(5): 17–36**.

Lin, D. (1996). On the structural complexity of natural language sentences. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 729–733.

Lombardo, V. and Lesmo, L. (1996). An Earley-type recognizer for Dependency Grammar. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 723–728.

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

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. and Schasberger, B. (1994). The Penn Treebank: Annotating predicate-argument structure. *Proceedings of the ARPA Human Language Technology Workshop*, pp. 114–119.

Marcus, S. (1965). Sur la notion de projectivité. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **11**: 181–192.

Maruyama, H. (1990). Structural disambiguation with constraint propagation. *Proceedings of the 28th Meeting of the Association for Computational Linguistics (ACL)*, Pittsburgh, PA, pp. 31–38.

McDonald, R., Crammer, K. and Pereira, F. (2005). Online large-margin training of dependency parsers. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 91–98.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Menzel, W. and Schröder, I. (1998). Decision procedures for dependency parsing using graded constraints. In Kahane, S. and Polguère, A. (eds), *Proceedings of the Workshop on Processing of Dependency-Based Grammars*, pp. 78–87.

Milward, D. (1994). Dynamic dependency grammar. *Linguistics and Philosophy* **17**: 561–605.

Misra, V. N. (1966). *The Descriptive Technique of Panini*. Mouton.

Nasr, A. and Rambow, O. (2004). A simple string-rewriting formalism for dependency grammar. *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pp. 25–32.

Nikula, H. (1986). *Dependensgrammatik*. Liber.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 149–160.

Nivre, J. (2005). *Inductive Dependency Parsing of Natural Language Text*. PhD thesis, Växjö University.

Nivre, J., Hall, J. and Nilsson, J. (2004). Memory-based dependency parsing. In Ng, H. T. and Riloff, E. (eds), *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pp. 49–56.

Obrebski, T. (2003). Dependency parsing using dependency graph. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 217–218.

Oflazer, K. (2003). Dependency parsing with an extended finite-state approach. *Computational Linguistics* **29**: 515–544.

Robins, R. H. (1967). *A Short History of Linguistics*. Longman.

Robinson, J. J. (1970). Dependency structures and transformational rules. *Language* **46**: 259–285.

Roche, E. (1997). Parsing with finite state transducers. In Roche, E. and Schabes, Y. (eds), *Finite-State Language Processing*, MIT Press, pp. 241–281.

Samuelsson, C. (2000). A statistical theory of dependency syntax. *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*.

Schröder, I. (2002). *Natural Language Parsing with Graded Constraints*. PhD thesis, Hamburg University.

Sgall, P., Hajičová, E. and Panevová, J. (1986). *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

Sleator, D. and Temperley, D. (1991). Parsing English with a link grammar, *Technical Report CMU-CS-91-196*, Carnegie Mellon University, Computer Science.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. *Third International Workshop on Parsing Technologies (IWPT)*, pp. 277–292.

Starosta, S. (1988). *The Case for Lexicase: An Outline of Lexicase Grammatical Theory*. Pinter Publishers.

Tapanainen, P. and Järvinen, T. (1997). A non-projective dependency parser. *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp. 64–71.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Editions Klincksieck.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer.

Wang, W. and Harper, M. P. (2004). A statistical constraint dependency grammar (CDG) parser. In Keller, F., Clark, S., Crocker, M. and Steedman, M. (eds), *Proceedings of the Workshop in Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pp. 42–29.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 195–206.

Yli-Jyrä, A. (2003). Multiplanarity – a model for dependency structures in treebanks. In Nivre, J. and Hinrichs, E. (eds), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press, pp. 189–200.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control* **10**: 189–208.

Zwicky, A. M. (1985). Heads. *Journal of Linguistics* **21**: 1–29.

Natural Language Engineering **13** (2): 95–135.
doi:10.1017/S1351324906004505 First published online 12 January 2007 Printed in the United Kingdom

MaltParser: A language-independent system for data-driven dependency parsing

JOAKIM NIVRE

Växjö University, School of Mathematics and Systems Engineering, 35195 Växjö, Sweden
Uppsala University, Department of Linguistics and Philology, Box 635, 75126 Uppsala, Sweden
e-mail: joakim.nivre@msi.vxu.se

JOHAN HALL, JENS NILSSON

Växjö University, School of Mathematics and Systems Engineering, 35195 Växjö, Sweden
e-mail: {johan.hall,jens.nilsson}@msi.vxu.se

ATANAS CHANEV

University of Trento, Dept. of Cognitive Sciences, 38068 Rovereto, Italy
ITC-irst, 38055 Povo-Trento, Italy
e-mail: chanev@form.unitn.it

GÜLŞEN ERYİĞİT

Istanbul Technical University, Dept. of Computer Engineering, 34469 Istanbul, Turkey
e-mail: gulsen.cebiroglu@itu.edu.tr

SANDRA KÜBLER

University of Tübingen, Seminar für Sprachwissenschaft, Wilhelmstr. 19, 72074 Tübingen, Germany
e-mail: kuebler@sfs.uni-tuebingen.de

SVETOSLAV MARINOV

University of Skövde, School of Humanities and Informatics, Box 408, 54128 Skövde, Sweden
Göteborg University & GSLT, Faculty of Arts, Box 200, 40530 Göteborg, Sweden
e-mail: svetoslav.marinov@his.se

ERWIN MARSI

Tilburg University, Communication and Cognition, Box 90153, 5000 LE Tilburg, The Netherlands
e-mail: e.c.marsi@uvt.nl

(*Received 16 February 2006; revised 15 August 2006*)

Abstract

Parsing unrestricted text is useful for many language technology applications but requires parsing methods that are both robust and efficient. MaltParser is a language-independent system for data-driven dependency parsing that can be used to induce a parser for a new language from a treebank sample in a simple yet flexible manner. Experimental evaluation confirms that MaltParser can achieve robust, efficient and accurate parsing for a wide range of languages without language-specific enhancements and with rather limited amounts of training data.

1 Introduction

One of the potential advantages of data-driven approaches to natural language processing is that they can be ported to new languages, provided that the necessary

linguistic data resources are available. In practice, this advantage can be hard to realize if models are overfitted to a particular language or linguistic annotation scheme. Thus, several studies have reported a substantial increase in error rate when applying state-of-the-art statistical parsers developed for English to other languages, such as Czech (Collins *et al.* 1999), Chinese (Bikel and Chiang 2000; Levy and Manning 2003), German (Dubey and Keller 2003), and Italian (Corazza *et al.* 2004). Another potential obstacle to successful reuse is that data-driven models may require large amounts of annotated training data to give good performance, while for most languages the availability of such resources is relatively limited. This is also a problem when porting parsers to new domains, even for languages where large amounts of annotated data are available (Titov and Henderson 2006). Given that approaches based on completely unsupervised learning are still vastly inferior in terms of accuracy, there is consequently a need for supervised approaches that are resilient against data sparseness.

In this article, we present a data-driven approach to dependency parsing that has been applied to a range of different languages, consistently giving a dependency accuracy in the range 80–90%, usually with less than a 5% increase in error rate compared to state-of-the-art parsers for the language in question. All these results have been obtained without any language-specific enhancements and in most cases with fairly modest data resources.

The methodology is based on three essential techniques:

1. Deterministic parsing algorithms for building dependency graphs (Kudo and Matsumoto 2002; Yamada and Matsumoto 2003; Nivre 2003)
2. History-based feature models for predicting the next parser action (Black *et al.* 1992; Magerman 1995; Ratnaparkhi 1997; Collins 1999)
3. Discriminative machine learning to map histories to parser actions (Veenstra and Daelemans 2000; Kudo and Matsumoto 2002; Yamada and Matsumoto 2003; Nivre *et al.* 2004)

The system uses no grammar but relies completely on inductive learning from treebank data for the analysis of new sentences and on deterministic parsing for disambiguation. This combination of methods guarantees that the parser is both robust, producing a well-formed analysis for every input sentence, and efficient, deriving this analysis in time that is linear or quadratic in the length of the sentence (depending on the particular algorithm used).

This methodology has been implemented in the MaltParser system, which can be applied to a labeled dependency treebank in order to induce a labeled dependency parser for the language represented by the treebank. MaltParser is freely available for research and educational purposes[1] and has been designed primarily as a tool for research on data-driven dependency parsing, allowing users to flexibly combine different parsing algorithms, feature models, and learning algorithms. However, given that the necessary data resources are available, MaltParser can also be used

[1] URL: `http://www.msi.vxu.se/users/nivre/research/MaltParser.html`.

for rapid development of robust and efficient dependency parsers, which can be used in language technology applications that require parsing of unrestricted text.

In this article, we begin by describing the general methodology of deterministic dependency parsing with history-based feature models and discriminative machine learning (section 2). We then describe the implemented MaltParser system, focusing on its functionality with respect to parsing algorithms, feature models, and learning algorithms (section 3). To support our claims about language-independence and resilience against data sparseness, we then present an experimental evaluation based on data from ten different languages, with treebanks of different sizes and with different annotation schemes (section 4). Finally, we draw some general conclusions and make some suggestions for future work (section 5).

2 Inductive dependency parsing

Mainstream approaches in statistical parsing are based on nondeterministic parsing techniques, usually employing some kind of dynamic programming, in combination with generative probabilistic models that provide an *n*-best ranking of the set of candidate analyses derived by the parser. This methodology is exemplified by the influential parsers of Collins (1997; 1999) and Charniak (2000), among others. The accuracy of these parsers can be further improved by reranking the analyses output by the parser, typically using a discriminative model with global features that are beyond the scope of the underlying generative model (Johnson *et al.* 1999; Collins 2000; Collins and Duffy 2002; Collins and Koo 2005; Charniak and Johnson 2005).

A radically different approach is to perform disambiguation deterministically, using a greedy parsing algorithm that approximates a globally optimal solution by making a series of locally optimal choices, guided by a classifier trained on gold standard derivation sequences derived from a treebank. Although this may seem like a futile strategy for a complex task like parsing, it has recently been used with some success especially in dependency-based parsing.[2] It was first applied to unlabeled dependency parsing by Kudo and Matsumoto (2002) (for Japanese) and by Yamada and Matsumoto (2003) (for English). It was later extended to labeled dependency parsing by Nivre *et al.* (2004) (for Swedish) and Nivre and Scholz (2004) (for English). More recently, it has also been applied with good results to lexicalized phrase structure parsing by Sagae and Lavie (2005).

One of the advantages of the deterministic, classifier-based approach is that it is straightforward to implement and has a very attractive time complexity, with parsing time being linear or at worst quadratic in the size of the input, although the constant associated with the classifier can sometimes become quite large. Moreover, while the accuracy of a deterministic parser is normally a bit lower than what can be attained with a more complex statistical model, trained and tuned on large amounts of data, the deterministic parser will often have a much steeper learning curve,

[2] In fact, essentially the same methodology has been proposed earlier for other frameworks by Berwick (1985), Simmons and Yu (1992), Zelle and Mooney (1993) and Veenstra and Daelemans (2000), among others, although these approaches have typically been evaluated only on artificially generated or very small data sets.

which means that it may in fact give higher accuracy with small training data sets. This is a natural consequence of the fact that the deterministic model has a much smaller parameter space, where only the mode of the distribution for each distinct history needs to be estimated, whereas a traditional generative model requires a complete probability distribution. Finally, and for essentially the same reason, the deterministic model can be less sensitive to differences in linguistic structure and annotation style across languages and should therefore be more easily portable without substantial adaptation.

In this study, we investigate these issues by applying the deterministic, classifier-based approach, as implemented in the MaltParser system for inductive dependency parsing, to a wide range of languages with varying annotation schemes and with data sets of varying sizes. By way of background, this section presents the theoretical foundations of inductive dependency parsing, defining syntactic representations, parsing algorithms, feature models, and learning algorithms.[3] In section 3, we then describe the implemented MaltParser system that has been used for the experiments reported in section 4.

2.1 Dependency graphs

In dependency parsing, the syntactic analysis of a sentence is represented by a dependency graph, which we define as a labeled directed graph, the nodes of which are indices corresponding to the tokens of a sentence. Formally:

Definition 1
Given a set R of dependency types (arc labels), a *dependency graph* for a sentence $x = (w_1, \ldots, w_n)$ is a labeled directed graph $G = (V, E, L)$, where:

1. $V = \mathbf{Z}_{n+1}$
2. $E \subseteq V \times V$
3. $L : E \rightarrow R$

Definition 2
A dependency graph G is *well-formed* if and only if:

1. The node 0 is a root (ROOT).
2. G is connected (CONNECTEDNESS).[4]

The set V of *nodes* (or *vertices*) is the set $\mathbf{Z}_{n+1} = \{0, 1, 2, \ldots, n\}$ ($n \in \mathbf{Z}^+$), i.e., the set of non-negative integers up to and including n. This means that every token index i of the sentence is a node ($1 \leq i \leq n$) and that there is a special node 0, which does not correspond to any token of the sentence and which will always be a root of the dependency graph (normally the only root).

[3] For an in-depth discussion of inductive dependency parsing and its relation to other parsing methods, see Nivre (2006).

[4] Strictly speaking, we require the graph to be *weakly connected*, which entails that the corresponding undirected graph is connected, whereas a *strongly connected* graph has a *directed* path between any pair of nodes.

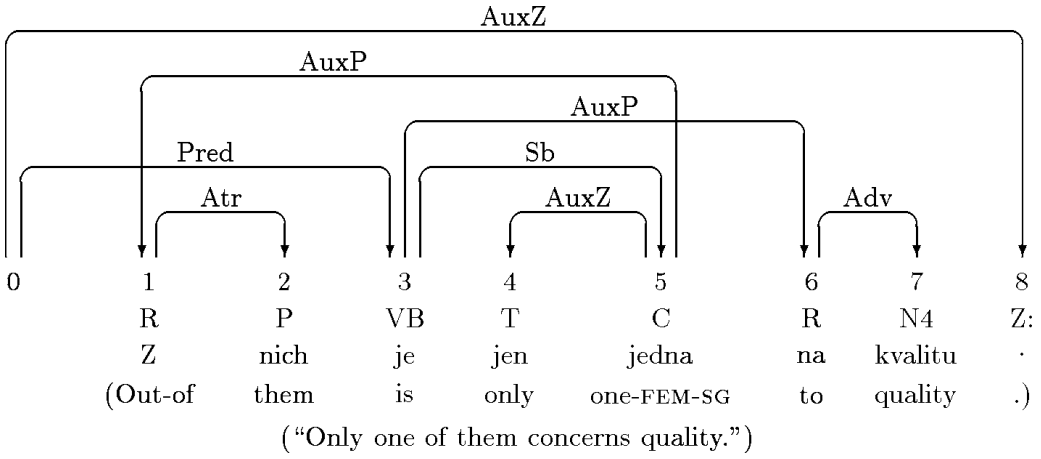


Fig. 1. Dependency graph for Czech sentence from the Prague Dependency Treebank.

In the following, we will reserve the term *token node* for a node that corresponds to a token of the sentence, and we will use the symbol V^+ to denote the set of token nodes of a sentence for which the set of nodes is V, i.e., $V^+ = V - \{0\}$. When necessary, we will write V_x and V_x^+ to indicate that V and V^+ are the nodes corresponding to a particular sentence $x = (w_1, \ldots, w_n)$. Note, however, that the only requirement imposed by x is that the number of nodes matches the length of x, i.e., $|V^+| = n$ and $|V| = n + 1$.

The set E of *arcs* (or *edges*) is a set of ordered pairs (i, j), where i and j are nodes. Since arcs are used to represent dependency relations, we will say that i is the *head* and j is the *dependent* of the arc (i, j). As usual, we will use the notation $i \rightarrow j$ to mean that there is an arc connecting i and j (i.e., $(i, j) \in E$) and we will use the notation $i \rightarrow^* j$ for the reflexive and transitive closure of the arc relation E (i.e., $i \rightarrow^* j$ if and only if $i = j$ or there is a path of arcs connecting i to j).

The function L assigns a dependency type (arc label) $r \in R$ to every arc $e \in E$. We will use the notation $i \stackrel{r}{\rightarrow} j$ to mean that there is an arc labeled r connecting i to j (i.e., $(i, j) \in E$ and $L((i, j)) = r$).

Figure 1 shows a Czech sentence from the Prague Dependency Treebank with a well-formed dependency graph according to Definition 1–2. Note that the use of a special root node (0) is crucial for the satisfaction of CONNECTEDNESS, since the graph would otherwise have consisted of two connected components rooted at nodes 3 and 8, respectively. The use of a special root node is thus a convenient way of ensuring CONNECTEDNESS, regardless of whether a particular annotation scheme requires that a single token node should dominate all the others. More importantly, it is a way of achieving robustness in parsing, since there will always be a single entry point into the graph even if the parser produces fragmented output.

The only conditions so far imposed on dependency graphs is that the special node 0 be a root and that the graph be connected. Here are three further constraints that are common in the literature:

3. Every node has at most one head, i.e., if $i \rightarrow j$ then there is no node k such that $k \neq i$ and $k \rightarrow j$ (SINGLE-HEAD).

4. The graph G is acyclic, i.e., if $i \rightarrow j$ then not $j \rightarrow^* i$ (ACYCLICITY).
5. The graph G is projective, i.e., if $i \rightarrow j$ then $i \rightarrow^* k$, for every node k such that $i < k < j$ or $j < k < i$ (PROJECTIVITY).

The SINGLE-HEAD constraint, together with the basic well-formedness conditions, entails that the graph is a tree rooted at the node 0, which means that any well-formed graph satisfying SINGLE-HEAD also satisfies ACYCLICITY. And whereas it is possible to require ACYCLICITY without SINGLE-HEAD, the two conditions are jointly assumed in almost all versions of dependency grammar, especially in computational systems.

By contrast, PROJECTIVITY is much more controversial. Broadly speaking, we can say that whereas most practical systems for dependency parsing do assume projectivity, most dependency-based linguistic theories do not. More precisely, most theoretical formulations of dependency grammar regard projectivity as the norm but also recognize the need for non-projective representations to capture non-local dependencies and discontinuities arising from free or flexible word order (Mel'čuk 1988; Hudson 1990). This theoretical preference for non-projective dependency graphs is usually carried over into treebank annotation schemes, so that virtually all treebanks annotated with dependency graphs contain non-projective structures. This is true, for example, of the Prague Dependency Treebank of Czech (Hajič *et al.* 2001), the Danish Dependency Treebank (Kromann 2003), and the Turkish Treebank (Oflazer *et al.* 2003), all of which are used in this study.

2.2 Deterministic parsing algorithms

The most commonly used deterministic algorithms for dependency parsing can be seen as variants of the basic shift-reduce algorithm, analyzing the input from left to right using two main data structures, a queue of remaining input tokens and a stack storing partially processed tokens. One example is the arc-eager algorithm introduced in Nivre (2003), which is used in all the experiments in this article and which we describe in detail in this section. Like most of the algorithms used for practical dependency parsing, this algorithm is restricted to projective dependency graphs. We begin by defining a parser *configuration* for a sentence $x = (w_1, \ldots, w_n)$, relative to a set R of dependency types (including a special symbol r_0 for dependents of the root):

Definition 3
Given a set $R = \{r_0, r_1, \ldots r_m\}$ of dependency types and a sentence $x = (w_1, \ldots, w_n)$, a *parser configuration* for x is a quadruple $c = (\sigma, \tau, h, d)$, where:

1. σ is a stack of token nodes i ($1 \leq i \leq j$ for some $j \leq n$).
2. τ is a sorted sequence of token nodes i ($j < i \leq n$).
3. $h : V_x^+ \rightarrow V_x$ is a function from token nodes to nodes.
4. $d : V_x^+ \rightarrow R$ is a function from token nodes to dependency types.
5. For every token node $i \in V_x^+$, $d(i) = r_0$ only if $h(i) = 0$.

The idea is that the sequence τ represents the remaining input tokens in a left-to-right pass over the input sentence x; the stack σ contains partially processed nodes

that are still candidates for dependency arcs, either as heads or dependents; and the functions h and d represent the (partially constructed) dependency graph for the input sentence x.

Representing the graph by means of two functions in this way is possible if we assume the SINGLE-HEAD constraint. Since, for every token node j, there is at most one arc (i, j), we can represent this arc by letting $h(j) = i$. Strictly speaking, h should be a partial function, to allow the possibility that there is no arc (i, j) for a given node j, but we will avoid this complication by assuming that every node j for which there is no token node i such that $i \rightarrow j$ is headed by the special root node 0, i.e., $h(j) = 0$. Formally, we establish the connection between configurations and dependency graphs as follows:

Definition 4
A configuration $c = (\sigma, \tau, h, d)$ for $x = (w_1, \ldots, w_n)$ defines the dependency graph $G_c = (V_x, E_c, L_c)$, where:

1. $E_c = \{(i, j) \mid h(j) = i\}$
2. $L_c = \{((i, j), r) \mid h(j) = i, d(j) = r\}$

We use the following notational conventions for the components of a configuration:

1. Both the stack σ and the sequence of input tokens τ will be represented as lists, although the stack σ will have its head (or top) to the right for reasons of perspicuity. Thus, $\sigma|i$ represents a stack with top i and tail σ, while $j|\tau$ represents a list of input tokens with head j and tail τ, and the operator $|$ is taken to be left-associative for the stack and right-associative for the input list. We use ϵ to represent the empty list/stack.
2. For the functions h and d, we will use the notation $f[x \mapsto y]$, given a specific function f, to denote the function g such that $g(x) = y$ and $g(z) = f(z)$ for all $z \neq x$. More formally, if $f(x) = y'$, then $f[x \mapsto y] = (f - \{(x, y')\}) \cup \{(x, y)\}$.

Initial and terminal parser configurations are defined in the following way:

Definition 5
A configuration c for $x = (w_1, \ldots, w_n)$ is *initial* if and only if it has the form $c = (\epsilon, (1, \ldots, n), h_0, d_0)$, where:

1. $h_0(i) = 0$ for every $i \in V_x^+$.
2. $d_0(i) = r_0$ for every $i \in V_x^+$.

A configuration c for $x = (w_1, \ldots, w_n)$ is *terminal* if and only if it has the form $c = (\sigma, \epsilon, h, d)$ (for arbitrary σ, h and d).

In other words, we initialize the parser with an empty stack, with all the token nodes of the sentence remaining to be processed, and with a dependency graph where all token nodes are dependents of the special root node 0 and all arcs are labeled with the special label r_0, and we terminate whenever the list of input tokens is empty, which happens when we have completed one left-to-right pass over the sentence. We use C for the set of all possible configurations (relative to some set

R of dependency types) and C^n for the set of non-terminal configurations, i.e., any configuration $c = (\sigma, \tau, h, d)$ where $\tau \neq \epsilon$.

A *transition* is a partial function $t : C^n \rightarrow C$. In other words, a transition maps non-terminal configurations to new configurations but may be undefined for some non-terminal configurations. The parsing algorithm uses four transitions, two of which are parameterized by a dependency type $r \in R$.

Definition 6
Given a set of dependency types R, the following transitions are possible for every $r \in R$:

1. LEFT-ARC(r):
   $(\sigma|i, j|\tau, h, d) \rightarrow (\sigma, j|\tau, h[i \mapsto j], d[i \mapsto r])$
   if $h(i) = 0$

2. RIGHT-ARC(r):
   $(\sigma|i, j|\tau, h, d) \rightarrow (\sigma|i|j, \tau, h[j \mapsto i], d[j \mapsto r])$
   if $h(j) = 0$

3. REDUCE:
   $(\sigma|i, \tau, h, d) \rightarrow (\sigma, \tau, h, d)$
   if $h(i) \neq 0$

4. SHIFT:
   $(\sigma, i|\tau, h, d) \rightarrow (\sigma|i, \tau, h, d)$

The transition LEFT-ARC(r) makes the top token i a (left) dependent of the next token j with dependency type r, i.e., $j \overset{r}{\rightarrow} i$, and immediately pops the stack. This transition can apply only if $h(i) = 0$, i.e., if the top token is previously attached to the root 0. The node i is popped from the stack because it must be complete with respect to left and right dependents at this point (given the assumption of projectivity).

The transition RIGHT-ARC(r) makes the next token j a (right) dependent of the top token i with dependency type r, i.e., $i \overset{r}{\rightarrow} j$, and immediately pushes j onto the stack. This transition can apply only if $h(j) = 0$, i.e., if the next token is previously attached to the root 0.[5] The node j is pushed onto the stack since it must be complete with respect to its left dependents at this point, but it cannot be popped because it may still need new dependents to the right.

The transition REDUCE pops the stack. This transition can apply only if $h(i) \neq 0$, i.e., if the top token i is already attached to a token node. This transition is needed for popping a node that was pushed in a RIGHT-ARC(r) transition and which has since found all its right dependents.

The transition SHIFT pushes the next token i onto the stack. This transition can apply unconditionally as long as there are input tokens remaining. It is needed for

[5] This condition is in fact superfluous, since it is impossible for the next input token to be attached to any other node, but it is included for symmetry.

processing nodes that have their heads to the right, as well as nodes that are to remain attached to the special root node.

The transition system just defined is nondeterministic in itself, since there is normally more than one transition applicable to a given configuration. In order to perform deterministic parsing, the transition system needs to be supplemented with a mechanism for predicting the next transition at each nondeterministic choice point, as well as choosing a dependency type r for the transitions LEFT-ARC(r) and RIGHT-ARC(r). Such a mechanism can be called an *oracle* (Kay 2000). Assuming that we have an oracle $o : C^n \rightarrow (C^n \rightarrow C)$, the algorithm for deterministic dependency parsing is very simple and straightforward:

```
PARSE(x = (w_1, ..., w_n))
1  c ← (ε, (1, ..., n), h_0, d_0)
2  while c = (σ, τ, h, d) is not terminal
3      if σ = ε
4          c ← SHIFT(c)
5      else
6          c ← [o(c)](c)
7  G ← (V_x, E_c, L_c)
8  return G
```

As long as the parser remains in a non-terminal configuration, it applies the SHIFT transition if the stack is empty and otherwise the transition $o(c)$ predicted by the oracle. When a terminal configuration is reached, the dependency graph defined by this configuration is returned.

The notion of an oracle is useful for the theoretical analysis of parsing algorithms and allows us to show, for example, that the parsing algorithm just described derives a well-formed projective dependency graph for any input sentence in time that is linear in the length of the input, and that any projective dependency graph can be derived by the algorithm (Nivre 2006). In practice, the oracle can only be approximated, but the fundamental idea in inductive dependency parsing is that we can achieve a good approximation using history-based feature models and discriminative machine learning, as described in the following subsections.

An alternative to the algorithm described in this section is to use an arc-standard strategy, more directly corresponding to the strict bottom-up processing in traditional shift-reduce parsing. In this scheme, the RIGHT-ARC(r) and REDUCE transitions are merged into a single transition that immediately pops the dependent in the same way as LEFT-ARC(r), which means that right dependents can only be attached after they have found all their descendants. This is the strategy used by Kudo and Matsumoto (2002), Yamada and Matsumoto (2003) and Cheng *et al.* (2004), although they also modify the algorithm by allowing multiple passes over the input. There are few studies comparing the performance of different algorithms, but Cheng *et al.* (2005) found consistently better accuracy for the arc-eager, single-pass strategy (over the arc-standard, multi-pass algorithm) in parsing the CKIP Treebank of Chinese.

A somewhat different approach is to use the incremental algorithms described by Covington (2001), where the stack is replaced by an open list where any token can be linked to the next input token. This allows non-projective graphs to be

derived at the cost of making parsing time quadratic in the length of the input. This is a technique that has not yet been evaluated on a large scale, and attempts at recovering non-projective dependencies within this tradition have so far relied on post-processing of projective dependency graphs, e.g., using the pseudo-projective technique proposed by Nivre and Nilsson (2005).

2.3 History-based feature models

History-based models for natural language processing were first introduced by Black *et al.* (1992) and have been used extensively for part-of-speech tagging and syntactic parsing. The basic idea is to map each pair (x, y) of an input string x and an analysis y to a sequence of decisions $D = (d_1, \ldots, d_n)$. In a generative probabilistic model, the joint probability $P(x, y)$ can then be expressed using the chain rule of probabilities as follows:

$$P(x, y) = P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid d_1, \ldots, d_{i-1}) \tag{1}$$

The conditioning context for each d_i, $(d_1, \ldots, d_{i-1})$, is referred to as the *history* and usually corresponds to some partially built structure. In order to get a tractable learning problem, histories are grouped into equivalence classes by a function Φ:

$$P(x, y) = P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid \Phi(d_1, \ldots, d_{i-1})) \tag{2}$$

Early versions of this scheme were integrated into grammar-driven systems. For example, Black *et al.* (1993) used a standard PCFG but could improve parsing performance considerably by using a history-based model for bottom-up construction of leftmost derivations. In more recent developments, the history-based model has replaced the grammar completely, as in the parsers of Collins (1997; 1999) and Charniak (2000).

With a generative probabilistic model, the parameters that need to be estimated are the conditional probabilities $P(d_i \mid \Phi(d_1, \ldots, d_{i-1}))$, for every possible decision d_i and non-equivalent history $H_i = \Phi(d_1, \ldots, d_{i-1})$. With a deterministic parsing strategy, we only need to estimate the mode of each conditional distribution, i.e., $\arg\max_{d_i} P(d_i \mid \Phi(d_1, \ldots, d_{i-1}))$. This reduces the parameter estimation problem to that of learning a classifier, where the classes are the possible decisions of the parser, e.g., the possible transitions of the algorithm described in the previous section.

Distinct parser histories are normally represented as sequences of attributes, so-called *feature vectors*, and the function Φ, referred to as the *feature model*, can therefore be defined in terms of a sequence $\Phi_{1,p} = (\phi_1, \ldots, \phi_p)$ of *feature functions*, where each function ϕ_i identifies some relevant feature of the history. The most important features in dependency parsing are the attributes of input tokens, such as their word form, part-of-speech or dependency type, and we will in fact limit ourselves in this article to features that can be defined as simple attributes of tokens.

Token attributes can be divided into *static* and *dynamic* attributes, where static attributes are properties that remain constant during the parsing of a sentence. This primarily includes the actual word form of a token, but also any kind of annotation that is the result of preprocessing, such as part-of-speech tag, lemma, or word sense annotation. In this article, we restrict our attention to two kinds of static attributes, word form and part-of-speech. Given a sentence $x = (w_1, \ldots, w_n)$, with part-of-speech annotation, we use $w(i)$ and $p(i)$ to refer to the word form and part-of-speech, respectively, of the ith token. We will also make use of fixed-length suffixes of word forms and write $s_m(w(i))$ for the m-character suffix of $w(i)$ (where $s_m(w(i)) = w(i)$ if $w(i)$ has length $l \leq m$).

Dynamic attributes, by contrast, are attributes that are defined by the partially built dependency graph, which in this article will be limited to the dependency type by which a token is related to its head, given by the function d of the current parser configuration $c = (\sigma, \tau, h, d)$.

To define complex history-based feature models, we need to refer to attributes of arbitrary tokens in the parser history, represented by the current parser configuration. For this purpose, we introduce a set of address functions.

Definition 7
Given a sentence $x = (w_1, \ldots, w_n)$ and a parser configuration $c = (\sigma, \tau, h, d)$ for x:

1. σ_i is the ith token from the top of the stack (starting at index 0).
2. τ_i is the ith token in the remaining input (starting at index 0).
3. $h(i)$ is the head of token i in the graph defined by h.
4. $l(i)$ is the leftmost child of token i in the graph defined by h.
5. $r(i)$ is the rightmost child of token i in the graph defined by h.

By combining these functions, we can define arbitrarily complex functions that identify tokens relative to a given parser configuration c. For example, while σ_0 is the token on top of the stack, $h(\sigma_0)$ is the head of the token on top of the stack, and $l(h(\sigma_0))$ is the leftmost dependent of the head of the token on top of the stack. It should be noted that these functions are generally partial functions on token nodes, which means that if one of the inner functions in a chain of applications returns 0 (because $h(i) = 0$) or is undefined (because the stack is empty, or a token does not have a leftmost child, etc.), then the outermost function is always undefined.

Finally, we can now define feature functions by applying attribute functions to complex combinations of address functions. For example, $p(\tau_0)$ is the part-of-speech of the next input token, while $d(h(\sigma_0))$ is the dependency type of the head of the token on top of the stack, which may or may not be defined in a given configuration. Any feature function that is undefined for a given configuration, because the complex address function fails to identify a token, is assigned a special *nil* value. Feature models used for inductive dependency parsing typically combine static part-of-speech features and lexical features (or suffix features) with dynamic dependency type features. The kind of models used in the experiments later on are described in section 3.2 below.

2.4 Discriminative machine learning

Given a function approximation problem with labeled training data from target function $f : X \rightarrow Y$, discriminative learning methods attempt to optimize the mapping from inputs $x \in X$ to outputs $y \in Y$ directly, without estimating a full generative model of the joint distribution of X and Y. Discriminatively trained models have in recent years been shown to outperform generative models for many problems in natural language processing, including syntactic parsing, by directly estimating a conditional probability distribution $P(Y|X)$ (Johnson *et al.* 1999; Collins 2000; Collins and Duffy 2002; Collins and Koo 2005; Charniak and Johnson 2005). With a deterministic parsing strategy, the learning problem can be further reduced to a pure classification problem, where the input instances are histories (represented by feature vectors) and the output classes are parsing decisions.

Thus, the training data for the learner consists of pairs $(\Phi(c), t)$, where $\Phi(c)$ is the representation of a parser configuration defined by the feature model $\Phi(c)$ and t is the correct transition out of c. Such data can be generated from a treebank of gold standard dependency graphs, by reconstructing the correct transition sequence for each dependency graph in the treebank and extracting the appropriate feature vectors for each configuration, as described in detail by Nivre (2006) for the parsing algorithm discussed in section 2.2.

Although in principle any learning algorithm capable of inducing a classifier from labeled training data can be used to solve the learning problem posed by inductive dependency parsing, most of the work done in this area has been based on support vector machines (SVM) and memory-based learning (MBL).[6]

SVM is a hyperplane classifier that relies on the maximum margin strategy introduced by Vapnik (1995). Furthermore, it allows the use of kernel functions to map the original feature space to a higher-dimensional space, where the classification problem may be (more) linearly separable. In dependency parsing, SVM has been used primarily by Matsumoto and colleagues (Kudo and Matsumoto 2002; Yamada and Matsumoto 2003; Cheng *et al.* 2004; Cheng *et al.* 2005).

MBL is a lazy learning method, based on the idea that learning is the simple storage of experiences in memory and that solving a new problem is achieved by reusing solutions from similar previously solved problems (Daelemans and Van den Bosch 2005). In essence, this is a k nearest neighbor approach to classification, although a variety of sophisticated techniques, including different distance metrics and feature weighting schemes can be used to improve classification accuracy. In dependency parsing, MBL has been used primarily by Nivre and colleagues (Nivre *et al.* 2004; Nivre and Scholz 2004; Nivre and Nilsson 2005), and it is also the learning method that is used for the experiments in this article.

3 MaltParser

MaltParser is an implementation of inductive dependency parsing, as described in the previous section, where the syntactic analysis of a sentence amounts to

[6] In addition, maximum entropy modeling was used in the comparative evaluation of Cheng *et al.* (2005).

the deterministic derivation of a dependency graph, and where discriminative machine learning is used to guide the parser at nondeterministic choice points, based on a history-based feature model. MaltParser can also be characterized as a data-driven parser-generator. While a traditional parser-generator constructs a parser given a grammar, a data-driven parser-generator constructs a parser given a treebank.

The system can be run in two basic modes. In learning mode, it takes as input a (training) set of sentences with dependency graph annotations, derives training data by reconstructing the correct transition sequences, and trains a classifier on this data set according to the specifications of the user. In parsing mode, it takes as input a (test) set of sentences and a previously trained classifier and parses the sentences using the classifier as a guide.

3.1 Parsing algorithms

MaltParser provides two main parsing algorithms, each with several options:

- The linear-time algorithm of Nivre (2003) can be run in arc-eager or arc-standard mode. The arc-standard version is similar to but not identical to the algorithm of Yamada and Matsumoto (2003), since the latter also uses multiple passes over the input (Nivre 2004). In both versions, this algorithm is limited to projective dependency graphs.
- The incremental algorithm of Covington (2001) can be run in projective or non-projective mode. In the latter case, graphs are still guaranteed to obey the constraints ROOT, CONNECTEDNESS, SINGLE-HEAD and ACYCLICITY.

The experiments reported in this article are all based on the arc-eager version of Nivre's algorithm.

3.2 Feature models

MaltParser allows the user to define arbitrarily complex feature models, using address functions and attribute functions as described in section 2.3.[7] The standard model used in most of the experiments reported below combines part-of-speech features, lexical features and dependency type features in the following way:

$p(\sigma_1)$	$w(h(\sigma_0))$	$d(l(\sigma_0))$
$p(\sigma_0)$	$w(\sigma_0)$	$d(\sigma_0)$
$p(\tau_0)$	$w(\tau_0))$	$d(r(\sigma_0))$
$p(\tau_1)$	$w(\tau_1)$	$d(l(\tau_0))$
$p(\tau_2)$		
$p(\tau_3)$		

[7] The feature models supported by MaltParser are in fact slightly more general in that they also allow address functions that refer to siblings. This option is not exploited in the experiments reported below and has therefore been excluded from the presentation in section 2.3.

This model includes six part-of-speech features, defined by the part-of-speech of the two topmost stack tokens ($p(\sigma_0)$, $p(\sigma_1)$) and by the first four tokens of the remaining input ($p(\tau_0)$, $p(\tau_1)$, $p(\tau_2)$, $p(\tau_3)$). The dependency type features involve the top token on the stack ($d(\sigma_0)$), its leftmost and rightmost dependent ($d(l(\sigma_0))$, $d(r(\sigma_0))$), and the leftmost child of the next input token ($d(l(\tau_0))$).[8] Finally, the standard model includes four lexical features, defined by the word form of the top token on the stack ($w(\sigma_0)$), the head of the top token ($w(h(\sigma_0))$), and the next two input tokens ($w(\tau_0)$, $w(\tau_1)$).

The standard model can be seen as the prototypical feature model used in the experiments reported below, although the tuned models for some languages deviate from it by adding or omitting features, or by replacing lexical features by suffix features (the latter not being used at all in the standard model). Deviations from the standard model are specified in table 3 below.

3.3 Learning algorithms

MaltParser provides two main learning algorithms, each with a variety of options:

- Memory-based learning (MBL) using TiMBL, a software package for memory-based learning and classification developed by Daelemans, Van den Bosch and colleagues at the Universities of Tilburg and Antwerp (Daelemans and Van den Bosch 2005).
- Support vector machines (SVM) using LIBSVM, a library for SVM learning and classification developed by Chang and Lin at National Taiwan University (Chang and Lin 2001).

The experiments reported in this paper are all based on MBL and make crucial use of the following features of TiMBL:

- Varying the number k of nearest neighbors
- Using the Modified Value Difference Metric (MVDM) for distances between feature values (for values seen more than l times)
- Distance-weighted class voting for determining the majority class

The optimal values for these parameters vary for different feature models, languages and data sets, but typical values are $k = 5$, MVDM down to $l = 3$ (with the simple Overlap metric for lower frequencies), and class voting weighted by inverse distance (ID). For more information about these and other TiMBL features, we refer to Daelemans and Van den Bosch (2005).

3.4 Auxiliary tools

MaltParser is supported by a suite of freely available tools for, among other things, parser evaluation and treebank conversion. Of special interest in this context are

[8] It is worth pointing out that, given the nature of the arc-eager parsing algorithm, the dependency type of the next input token and its rightmost child will always be undefined at decision time (hence their omission in the standard model and all other models).

the tools for pseudo-projective dependency parsing (Nivre and Nilsson 2005). This is a method for recovering non-projective dependencies through a combination of data-driven projective dependency parsing and graph transformation techniques in the following way:

1. Dependency graphs in the training data sets are transformed (if necessary) to projective dependency graphs, by minimally moving non-projective arcs upwards towards the root and encoding information about these transformations in arc labels.
2. The projective parser is trained as usual, except that the dependency graphs in the training set are labeled with the enriched arc labels.
3. New sentences are parsed into projective dependency graphs with enriched arc labels.
4. Dependency graphs produced by the parser are transformed (if possible) to non-projective dependency graphs, using an inverse transformation guided by information in the arc labels.

This methodology has been used in a few of the experiments reported below, in particular for the parsing of Czech (section 4.2.5).

4 Experimental evaluation

In this section, we summarize experiments with the MaltParser system on data from ten different languages: Bulgarian, Chinese, Czech, Danish, Dutch, English, German, Italian, Swedish and Turkish.[9] Although the group is dominated by Indo-European languages, in particular Germanic languages, the languages nevertheless represent fairly different language types, ranging from Chinese and English, with very reduced morphology and relatively inflexible word order, to languages like Czech and Turkish, with rich morphology and flexible word order, and with Bulgarian, Danish, Dutch, German, Italian and Swedish somewhere in the middle. In addition, the treebank annotation schemes used to analyze these languages differ considerably. Whereas the treebanks for Czech, Danish, Italian and Turkish are proper dependency treebanks, albeit couched in different theoretical frameworks, the annotation schemes for the remaining treebanks are based on constituency in combination with grammatical functions, which necessitates a conversion from constituent structures to dependency structures.

Below we first describe the general methodology used to evaluate the system, in particular the evaluation metrics used to assess parsing accuracy, and give an overview of the different data sets and experiments performed for different languages (section 4.1). This is followed by a presentation of the results (section 4.2), with specific subsections for each language (section 4.2.1–4.2.10), where we also give a more detailed description of the respective treebanks and the specific settings

[9] Results have been published previously for Swedish (Nivre *et al.* 2004; Nivre 2006), English (Nivre and Scholz 2004; Nivre 2006), Czech (Nivre and Nilsson 2005), Bulgarian (Marinov and Nivre 2005), Danish (Nivre and Hall 2005) and Italian (Chanev 2005) but not for Chinese, German and Turkish.

Table 1. *Data sets. AS = Annotation scheme (C = Constituency, D = Dependency, G = Grammatical functions); Pro = Projective; #D = Number of dependency types; #P = Number of PoS tags; TA =Tagging accuracy; #W = Number of words; #S = Number of sentences; SL = Mean sentence length; EM = Evaluation method (T = Held-out test set, CV_k = k-fold cross-validation)*

Language	AS	Pro	#D	#P	TA	#W	#S	SL	EM
Bulgarian	C	no	14	51	93.5	72k	5.1k	14.1	CV_8
Chinese	CG	yes	12	35	100.0	509k	18.8k	27.1	T
Czech	D	no	26	28	94.1	1507k	87.9k	17.2	T
Danish	D	no	54	33	96.3	100k	5.5k	18.2	T
Dutch	CD	no	23	165	95.7	186k	13.7k	13.6	T
English	CG	yes	12	48	96.1	1174k	49.2k	23.8	T
German	CG	no	31	55	100.0	382k	22.1k	17.3	CV_{10}
Italian	D	no	17	89	93.1	42k	1.5k	27.7	CV_{10}
Swedish	CG	yes	17	46	95.6	98k	6.3k	15.5	T
Turkish	D	no	24	484	100.0	48k	5.6k	8.6	CV_{10}

used for individual experiments, followed by a general discussion, where we bring together the results from different languages and try to discern some general trends (section 4.3).

4.1 Method

Table 1 gives an overview of the data sets for the ten languages. The first column characterizes the annotation scheme and the second indicates whether the (possibly converted) annotation is restricted to projective dependency graphs. The next two columns contain the number of distinct dependency types and part-of-speech tags, respectively, where the latter refers to the tagset actually used in parsing, which may be a reduced version of the tagset used in the original treebank annotation. The fifth column gives the mean accuracy of the part-of-speech tagging given as input to the parser, where 100.0 indicates that experiments have been performed using gold standard tags (i.e., manually assigned or corrected tags) rather than the output of an automatic tagger. The next three columns give the number of tokens and sentences, and the mean number of words per sentence. These figures refer in each case to the complete treebank, of which at most 90% has been used for training and at least 10% for testing (possibly using *k*-fold cross-validation).

Table 2 gives a little more information about the syntactic analysis adopted in the different treebank annotation schemes. Whereas all the schemes agree on basic structures such as verbs taking their core arguments as dependents and adjuncts being dependents of the heads they modify, there are a number of constructions that have competing analyses with respect to their dependency structure. This holds in particular for constructions involving function words, such as auxiliary verbs, prepositions, determiners, and complementizers, but also for the ubiquitous phenomenon of coordination. Table 2 shows the choices made for each of these cases in the different treebanks, and we see that there is a fair amount of variation

Table 2. *Annotation style (choice of head). VG = Verb group (Aux = Auxiliary verb, MV = Main verb); AP = Adpositional phrase (Ad = Adposition, N = Nominal head); NP = Noun phrase (Det = Determiner, N = Noun); SC = Subordinate clause (Comp = Complementizer, V = Verb); Coord = Coordination (CC = Coordinating conjunction, $Conj_1$ = First conjunct, $Conj_n$ = Last conjunct); NA = Not applicable*

Language	VG	AP	NP	SC	Coord
Bulgarian	MV	Ad	N	Comp	$Conj_1$
Chinese	Aux	Ad	N	Comp	$Conj_1/Conj_n$
Czech	MV	Ad	N	V	CC
Danish	Aux	Ad	Det	Comp	$Conj_1$
Dutch	Aux	Ad	N	Comp	CC
English	Aux	Ad	N	Comp	$Conj_1/Conj_n$
German	Aux	Ad	N	V	$Conj_1$
Italian	MV	Ad	Det	Comp	$Conj_1$
Swedish	Aux	Ad	N	Comp	$Conj_1$
Turkish	NA	(Ad)	N	NA	$Conj_n$

especially with respect to verb groups and coordination.[10] It is worth noting that for Turkish, which is a richly inflected, agglutinative language, some of the distinctions are not applicable, since the relevant construction is encoded morphologically rather than syntactically.[11] It is also important to remember that, for the treebanks that are not originally annotated with dependency structures, the analysis adopted here only represents one conversion out of several possible alternatives. More information about the conversions are given for each language below.

All the experiments reported in this article have been performed with the parsing algorithm described in Nivre (2003; 2004; 2006) and with memory-based learning and classification as implemented in the TiMBL software package by Daelemans and Van den Bosch (2005). A variety of feature models have been tested, but we only report results for the optimal model for each language, which is characterized in relation to the standard model defined in section 3.2. Standard settings for the TiMBL learner include $k = 5$ (number of nearest distances), MVDM metric down to a threshold of $l = 3$, and distance weighted class voting with Inverse Distance weights (ID).

Final evaluation has been performed using either k-fold cross-validation or a held-out test set, as shown in the last column in table 1. Evaluation on held-out data has in turn been preceded by a tuning phase using either k-fold cross-validation or a development test set, as described for each language below. The diversity in evaluation methods is partly a result of practical circumstances and partly motivated by the concern to make results comparable to previously published results for a

[10] The notation $Conj_1/Conj_n$ under Coord for Chinese and English signifies that coordination is analyzed as a head-initial or head-final construction depending on whether the underlying phrase type is head-initial (e.g., verb phrases) or head-final (e.g., noun phrases).

[11] Whereas postpositions generally appear as suffixes on nouns, there are marginal cases where they occur as separate words and are then treated as heads. Hence, the brackets around Ad in the AP column for Turkish.

given language. Thus, while results on the Penn Treebank are customarily obtained by training on sections 2–21 and testing on section 23 (using any of the remaining sections as a development test set), results on the Turkish Treebank have so far been based on ten-fold cross-validation, which is well motivated by the limited amount of data available. It should also be noted that the amount of work devoted to model selection and parameter optimization varies considerably between the languages, with Swedish and English being most thoroughly investigated while the results for other languages, notably Dutch, German and Turkish, are still preliminary and can probably be improved substantially.

The evaluation metrics used throughout are the *unlabeled attachment score* AS_U, which is the proportion of tokens that are assigned the correct head (regardless of dependency type), and the *labeled attachment score* AS_L, which is the proportion of tokens that are assigned the correct head and the correct dependency type, following the proposal of Lin (1998). All results are presented as mean scores per token, with punctuation tokens excluded from all counts.[12] For each language, we also provide a more detailed breakdown with (unlabeled) attachment score, precision, recall and F measure for individual dependency types.

Before we turn to the experimental results, a caveat is in order concerning their interpretation and in particular about cross-linguistic comparability. The main point of the experimental evaluation is to corroborate the claim that MaltParser is language-independent enough to achieve reasonably accurate parsing for a wide variety of languages, where the level of accuracy is related, whenever possible, to previously obtained results for that language. In order to facilitate this kind of comparison, we have sometimes had to sacrifice comparability between languages, notably by using training sets of different size or different procedures for obtaining accuracy scores as explained earlier. This means that, even though we sometimes compare results across languages, such comparisons must be taken with a pinch of salt. Although a more controlled cross-linguistic comparison would be very interesting, it is also very difficult to achieve given that available resources are very diverse with respect to standards of annotation, the amount of annotated data available, the existence of accurate part-of-speech taggers, etc. Faced with this diversity, we have done our best to come up with a reasonable compromise between the conflicting requirements of ensuring cross-linguistic comparability and being faithful to existing theoretical and practical traditions for specific languages and treebanks. This means, for example, that we retain the original arc labels for all treebanks, so that users of these treebanks can easily relate our results to theirs, even though this has the consequence that, e.g., subjects will be denoted by a variety of labels such as SUB, SBJ, SUBJ and Sb, but all arc labels will be accompanied by descriptions that should make them understandable also for readers who are not familiar with a given treebank annotation scheme.

[12] Although punctuation tokens are excluded in the calculation of accuracy scores, they are *included* during parsing. No changes have been made to the tokenization or sentence segmentation found in the respective treebanks, except for Turkish (see section 4.2.10).

Table 3. *Overview of results. Model = Best feature model (− = omitted, + = added, → = replaced by); Settings =* TiMBL *settings; AS_U = Unlabeled attachment score; AS_L = Labeled attachment score*

Language	Model	Settings	AS_U	AS_L
Bulgarian	$\forall a[w(a) \rightarrow s_6(w(a))]$	Standard	81.3	73.6
Chinese	Standard	$k = 6$, $l = 8$	81.1	79.2
Czech	Standard	Standard	80.1	72.8
Danish	$[w(h(\sigma_0)) \rightarrow s_6(w(h(\sigma_0)))); -w(\tau_1)]$	Standard	85.6	79.5
Dutch	Standard	$k = 10$	84.7	79.2
English	Standard	$k = 7$, $l = 5$	88.1	86.3
German	$[-w(h(\sigma_0)); -w(\tau_1); +p(\sigma_2)]$	$k = 13$, IL	88.1	83.4
Italian	Standard	Standard	82.9	75.7
Swedish	Standard	Standard	86.3	82.0
Turkish	$[-p(\sigma_1); -p(\tau_2); -p(\tau_3); -w(h(\sigma_0)); -w(\tau_1)]$	Standard	81.6	69.0

4.2 Results

Table 3 gives an overview of the results, summarizing for each language the optimal feature model and TiMBL parameter settings, as well as the best unlabeled and labeled attachment scores. In the following subsections, we analyze the results for each language in a little more detail, making state-of-the-art comparisons where this is possible. The earliest experiments were those performed on Swedish and English and the standard models and settings are mainly based on the results of these experiments. It is therefore natural to treat Swedish and English first, with the remaining languages following in alphabetical order.

4.2.1 Swedish

The Swedish data come from Talbanken (Einarsson 1976), a manually annotated corpus of both written and spoken Swedish, created at Lund University in the 1970s. We use the professional prose section, consisting of material taken from textbooks, newspapers and information brochures. Although the original annotation scheme is an eclectic combination of constituent structure, dependency structure, and topological fields (Teleman 1974), it has been possible to convert the annotated sentences to dependency graphs with very high accuracy. In the conversion process, we have reduced the original fine-grained classification of grammatical functions to a more restricted set of 17 dependency types, mainly corresponding to traditional grammatical functions such as *subject*, *object* and *adverbial*. We have used a pseudo-randomized data split, dividing the data into 10 sections by allocating sentence i to section $i \bmod 10$. We have used sections 1–9 for 9-fold cross-validation during development and section 0 for final evaluation.

The overall accuracy scores for Swedish, obtained with the standard model and standard settings, are $AS_U = 86.3\%$ and $AS_L = 82.0\%$. Table 4 gives unlabeled attachment score (AS_U), labeled precision (P), recall (R) and F measure (F) for individual dependency types in the Swedish data. These types can be divided into three groups according to accuracy. In the high-accuracy set, with a labeled F

Table 4. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Swedish (held-out test set, section 0)*

Dependency Type	n	AS_U	P	R	F
Adverbial (ADV)	1607	79.8	75.8	76.8	76.3
Apposition (APP)	42	23.8	38.1	19.0	25.4
Attribute (ATT)	950	81.3	79.9	78.5	79.2
Coordination (CC)	963	82.5	78.1	79.8	78.9
Determiner (DET)	947	92.6	88.9	90.2	89.5
Idiom (ID)	254	72.0	72.5	58.3	64.6
Infinitive marker (IM)	133	98.5	98.5	98.5	98.5
Infinitive complement (INF)	10	100.0	100.0	30.0	46.2
Object (OBJ)	585	88.0	78.2	77.3	77.7
Preposition dependent (PR)	985	94.2	88.6	92.7	90.6
Predicative (PRD)	244	90.6	76.7	77.0	76.8
Root (ROOT)	607	91.3	84.6	91.3	87.8
Subject (SUB)	957	89.8	86.7	82.5	84.5
Complementizer dependent (UK)	213	85.0	89.4	83.6	86.4
Verb group (VC)	238	93.7	82.1	90.6	86.1
Other (XX)	29	82.8	85.7	20.7	33.3
Total	8764	86.3	82.0	82.0	82.0

measure from 84% to 98%, we find all dependency types where the head is a closed class word: IM (marker → infinitive), PR (preposition → noun), UK (complementizer → verb) and VC (auxiliary verb → main verb). We also find the type DET (noun → determiner), which has similar characteristics although the determiner is not treated as the head in the Swedish annotation. The high-accuracy set also includes the central dependency types ROOT and SUB, which normally identify the finite verb of the main clause and the grammatical subject, respectively.

In the medium-accuracy set, with a labeled F measure in the range of 75–80%, we find the remaining major dependency types, ADV (adverbial), ATT (nominal modifier), CC (coordination), OBJ (object) and PRD (predicative). However, this set can be divided into two subsets, the first consisting of ADV, ATT and CC, which have an unlabeled attachment score not much above the labeled F measure, indicating that parsing errors are mainly due to incorrect attachment. This is plausible since ADV and ATT are the dependency types typically involved in modifier attachment ambiguities and since coordination is also a source of attachment ambiguities. The second subset contains OBJ and PRD, which both have an unlabeled attachment score close to 90%, which means that they are often correctly attached but may be incorrectly labeled. This is again plausible, since these types identify nominal arguments of the verb (other than the subject), which can often occur in the same structural positions.

Finally, we have a low-accuracy set, with a labeled F measure below 70%, where the common denominator is mainly low frequency: INF (infinitive complements), APP (appositions), XX (unclassifiable). The only exception to this generalization is the type ID (idiom constituent), which is not that rare but which is rather special for other reasons. All types in this set except APP have a relatively high unlabeled attachment score, but their labels are seldom used correctly.

Table 5. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for English (held-out test set, section 23)*

Dependency Type	*n*	AS_U	P	R	F
Adjective/adverb modifier (AMOD)	2072	78.2	80.7	73.0	76.7
Other (DEP)	259	42.9	56.5	30.1	39.3
Noun modifier (NMOD)	21002	91.2	91.1	90.8	91.0
Object (OBJ)	1960	86.5	78.9	83.5	81.1
Preposition modifier (PMOD)	5593	90.2	87.7	89.5	88.6
Predicative (PRD)	832	90.0	75.9	71.8	73.8
Root (ROOT)	2401	86.4	78.8	86.4	82.4
Complementizer dependent (SBAR)	1195	86.0	87.1	85.1	86.1
Subject (SBJ)	4108	90.0	90.6	88.1	89.3
Verb group (VC)	1771	98.8	93.4	96.6	95.0
Adverbial (VMOD)	8175	80.3	76.5	77.1	76.8
Total	49368	88.1	86.3	86.3	86.3

Relating the Swedish results to the state of the art is rather difficult, since there is no comparable evaluation reported in the literature, let alone based on the same data. Voutilainen (2001) presents a partial and informal evaluation of a Swedish FDG parser, based on manually checked parses of about 400 sentences from newspaper text, and reports F measures of 95% for subjects and 92% for objects. These results clearly indicate a higher level of accuracy than that attained in the experiments reported here, but without knowing the details of the data selection and evaluation procedure it is very difficult to draw any precise conclusions.

4.2.2 English

The data set used for English is the standard data set from the Wall Street Journal section of the Penn Treebank, with sections 2–21 used for training and section 23 for testing (with section 00 as the development test set). The data has been converted to dependency trees using the head percolation table of Yamada and Matsumoto (2003), and dependency type labels have been inferred using a variation of the scheme employed by Collins (1999), which makes use of the nonterminal labels on the head daughter, non-head daughter and parent corresponding to a given dependency relation. However, instead of simply concatenating these labels, as in the Collins scheme, we use a set of rules to map these complex categories onto a set of 10 dependency types, including traditional grammatical functions such as *subject*, *object*, etc. More details about the conversion can be found in Nivre (2006).

The best performing model for English is the standard model and the TiMBL parameter settings deviate from the standard ones only by having a higher k value ($k = 7$) and a higher threshold for MVDM ($l = 5$). The overall accuracy scores for English are $AS_U = 88.1\%$ and $AS_L = 86.3\%$. The relatively narrow gap between unlabeled and labeled accuracy is probably due mainly to the coarse-grained nature of the dependency type set and perhaps also to the fact that these labels have been inferred automatically from phrase structure representations. Table 5 shows the accuracy for individual dependency types in the same way as for Swedish in

table 4, and again we can divide dependency types according to accuracy into three sets. In the high-accuracy set, with a labeled F measure from 86% to 95%, we find SBJ (subject) and three dependency types where the head is a closed class word: PMOD (preposition → complement/modifier), VC (auxiliary verb → main verb) and SBAR (complementizer → verb). In addition, this set includes the type NMOD, which includes the noun-determiner relation as an important subtype.

In the medium-accuracy set, with a labeled F measure from 74% to 82%, we find the types AMOD, VMOD, OBJ, PRD and ROOT. The former two dependency types mostly cover adverbial functions, and have a labeled accuracy not too far below their unlabeled attachment score, which is an indication that the main difficulty lies in finding the correct head. By contrast, the argument functions OBJ and PRD have a much better unlabeled attachment score, which shows that they are often attached to the correct head but misclassified. This tendency is especially pronounced for the PRD type, where the difference is more than 15 percentage points, which can probably be explained by the fact that this type is relatively infrequent in the annotated English data. The low-accuracy set for English only includes the default classification DEP. The very low accuracy for this dependency type can be explained by the fact that it is both a heterogeneous category and the least frequent dependency type in the data.

Compared to the state of the art, the unlabeled attachment score is about 4% lower than the best reported results, obtained with the parser of Charniak (2000) and reported in Yamada and Matsumoto (2003).[13] For the labeled attachment score, we are not aware of any strictly comparable results, but Blaheta and Charniak (2000) report an F measure of 98.9% for the assignment of grammatical role labels to phrases that were correctly parsed by the parser described in Charniak (2000), using the same data set. If null labels are excluded, the F score drops to 95.6%. The corresponding F measures for MaltParser are 98.0% and 97.8%, treating the default label DEP as the equivalent of a null label. The experiments are not strictly comparable, since they involve different sets of functional categories (where only the labels SBJ and PRD are equivalent) and one is based on phrase structure and the other on dependency structure, but it nevertheless seems fair to conclude that MaltParser's labeling accuracy is close to the state of the art, even if its capacity to derive correct structures is not.

4.2.3 Bulgarian

For the current experiments we used a subset of BulTreeBank (Simov *et al.* 2002), since the complete treebank is not officially released and still under development. The set contains 71703 words of Bulgarian text from different sources, annotated with constituent structure. Although the annotation scheme is meant to be compatible with the framework of HPSG, syntactic heads are not explicitly annotated, which

[13] The score for the Charniak parser has been obtained by converting the output of the parser to dependency structures using the same conversion as in our experiments, which means that the comparison is as exact as possible. For further comparisons, see Nivre (2006).

Table 6. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Bulgarian (mean of 8-fold cross-validation, frequency counts rounded to whole integers)*

Dependency Type	*n*	AS_U	P	R	F
Adverbial (ADV)	914	67.2	59.4	51.2	55.0
Apposition (APP)	120	65.5	54.1	49.0	51.9
Attribute (ATT)	1297	79.6	74.0	75.4	74.7
Coordination (CC)	555	53.6	52.8	48.5	50.6
Determiner (DET)	259	82.9	80.2	76.5	78.3
Idiom (ID)	214	94.6	90.2	89.5	89.8
Object (OBJ)	949	85.9	66.9	70.4	68.6
Preposition dependent (PR)	1137	93.6	91.8	93.2	92.5
Predicative (PRD)	254	89.8	65.7	73.3	69.3
Root (ROOT)	635	88.7	76.8	88.7	82.3
Subject (SUBJ)	600	82.7	68.9	66.8	67.8
Complementizer dependent (UK)	418	88.1	87.5	88.7	88.1
Verb group (VC)	397	79.8	71.2	72.5	71.8
Total	7748	81.3	73.6	73.6	73.6

means that the treebank must be converted to dependency structures using the same kind of head percolation tables and inference rules that were used for the English data, except that for Bulgarian the converted treebank also contains non-projective dependencies. In most cases, these involve subordinate *da*-clauses, where we often find subject-to-object or object-to-object raising. In these cases, we have taken *da* to be the head of the subordinate clause with the main verb dependent on *da* and the raised subject or object dependent on the main verb. More details about the conversion can be found in Marinov and Nivre (2005).

Experiments were performed with several models but the highest accuracy was achieved with a variant of the standard model, where all lexical features are based on suffixes of length 6, rather than the full word forms. That is, every lexical feature $w(a)$ (with address function a) is replaced by $s_6(w(a))$ (cf. section 2.3). The overall accuracy scores for Bulgarian are 81.3% (AS_U) and 73.6% (AS_L). Using suffixes instead of full forms makes the data less sparse, which can be an advantage for languages with limited amounts of data, especially if the endings of content words can be expected to carry syntactically relevant information. The optimal suffix length can be determined using cross-validation, and a length of 6 seems to work well for several languages, presumably because it captures the informative endings of content words while leaving most function words intact.

Table 6 gives accuracy, precision, recall and balanced F measures for individual dependency types. The overall trend is the same as for Swedish and English in that dependency relations involving function words tend to have higher accuracy than relations holding primarily between content words. Thus, the highest ranking dependency types with respect to the F measure are PR (preposition → noun) and UK (complementizer → verb), together with ID (multi-word unit), which in the Bulgarian data includes verbs taking the reflexive/possessive pronouns *se* and *si*. Further down the list we find as expected the major verb complement types OBJ (object) and PRD (predicative complement) but also SUBJ (subject), which has

considerably lower accuracy than the corresponding type in Swedish and English. This is a reflection of the more flexible word order in Bulgarian.

Other dependency types that are ranked lower for Bulgarian than for the other languages considered so far are DET (noun → determiner) and VC (auxiliary verb ← main verb). In the former case, since Bulgarian lacks free-standing determiners like English *the*, this category was reserved for demonstratives (*this*, *that*, etc.), which occurred infrequently. In the latter case, this again seems to be related to word order properties, allowing the verbs to be separated by adverbials or even subordinate clauses (which will also lead the parser to erroneously connect verbs that belong to different clauses). Finally, we note that coordinate structures (CC) and adverbials (ADV) have very low accuracy (with an F measure below 60%). For adverbials, one possible error source is the fact that many adverbs coincide in form with the third person singular form of adjectives.

There are no other published results for parsing Bulgarian, except for a paper by Tanev and Mitkov (2002), who report precision and recall in the low 60s for a rule-based parser. However, this parser has only been tested on 600 syntactic phrases, as compared to the 5080 sentences used in the present study, so it is very difficult to draw any conclusions about the relative quality of the parsers.

4.2.4 Chinese

The Chinese data are taken from the Penn Chinese Treebank (CTB), version 5.1 (Xue *et al.* 2005), and the texts are mostly from Xinhua newswire, Sinorama news magazine and Hong Kong News. The annotation of CTB is based on a combination of constituent structure and grammatical functions and has been converted in the same way as the data for English and Bulgarian, with a head percolation table created by a native speaker for the purpose of machine translation. Dependency type labels have been inferred using an adapted version of the rules developed for English, which is possible given that the treebank annotation scheme for CTB is modeled after that for the English Penn Treebank. More details about the conversion can be found in Hall (2006).

One often underestimated parameter in parser evaluation is the division of data into training, development and evaluation sets. Levy and Manning (2003) report up to 10% difference in parsing accuracy for different splits of CTB 2.0. We have used the same pseudo-randomized split as for Swedish (cf. section 4.2.1), with sections 1–8 for training, section 9 for validation, and section 0 for final evaluation. The results presented in this article are based on gold-standard word segmentation and part-of-speech tagging.

The best performing model for Chinese is the standard one and the same goes for TiMBL settings except that $k = 6$ and $l = 8$. Table 7 presents the unlabeled attachment score (AS_U), labeled precision (P), recall (R) and F measure (F) for individual dependency types in the Chinese data. We see that the overall accuracy scores for Chinese are $AS_U = 81.1\%$ and $AS_L = 79.2\%$, and the difference between labeled and unlabeled accuracy is generally very small also on the level of individual dependency types, with a few notable exceptions. Both SBJ (subject) and VC (verb

Table 7. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Chinese (held-out test set, section 0)*

Dependency Type	*n*	AS_U	P	R	F
Adjective/adverb modifier (AMOD)	1503	95.2	95.8	94.5	95.1
Other (DEP)	2999	90.5	92.4	89.5	90.9
Noun modifier (NMOD)	13046	85.4	86.3	85.2	85.7
Object (OBJ)	2802	86.0	82.8	85.3	84.0
Preposition modifier (PMOD)	1839	77.3	81.3	77.2	79.2
Predicative (PRD)	467	78.8	81.4	76.0	78.6
Root (ROOT)	1880	70.5	55.4	70.5	62.0
Complementizer dependent (SBAR)	1296	83.6	83.6	83.3	83.4
Subject (SBJ)	3242	83.2	73.3	78.5	75.8
Verb group (VC)	940	80.0	76.0	75.1	75.5
Adverbial (VMOD)	12043	72.6	71.3	68.8	70.0
Total	42057	81.1	79.2	79.2	79.2

chain) have considerably lower labeled F measure than unlabeled attachment score, which indicates that these relations are difficult to classify correctly even if the head-dependent relations are assigned correctly. For the special ROOT label, we find a very low precision, which reflects fragmentation in the output (since too many tokens remain attached to the special root node), but even the recall is substantially lower than for any other language considered so far. This may indicate that the feature model has not yet been properly optimized for Chinese, but it may also indicate a problem with the arc-eager parsing strategy adopted in all the experiments.

It is rather difficult to compare results on parsing accuracy for Chinese because of different data sets, word segmentation strategies, dependency conversion methods, and data splits. But the unlabeled attachment score obtained in our experiments is within 5% of the best reported results for CTB (Cheng *et al.* 2005).

4.2.5 Czech

The Prague Dependency Treebank (PDT) consists of 1.5M words of newspaper text, annotated on three levels, the morphological, analytical and tectogrammatical levels (Hajič *et al.* 2001). Our experiments all concern the analytical annotation, which uses a set of 28 surface-oriented grammatical functions (Böhmová *et al.* 2003). Unlike the treebanks discussed so far, PDT is a genuine dependency treebank also including non-projective dependencies.

The best results for Czech are again based on the standard model with standard settings, although it should be acknowledged that the sheer size of the Czech data sets makes it hard to perform extensive optimization of feature model and learning algorithm parameters. The experiments are based on the designated training and development sets in the treebank distribution, with final evaluation on the separate test set (Hajič *et al.* 2001).

Although less than 2% of all arcs in the training data are non-projective, they are distributed over as many as 23% of the sentences. It follows that the configuration of

Table 8. *Attachment score (AS_U), precision (P), recall (R) and F measure for selected dependency types for Czech (held-out test set, etest section)*

Dependency Type	n	AS_U	P	R	F
Adverbial (Adv)	12948	88.0	75.3	74.2	74.7
Attribute (Atr)	36239	86.9	82.8	83.6	83.2
Subordinate conjunction (AuxC)	2055	75.9	80.5	75.8	78.1
Preposition (AuxP)	12658	72.0	73.7	71.7	72.4
Auxiliary Verb (AuxV)	1747	85.6	91.3	85.1	88.2
Rhematizer (AuxZ)	1962	76.9	70.0	73.9	71.9
Coordination node (Coord)	2716	31.4	39.0	31.0	34.5
Ellipsis handling (ExD)	2529	59.9	43.6	31.2	36.4
Object (Obj)	10480	81.6	66.5	62.6	64.5
Nominal predicate's nominal part (Pnom)	1668	80.2	63.8	70.3	66.9
Main predicate (Pred)	2892	58.2	45.7	53.1	49.1
Root node (ROOT)	7462	77.0	61.5	77.0	68.4
Subject (Sb)	9364	79.8	68.6	69.8	69.3
Total	108128	80.1	72.8	72.8	72.8

MaltParser used for all languages, constructing only projective graphs, cannot even in theory achieve an exact match for these sentences. To cope with non-projectivity, the concept of pseudo-projective parsing was introduced and evaluated in Nivre and Nilsson (2005). An overview of this approach is given in section 3.4.

Using non-projective training data, i.e., without applying any tree transformations and encodings, the overall accuracy scores are $AS_U = 78.5\%$ and $AS_L = 71.3\%$. By simply transforming all non-projective sentences to projective, without encoding the transformations in dependency type labels (baseline), an improvement is achieved for both $AS_U = 79.1\%$ and $AS_L = 72.0\%$. This indicates that it helps to make the input conform to the definition of projectivity, despite the fact that the trees are distorted and that it is not possible to recover non-projective arcs in the output of the parser.

In Nivre and Nilsson (2005), three types of encoding schemes were evaluated in order to recover the non-projective structure by an inverse transformation. The encodings increase the burden on the parser, since it now also has to distinguish between pseudo-projective arcs and the original projective arcs. The differences between different encodings are small and not statistically significant, but all three encodings increase both labeled and unlabeled attachment score in comparison both to the projectivized baseline and to the use of non-projective training data (all differences being significant beyond the 0.01 level according to McNemar's test). Compared to the projectivized baseline, the improvement is as high as 1 precentage point for $AS_U = 80.1\%$ and 0.8 percentage points for $AS_L = 72.8\%$.

A closer look at the 13 most frequent dependency types in table 8 reveals a larger drop from unlabeled to labeled accuracy compared to other languages such as English and Chinese. This is partly a result of the more fine-grained set of dependency types for Czech, but the more flexible word order for major clause constituents like Sb (subject) and Obj (object) is probably important as well. On the other hand, dependents of the types AuxC (subordinate conjunction), AuxP

(preposition), AuxV (auxiliary verb) or Coord (conjunction) actually have a higher F measure than AS_U, due to higher precision. In contrast to Sb and Obj, these dependents all come from closed word classes, which often uniquely identifies the dependency type. In addition, it is worth noting the surprisingly low accuracy for Coord, lower than for most other languages. This may indicate that the analysis of coordination in PDT, treating the coordinating conjunction as the head, does not interact well with the parsing strategy and/or feature models adopted in the experiments.[14]

We are not aware of any published results for labeled accuracy, but the unlabeled attachment score obtained is about 5% lower than the best results reported for a single parser, using the parser of Charniak (2000), adapted for Czech, with corrective post-processing to recover non-projective dependencies (Hall and Novák 2005).

4.2.6 Danish

The Danish experiments are based on the Danish Dependency Treebank (DDT), which is based on a subset of the Danish PAROLE corpus and annotated according to the theory of Discontinuous Grammar (Kromann 2003). This annotation involves *primary* dependencies, capturing grammatical functions, and *secondary* dependencies, capturing other relations such as co-reference. Our experiments only concern primary dependencies, since including secondary dependencies as well would have violated the SINGLE-HEAD constraint (cf. section 2.1), but the dependency type set is still the most fine-grained of all, with 54 distinct dependency types. The annotation is not restricted to projective dependency graphs, and while only about 1% of all dependencies are non-projective, the proportion of sentences that contain at least one non-projective dependency is as high as 15%.

The treebank has been divided into training, validation and test sets using the same pseudo-randomized splitting method described earlier for Swedish and Chinese. The training data for the experiments have been projectivized in the same way as the Czech data, with a similar improvement compared to the use of non-projective training data. However, none of the encoding schemes for recovering non-projective dependencies in the output of the parser led to any improvement in accuracy (nor to any degradation), which is probably due to the fact that the training data for non-projective dependencies are much more sparse than for Czech.

The best performing model for Danish is a modification of the standard model, where the feature $w(\tau_1)$ (the word form of the first lookahead token) is omitted, and the feature $w(h(\sigma_0))$ (the word form of the head of the top token) is replaced by the suffix feature $s_6(w(h(\sigma_0)))$. The TIMBL settings are standard. The overall accuracy scores for Danish are $AS_U = 85.6\%$ and $AS_L = 79.5\%$.[15] The relatively wide gap between unlabeled and labeled accuracy is probably due mainly to the fine-grained

[14] In more recent work, Nilsson *et al.* (2006) have shown how parsing accuracy for coordination in Czech can be improved by transforming the representations so that coordinating conjunctions are not treated as heads internally.

[15] The labeled attachment score is slightly lower than the one published in Nivre and Hall (2005), where results were based on the development test set.

Table 9. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Danish, $n \geq 10$ (held-out test set, section 0)*

Dependency Type	n	AS_U	P	R	F
Elliptic modifier (<MOD>)	11	45.5	0.0	0.0	–
Root (ROOT)	554	91.2	87.5	91.2	89.3
Adjectival object (AOBJ)	17	70.6	50.0	17.6	26.0
Parenthetical apposition (APPA)	20	50.0	53.8	35.0	42.4
Restrictive apposition (APPR)	23	43.5	69.2	39.1	50.0
Adverbial object (AVOBJ)	19	78.9	30.8	21.1	25.0
Conjunct (CONJ)	399	80.7	77.4	77.4	77.4
Coordinator (COORD)	299	75.6	75.4	74.9	75.1
Direct object (DOBJ)	504	90.1	77.5	77.8	77.6
Expletive subject (EXPL)	36	100.0	89.5	94.4	91.9
Indirect object (IOBJ)	13	100.0	66.7	15.4	25.0
List item (LIST)	17	29.4	57.1	23.5	33.3
Locative object (LOBJ)	117	88.0	53.0	45.3	48.8
Modifier (MOD)	1809	77.9	70.6	71.0	70.8
Parenthetical modifier (MODP)	15	26.7	0.0	0.0	–
Modifying proper name (NAME)	13	30.8	22.2	15.4	18.2
Modifying first name (NAMEF)	96	91.7	79.8	90.6	84.9
Nominal object (NOBJ)	1831	92.6	88.5	91.6	90.0
Verbal particle (PART)	21	85.7	62.5	23.8	34.5
Prepositional object (POBJ)	501	79.6	64.4	66.7	65.5
Possessed (POSSD)	171	90.1	91.3	85.4	87.1
Predicative (PRED)	251	86.5	62.0	65.7	63.8
Quotation object (QOBJ)	37	78.4	51.9	75.7	61.6
Relative clause modification (REL)	131	59.5	62.7	56.5	59.4
Subject (SUBJ)	892	93.6	90.7	90.7	90.7
Title of person (TITLE)	19	78.9	63.6	73.7	68.3
Temporal adjunct (TOBJ)	16	50.0	62.5	31.3	41.7
Verbal object (VOBJ)	635	95.1	92.7	93.4	93.0
Direct quotation (XPL)	12	0.25	0.0	0.0	–
Total	8530	85.6	79.5	79.5	79.5

nature of the dependency type set in combination with a relatively small training data set.

Table 9 shows the unlabeled attachment score (AS_U), precision (P), recall (R) and F measure (F) for dependency types occurring at least 10 times in the test set. It is clear that low-frequency types ($n < 100$) generally have very low labeled precision and recall, despite sometimes having a quite high unlabeled accuracy. A striking example is indirect object (IOBJ), which has perfect unlabeled accuracy but only 15% labeled recall. Concentrating on types that occur at least 100 times in the test set, we see a pattern that is very similar to the one observed for the closely related language Swedish, despite important differences in the style of annotation. Thus, we can observe a very high accuracy ($F \geq 90$) for dependencies involving function words, notably VOBJ, which includes dependencies linking verbs to function words (auxiliary verb → main verb, marker → infinitive, complementizer → verb), and NOBJ, which includes dependencies linking prepositions and determiners to nominals, but also for subjects, both normal subjects (SUBJ) and the much less frequent expletive subjects (EXPL), and roots (ROOT). Furthermore, we see that other arguments of the verb (DOBJ, IOBJ, LOBJ, PRED) have a high unlabeled accuracy but (sometimes substantially) lower labeled accuracy, while the generic

adjunct type MOD has lower accuracy, both labeled and unlabeled. Finally, both Danish and Swedish have comparatively high accuracy for coordination, which in Danish is split into CC (conjunct → coordinator) and COORD (conjunct$_i$ → conjunct$_{i+1}$). Compared to the results for Czech, this indicates that an analysis of coordination where a conjunct, rather than the coordinator, is treated as the head is easier to cope with for the parser.

McDonald and Pereira (2006) report an unlabeled attachment score for primary dependency types in DDT of 86.8%.[16] However, these results are based on gold standard part-of-speech tags, whereas our experiments use automatically assigned tags with an accuracy rate of 96.3%. Replicating the experiment with gold standard tags, using the same feature model and parameter settings, results in an unlabeled attachment score of 87.3%, which indicates that MaltParser gives state-of-the-art performance for Danish.

4.2.7 Dutch

The Dutch experiments are based on the Alpino Treebank (Beek *et al.* 2003). The text material (186k non-punctuation tokens) consists primarily of two sections of newspaper text (125k and 21k), plus two smaller segments containing questions (21k) and (in part) manually constructed sentences for parser development and annotation guide examples (19k). As the latter type of material is atypical, it is only used for training purposes, whereas the smaller newspaper text section is used as held out material for final testing.

The syntactic annotation of the Alpino Treebank is a mix of constituent structure and dependency relations, nearly identical to the syntactic annotation of the Spoken Dutch Corpus (Wouden *et al.* 2002). It was converted to a pure dependency structure employing a head percolation table, removing secondary relations as indicated by traces. Multi-word units, consisting of a sequence of words without any further syntactic analysis, were concatenated into a single word using underscores. Finally, non-projective structures were projectivized using the same baseline procedure as for Danish (i.e., without extending the dependency type labels or attempting to recover non-projective dependencies in the output of the parser). Since the original part-of-speech tags in the Alpino Treebank are coarse-grained and lack any additional feature information besides the word class, all tokens were retagged with the memory-based tagger for Dutch (Daelemans *et al.* 2003).

Ten-fold cross-validation was used to manually optimize the TiMBL settings. Experimentation confirmed that the standard settings with MVDM, no feature weighting, and distance weighted class voting generally performs best. However, choosing a higher value for k ($k = 10$) usually gives an improvement of one to two percentage points for Dutch. The results obtained on held out data using the standard model are $AS_U = 84.7\%$ and $AS_L = 79.2\%$. The relatively large

[16] It should be pointed out that McDonald and Pereira (2006) also consider secondary dependency arcs, which are beyond the reach of MaltParser in its current configuration, and that the result reported is actually the highest precision of their parser when restricted to primary dependencies.

Table 10. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Dutch (held-out test set)*

Dependency Type	*n*	AS_U	P	R	F
Apposition (APP)	299	73.6	78.8	71.9	75.2
Body of embedded clause (BODY)	88	85.8	83.9	84.3	84.1
Conjunct (CNJ)	997	70.0	72.8	68.6	70.6
Coordinator (CRD)	9	44.4	28.6	22.2	25.0
Determiner (DET)	3239	97.2	96.1	96.9	97.0
Closing element of circumposition (HDF)	13	53.8	70.0	53.8	60.8
Locative/directional complement (LD)	239	68.6	40.2	21.3	27.9
Measure complement (ME)	33	72.7	69.2	54.5	61.0
Modifier (MOD)	5069	78.3	71.1	73.9	72.5
Object of adjective or adverb (OBCOMP)	51	74.5	90.0	52.9	66.6
Direct object (OBJ1)	3392	90.3	86.0	86.4	86.2
Indirect object (OBJ2)	56	80.4	77.8	12.5	21.5
Prepositional complement (PC)	344	73.8	51.6	28.5	36.7
Suppletive object (POBJ1)	14	78.6	33.3	35.7	34.5
Predicative complement (PREDC)	428	79.4	65.6	56.1	60.5
Predicate modifier (PREDM)	61	65.6	54.5	9.8	16.6
Root (ROOT)	1874	82.7	70.8	82.7	76.3
Obligatory reflexive object (SE)	53	83.0	72.2	73.6	72.9
Subject (SU)	186	85.2	80.8	78.1	79.4
Suppletive subject (SUP)	19	89.5	45.0	47.4	46.2
Separable verbal particle (SVP)	259	85.3	69.6	61.8	65.5
Verbal complement (VC)	1074	89.0	80.4	85.6	82.9
Total	20263	84.7	79.2	79.2	79.2

gap between the labeled and unlabeled scores may be attributed to the relatively fine-grained set of dependency labels. Table 10 gives unlabeled attachment score (AS_U), labeled precision (P), recall (R) and F measure (F) for individual dependency types. We can observe a general trend towards better scores for the more frequent dependency labels, but there are notable exceptions such as the relatively high score for the infrequently occurring SE (reflexive object) and the low score on the more frequent PC (prepositional complement) and LD (locative/directional complement).

As for several other languages, we can distinguish three groups with high, medium and low F measures respectively. The high score set ($F > 80\%$) includes the dependency relations indicated by closed class words: DET (determiner → noun), VC (auxiliary verb → main verb), and BODY (complementizer → verb). Somewhat surprisingly, this group also includes OBJ1 (direct object), perhaps because this is the second most frequent dependency relation.

The low score group ($F < 60\%$) includes the rather infrequent suppletive subject (SUP) and object (POBJ1). Furthermore, it involves four classes which are canonically expressed in the form of a prepositional phrase – PC (prepositional complement), OBJ2 (indirect object), LD (locative/directional complement) and PREDM (predicate modifier) – and where the sometimes subtle distinction is often of a semantic rather than a syntactic nature. The fact that coordinator (CRD) is also in the low score group is somewhat counter-intuitive, because it is indicated by a closed word class, normally the word *en* 'and', but the result is consistent with

the low accuracy for coordination in Czech, given that both treebanks treat the coordinating conjunction as the head of a coordinate structure.

The remaining 11 types belong to the medium score group ($60\% < F < 80\%$), which includes the by far most frequent class MOD (modifier). It is interesting to note that the scores for a conceptually difficult class like APP (apposition) are still quite good. The same goes for the potentially highly ambiguous CONJ (conjunct), although there seems to be a trade-off here with the low scores noted for CRD earlier.

The Alpino parser is a rule-based, HPSG-style parser that is currently the state-of-art parser for Dutch (Bouma *et al.* 2001). It has an extensive and detailed lexicon (including, e.g., subcategorization information) and a MaxEnt-based disambiguation module. Its output is in the same format as the Alpino Treebank. We used it to parse the held out material and converted the parse trees to dependency structures, using exactly the same procedure as for converting the treebank, which includes transforming non-projective to projective structures. Evaluation resulted in the scores $AS_U = 93.2\%$ and $AS_L = 91.2\%$. Clearly, there is still a substantial gap between the two parsers. Also, the Alpino parser provides additional information, e.g., traces and non-projective analyses, which is ignored here. Yet, given all the effort invested in building the Alpino grammar, lexicon, and disambiguation strategy, it is interesting to see that its performance can be approximated by a purely inductive approach using fairly limited amounts of data.

4.2.8 German

The experiments for German are based on the Tübingen Treebank of Written German (TüBa-D/Z) (Telljohann *et al.* 2005). The treebank is based on issues of the German daily newspaper 'die tageszeitung' (taz) that appeared in April and May of 1999. The annotation of the treebank is constituency-based, but it is augmented by function-argument structure on all levels, which allows a straightforward conversion to dependencies for most phenomena. Heuristics are used only for apposition, embedded infinitive clauses, and nominal postmodifications. Long-distance relations, which are annotated in the constituency model via special labels, are translated into non-projective dependencies. The set of dependency types is modeled after the one used for the Constraint Dependency Grammar for German (Foth *et al.* 2004), a manually written dependency grammar for German.

The best performing model for German modifies the standard model by omitting the two lexical features $w(h(\sigma_0))$ and $w(\tau_1)$ and by adding the part-of-speech of an additional stack token $p(\sigma_2)$. The TiMBL settings for German deviate from the standard settings by using $k = 13$ and voting based on inverse linear weighting (IL).

The overall accuracy scores for German are $AS_U = 88.1\%$ and $AS_L = 83.4\%$. The (unlabeled) results are comparable to results by Foth *et al.* (2004), who reached 89.0% accuracy when parsing the NEGRA treebank (Skut *et al.* 1997), another treebank for German, which is also based on newspaper texts (but which uses a different constituency-based annotation scheme). The labeled results are considerably

Table 11. *Attachment score (AS_U), precision (P), recall (R) and F measure for selected dependency types for German (mean of 10-fold cross-validation, frequency counts rounded to whole integers)*

Dependency Type	n	AS_U	P	R	F
Adverbial (ADV)	2762	80.5	78.7	79.3	79.0
Determiner (DET)	4485	99.1	99.1	99.0	99.0
Genitive modifier (GenMOD)	571	79.5	59.1	66.3	62.5
Accusative Object (AccOBJ)	1466	82.4	66.6	73.4	69.8
Dative Object (DatOBJ)	219	79.0	62.4	16.4	26.0
Genitive Object (GenOBJ)	4	78.0	16.7	6.8	9.7
Predicate (PRED)	549	84.8	69.6	64.3	66.8
PP complement (PPOBJ)	399	83.1	54.7	41.6	47.3
Relative clause (RelCL)	241	54.1	56.9	52.6	54.7
Subject (SUBJ)	2931	92.0	85.7	86.3	86.0
Total	32555	88.1	83.4	83.4	83.4

higher than constituency parsing results reported for German, which reach a labeled F measure of 75.3% when constituent nodes also include grammatical functions (Kübler *et al.* 2006).

Table 11 gives unlabeled attachment scores (AS_U), labeled precision (P), recall (R), and F measure (F) for selected dependency types. The overall trends are very similar to what we have observed for other languages, notably Germanic languages like Swedish and Danish. For example, both determiners (DET) and adverbials (ADV) have labeled and unlabeled accuracy at about the same level (although considerably higher for DET than for ADV), while arguments of the verb (AccOBJ, DatOBJ, GenOBJ, PRED and PPOBJ) have substantially better unlabeled than labeled accuracy. One difference, compared to Danish and Swedish, is that the lower labeled accuracy also affects subjects (SUBJ), which is probably a reflection of the fact that German exhibits freer word order thanks to case marking. The relatively low labeled accuracy for different case-marked arguments is also an indication that the parser would benefit from morphological information, which is currently not included in the German part-of-speech tags.

Contrary to expectations that, with growing data size, adding more lexical features would improve performance, experiments with all the lexical features of the standard model showed a decrease in performance by 1.5 percentage points. The hypothesis that this decrease is due to data sparseness is refuted by experiments with only 2000 sentences for training, where the decrease in performance is only 7.5%. These results are consistent with those of Dubey and Keller (2003), who found that lexicalizing a PCFG grammar for NEGRA results in a decrease in performance, although it should be remembered that the first two lexical features are beneficial in the case of MaltParser.

4.2.9 Italian

The Italian treebank used in the experiments is the Turin University Treebank (TUT) (Bosco 2004), consisting of 1500 sentences and 41771 tokens. It is balanced

Table 12. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Italian (mean of 10-fold cross-validation, frequency counts rounded to whole integers)*

Dependency Type	*n*	AS_U	P	R	F
Apposition (APPOSITION)	69	44.4	54.2	47.8	50.8
Argument (ARG)	1351	95.0	92.7	94.5	93.6
Auxiliary verb (AUX)	96	92.1	90.5	94.2	92.3
Part of expression (CONTIN)	75	86.9	78.2	54.4	64.2
Coordination (COORDINATOR)	271	66.6	63.6	63.6	63.6
Other (DEPENDENT)	1	40.0	0.0	0.0	–
Reflexive complement (EMPTYCOMPL)	15	94.5	35.7	50.0	41.7
Indirect complement (INDCOMPL)	82	85.9	70.4	47.5	56.7
Indirect object (INDOBJ)	18	81.5	33.3	33.3	33.3
Interjection (INTERJECTION)	1	20.0	0.0	0.0	–
Object (OBJ)	222	84.9	33.3	33.3	33.3
Predicative complement (PREDCOMPL)	52	78.4	54.3	37.3	44.2
Restrictive modifier (RMOD)	1013	74.3	69.5	70.2	69.8
Subject (SUBJ)	256	75.5	64.8	58.6	61.5
Root (TOP)	150	75.5	63.5	77.2	69.7
Adverbial extraction (VISITOR)	13	74.6	0.0	0.0	–
Total	3683	82.9	75.7	75.7	75.7

over genres with 60% newspaper text, 30% legal text, and 10% from novels and academic literature. The dependency annotation involves traces in order to avoid non-projective structures, although there is in fact a certain number of non-projective trees in the treebank.

The treebank has been converted to the format required by MaltParser without significant loss of linguistic information, as described in Chanev (2005), replacing traces if necessary by (possibly non-projective) dependency arcs. The dependency tag set was reduced from 283 to 17 distinct tags, keeping only information about syntactic dependency relations. The training data were projectivized using the same procedure as for Danish and Dutch and tagged for part-of-speech using TnT (Brants 2000). All experiments were performed using 10-fold cross-validation with a randomized split.

The best performing feature model for Italian is the standard model, although several simpler models give nearly the same results. The accuracy scores for Italian are $AS_U = 82.9\%$ and $AS_L = 75.7\%$, and table 12 shows the accuracy obtained for different dependency types. It is striking that there are only two types that obtain a really high accuracy in the Italian data, the type ARG, which is usually used for relations between articles and nouns or prepositions and articles, and the type AUX, which is used for auxiliary verbs. While these two types have a labeled F measure well above 90%, no other type has a score higher than 70%. There is also a set of low-frequency types that all have zero recall and precision. The relatively low labeled accuracy for most dependency types in Italian is undoubtedly due partly to sparse data, but it is also relevant that the inventory of dependency types is more semantically oriented than for most other languages.

For Italian there are not any published results for statistical dependency parsing except the preliminary results for MaltParser reported in Chanev (2005). Compared

to Corazza *et al.* (2004), where state-of-the-art constituency parsers were tested on the Italian Syntactic-Semantic Treebank (Montemagni *et al.* 2003), an improvement seems to have been achieved, although it is not straightforward to compare evaluation metrics for constituency and dependency parsing. A more relevant comparison is the rule-based parser of Lesmo *et al.* (2002), which uses the TUT dependency type set and which has been reported to achieve a labeled attachment score of 76.65% when evaluated during the development of the treebank. Since this is within a percentage point of the results reported in this article and the evaluation is based on the same kind of data, it seems clear that MaltParser achieves highly competitive results for Italian.

4.2.10 Turkish

The Turkish Treebank (Oflazer *et al.* 2003), created by Metu and Sabancı Universities is used in the experiments for Turkish. This treebank is composed of 5635 sentences, annotated with dependency structures, of which 7.2% are non-projective (not counting punctuation that is not connected to a head). As can be seen from table 1, even though the number of sentences in the Turkish Treebank is in the same range as for Danish, Swedish and Bulgarian, the number of words is considerably smaller (54k as opposed to 70–100k for the other treebanks). This significant difference arises from the very rich morphological structure of the language due to which a word may sometimes correspond to a whole sentence in another language.

As a result of their agglutinative morphology, Turkish words can change their main part-of-speech after the concatenation of multiple suffixes. This structure is represented in the treebank by dividing words into inflectional groups (IG). The root and derived forms of a word are represented by different IGs separated from each other by derivational boundaries (DB). Each IG is annotated with its own part-of-speech and inflectional features, as illustrated in the following example:[17]

okulunuzdaydı
(he was at your school)

okulunuzda	DB	ydı
okul+Noun+A3sg+P2pl+Loc	DB	+Verb+Zero+Past+A3sg
IG_1		IG_2

The part-of-speech of the stem of the word *okulunuzdaydı* is a noun, from which a verb is derived in a separate IG. In the treebank, dependencies hold between specific IGs of the dependent and head word.

For the parsing experiments, we have concatenated IGs into word forms to get a word-based tokenization and used a reduced version of the part-of-speech tagset given by the treebank, very similar to the reduced tagset used in the parser of Eryiğit and Oflazer (2006). For each word, we use the part-of-speech of each IG and in addition include the case and possessive information if the stem is a noun or pronoun. Using this approach, the tag of the word *okulunuzdaydı* becomes

[17] A3sg = 3sg number agreement, P2pl = 2pl possessive agreement, Loc = locative case.

Table 13. *Attachment score (AS_U), precision (P), recall (R) and F measure per dependency type for Turkish (mean of 10-fold cross-validation, frequency counts rounded to whole integers)*

Dependency Type	*n*	AS_U	P	R	F
ABLATIVE.ADJUNCT	52	82.8	58.8	54.7	56.7
APPOSITION	190	40.6	8.5	5.9	7.0
CLASSIFIER	205	87.0	72.8	70.4	71.6
COLLOCATION	5	41.2	25.0	5.9	9.5
COORDINATION	81	53.6	56.0	48.6	52.0
DATIVE.ADJUNCT	136	86.8	55.0	54.9	54.9
DETERMINER	195	91.1	83.7	85.8	84.7
EQU.ADJUNCT	2	62.5	0.0	0.0	–
ETOL	1	70.0	0.0	0.0	–
FOCUS.PARTICLE	2	78.3	0.0	0.0	–
INSTRUMENTAL.ADJUNCT	27	71.6	34.7	18.8	24.4
INTENSIFIER	90	93.9	82.9	86.0	84.4
LOCATIVE.ADJUNCT	114	73.0	59.5	58.1	58.8
MODIFIER	1168	76.5	68.7	68.2	68.4
NEGATIVE.PARTICLE	16	90.0	89.6	80.6	84.9
OBJECT	796	88.3	68.6	69.4	69.0
POSSESSOR	152	80.0	81.7	69.9	75.3
QUESTION.PARTICLE	29	93.8	85.9	80.2	83.0
RELATIVIZER	8	91.8	54.5	49.4	51.8
ROOT	2	0.0	0.0	0.0	–
SENTENCE.MODIFIER	59	52.4	33.8	47.6	39.5
SENTENCE	725	91.2	84.4	89.2	86.7
SUBJECT	448	72.0	50.7	50.8	50.7
VOCATIVE	24	51.0	20.4	19.1	19.7
Total	4357	81.6	69.0	69.0	69.0

Noun+P2pl+Loc+Verb. Even after this reduction, the tagset contains 484 distinct tags, making it by far the biggest tagset used in the experiments.

The best performing model for Turkish omits five of the features of the standard model, three part-of-speech features ($p(\sigma_1)$, $p(\tau_2)$, $p(\tau_3)$) and two lexical features ($w(h(\sigma_0))$, $w(\tau_1)$). In addition, the stem of a word is used as its word form in lexical features. This leads to an accuracy of $AS_U = 81.6\%$ and $AS_L = 69.0\%$. These are the mean results obtained after 10-fold cross-validation.

Table 13 gives unlabeled attachment scores (AS_U), labeled precision (P), recall (R), and F measure (F) for individual dependency types. First of all, we see that types with a frequency below 5 in the test set have very low labeled accuracy, which is consistent with results reported for other languages earlier. Secondly, we may note that the frequency of tokens analyzed as roots (ROOT) is very low, which is a consequence of the fact that punctuation tokens are excluded in evaluation, since final punctuation is generally treated as the root node of a sentence in the Turkish Treebank.[18] Therefore, the closest correspondent to ROOT for other languages is SENTENCE, which is the type assigned to a token dependent on the final punctuation token (normally the final verb of the sentence) and which has a very

[18] The few roots that do occur are unconnected words that give rise to non-projective dependency structures.

high accuracy, on a par with the ROOT type for most other languages. Finally, there is a clear tendency that dependency types with high accuracy (INTENSIFIER, QUESTION.PARTICLE, RELATIVIZER, SENTENCE, DETERMINER, NEGATIVE.PARTICLE) are types that are generally adjacent to their head, whereas types with lower accuracy (COORDINATION, SENTENCE.MODIFIER, APPOSITION, COLLOCATION, VOCATIVE) are types that are either more distant or hard to differentiate from other types.

The only comparable results for Turkish are for the unlexicalized dependency parser of Eryiğit and Oflazer (2006). These results are based on a selected subset of the treebank sentences containing only projective dependencies with the heads residing on the right side of the dependents and the main evaluation metrics are based on IGs rather than words, but word-based scores are presented for the purpose of comparison with a top score of $AS_U = 81.2\%$. Applying MaltParser with the best feature model to the same subset of the treebank resulted in an unlabeled attachment score of 84.0%, which is a substantial improvement.[19]

4.3 Discussion

Although MaltParser achieves an unlabeled dependency accuracy above 80% for all languages, there is also a considerable range of variation, which seems to correlate fairly well with the linguistic dimensions of morphological richness and word order flexibility, exemplified by high accuracy for English and lower accuracy for Czech, which represent extreme positions on these scales. Given that English and Czech are also the languages with the largest data sets, the linguistic properties seem to be more important than the amount of data available. Another influencing factor is the level of detail of the dependency annotation, as given by the number of dependency types used, where Czech has a more fine-grained classification than English. However, Danish has an even more fine-grained classification but still comes out with higher parsing accuracy than Czech, despite a much smaller training data set.

If morphological richness and word order flexibility are indeed the most important factors determining parsing accuracy, the results for German are surprisingly good, given that German has both richer morphology and freer word order than English. On the other hand, the results for Chinese are on the low side. This points to another important factor, namely the complexity of the sentences included in the treebank data, which can be roughly approximated by considering the mean sentence length in the sample. Here we see that Chinese has the second highest value of all languages, while the sentence length for German is at least considerably lower than for English. At the same time, we have to remember that the number of words per sentence is not strictly comparable between languages with different morphological properties, as illustrated especially by the data for Turkish (cf. section 4.2.10).

[19] Strictly speaking, the subset used by Eryiğit and Oflazer (2006) only contains non-crossing dependencies, although it does contain punctuation that is not connected to other tokens. In order to make these graphs projective, the punctuation tokens were attached to the immediately following word. However, since punctuation is excluded in all evaluation scores, this nevertheless seems like a fair comparison.

Comparing individual dependency types across languages is very difficult, given the diversity in annotation, but a few recurrent patterns are clearly discernible. The first is that dependencies involving function words generally have the highest accuracy. The second is that core arguments of the verb often have high unlabeled accuracy but lower labeled accuracy, with the possible exception of subjects, which have high labeled accuracy in languages where they are distinguished configurationally. The third is that the parsing accuracy for coordinate structures tends to be higher if the dependency analysis treats conjuncts, rather than coordinating conjunctions, as heads.

Needless to say, a more detailed error analysis will be needed before we can draw any reliable conclusions about the influence of different factors, so the tentative conclusions advanced here are best regarded as conjectures to be corroborated or refuted by future research. However, given the fact that unlabeled dependency accuracy is consistently above 80%, the parsing methodology has proven to be relatively insensitive to differences in language typology as well as in annotation schemes. Moreover, respectable results can be obtained also with fairly limited amounts of data, as illustrated in particular by the results for Italian and Turkish.

Finally, we note that MaltParser achieves state-of-the-art performance for most of the languages investigated in this article, although the possibility of comparison differs widely between languages. For English, Chinese, Czech and Dutch, parsing accuracy does not quite reach the highest level, but the difference is never more than about 5% (slightly more for Dutch).[20]

5 Conclusion

We have presented MaltParser, a data-driven system for dependency parsing that can be used to construct syntactic parsers for research purposes or for practical language technology applications. Experimental evaluation using data from ten different languages shows that MaltParser generally achieves good parsing accuracy without language-specific enhancements and with fairly limited amounts of training data. Unlabeled dependency accuracy is consistently above 80% and the best results are normally within a 5% margin from the best performing parsers, where such comparisons are possible. MaltParser is freely available for research and educational purposes.

Acknowledgments

We want to express our gratitude for assistance with data sets, conversions and many other things to Christina Bosco, Yuchang Cheng, Yuan Ding, Jan Hajič,

[20] More recent work using SVM, rather than MBL, for discriminative learning has shown that this gap can be closed, and in the recent shared task of multilingual dependency parsing at the Tenth Conference on Computational Natural Language Learning (CoNLL-X), MaltParser was one of the two top performing systems (Buchholz and Marsi 2006; Nivre *et al.* 2006; Hall 2006).

Matthias Trautner Kromann, Alberto Lavelli, Haitao Liu, Yuji Matsumoto, Ryan McDonald, Kemal Oflazer, Petya Osenova, Kiril Simov, Yannick Versley, Hiroyasu Yamada, and Daniel Zeman. We are also grateful for the support of GSLT (Swedish National Graduate School of Language Technology), TÜBİTAK (The Scientific and Technical Research Council of Turkey), and The Swedish Research Council. Finally, we want to thank our three anonymous reviewers for insightful comments and suggestions that helped us improve the final version of the article.

References

Van der Beek, L., Bouma, G., Malouf, R. and Van Noord, G. 2003. The Alpino Dependency Treebank. In Gaustad, T. (ed.) *Computational Linguistics in the Netherlands 2002. Selected Papers from the Thirteenth CLIN Meeting*, pp. 8–22. Rodopi.

Berwick, R. C. 1985. *The Acquisition of Syntactic Knowledge*. MIT Press.

Bikel, D. and Chiang, D. 2000. Two statistical parsing models applied to the Chinese Treebank. In *Proceedings of the Second Chinese Language Processing Workshop*, pp. 1–6.

Black, E. and Garside, R. and Leech, G. (eds.) 1993. *Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach*. Rodopi.

Black, E., Jelinek, F., Lafferty, J., Magerman, D., Mercer, R. and Roukos, S. 1992. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pp. 31–37.

Blaheta, D. and Charniak, E. 2000. Assigning function tags to parsed text. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 234–240.

Böhmová, A., Hajič, J., Hajičová, E. and Hladká, B. 2003. The Prague Dependency Treebank: A three-level annotation scenario. In Abeillé, A. (ed.), *Treebanks: Building and Using Parsed Corpora*, pp. 103–127. Dordrecht: Kluwer.

Bosco, C. 2004. *A Grammatical Relation System for Treebank Annotation*. PhD thesis, Turin University.

Bouma, G., Van Noord, G. and Malouf, R. 2001. Alpino: Wide-coverage computational analysis of Dutch. In Daelemans, W., Sima'an, K., Veenstra, J. and Zavrel, J. (eds.) *Computational Linguistics in the Netherlands 2000. Selected Papers from the Eleventh CLIN Meeting*, pp. 45-59. Rodopi.

Brants, T. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP'2000)*, pp. 224–231.

Buchholz, S. and Marsi, E. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pp. 149–164.

Chanev, A. 2005. Portability of dependency parsing algorithms – an application for Italian. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 29–40.

Chang, C.-C. and Lin, C.-J. 2001. LIBSVM: A library for support vector machines. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Charniak, E. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 132–139.

Charniak, E. and Johnson, M. 2005. Coarse-to-fine *n*-best parsing and discriminative MaxEnt reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 173–180.

Cheng, Y., Asahara, M. and Matsumoto, Y. 2004. Deterministic dependency structure analyzer for Chinese. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP)*, pp. 500–508.

Cheng, Y., Asahara, M. and Matsumoto, Y. 2004. Machine learning-based dependency analyzer for Chinese. In *Proceedings of International Conference on Chinese Computing (ICCC)*, pp. 66–73.

Cheng, Y., Asahara, M. and Matsumoto, Y. 2005. Chinese deterministic dependency analyzer: Examining effects of global features and root node finder. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, pp. 17–24.

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 16–23.

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

Collins, M. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 175–182.

Collins, M. and Duffy, N. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 263–270.

Collins, M., Hajič, J., Ramshaw, L. and Tillmann, C. 1999. A statistical parser for Czech. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 505–512.

Collins, M. and Duffy, N. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics* 31(1), 25–70.

Corazza, A., Lavelli, A., Satta, G. and Zanoli, R. 2004. Analyzing an Italian treebank with state-of-the-art statistical parsers. In *Proceedings of the Third Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 39–50.

Covington, M. A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102.

Daelemans, W. and Van den Bosch, A. 2005. *Memory-Based Language Processing*. Cambridge University Press.

Daelemans, W., Zavrel, J., Van den Bosch, A. and Van der Sloot, K. 2003. MBT: Memory Based Tagger, version 2.0, Reference Guide. ILK Technical Report 03-13, Tilburg University.

Dubey, A. and Keller, F. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 96–103.

Einarsson, J. 1976. Talbankens skriftspråkskonkordans. Lund University, Department of Scandinavian Languages.

Eryiğit, G. and Oflazer, K. 2006. Statistical dependency parsing of Turkish. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 89–96.

Foth, K., Daum, M. and Menzel, W. 2004. A broad-coverage parser for German based on defeasible constraints. In *KONVENS 2004, Beiträge zur 7. Konferenz zur Verarbeitung natürlicher Sprache*, pp. 45–52.

Hajič, J., Vidova Hladka, B., Panevová, J., Hajičová, E., Sgall, P. and Pajas, P. 2001. Prague Dependency Treebank 1.0. LDC, 2001T10.

Hall, J. 2006. *MaltParser – An Architecture for Labeled Inductive Dependency Parsing*. Licentitate thesis, Växjö University.

Hall, K. and Novák, V. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 42–52.

Hudson, R. A. 1990. *English Word Grammar*. Blackwell.

Johnson, M., Geman, S., Canon, S., Chi, Z. and Riezler, S. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 535–541.

Kay, M. 2000. Guides and oracles for linear-time parsing. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT)*, pp. 6–9.

Kromann, M. T. 2003. The Danish Dependency Treebank and the DTAG treebank tool. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 217–220. Växjö University Press.

Kübler, S., Hinrichs, E. W. and Maier, W. 2006. Is it really that difficult to parse German? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 111–119.

Kudo, T. and Matsumoto, Y. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pp. 63–69.

Lesmo, L., Lombardo, V. and Bosco, C. 2002. Treebank development: The TUT approach. In Sangal, R. and Bendre, S. M. (eds.) *Recent Advances in Natural Language Processing*, pp. 61–70. New Delhi: Vikas Publishing House.

Levy, R. and Manning, C. 2003. Is it harder to parse Chinese, or the Chinese Treebank? In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 439–446.

Lin, D. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering* 4, 97–114.

Magerman, D. M. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 276–283.

Marinov, S. and Nivre, J. 2005. A data-driven parser for Bulgarian. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 89–100.

Maruyama, H. 1990. Structural disambiguation with constraint propagation. In *Proceedings of the 28th Meeting of the Association for Computational Linguistics (ACL)*, pp. 31–38.

McDonald, R. and Pereira, F. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 81–88.

Mel'čuk, I. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Montemagni, S., Barsotti, F., Battista, M., Calzolari, N., Corazzari, O., Lenci, A., Zampolli, A., Fanciulli, F., Massetani, M., Raffaelli, R., Basili, R., Pazienza, M. T., Saracino, D., Zanzotto, F., Pianesi, F., Mana, N. and Delmonte, R. 2003. Building the Italian syntactic-semantic treebank. In Anne Abeillé (ed.) *Building and Using Syntactically Annotated Corpora*, pp. 189–210. Dordrecht: Kluwer.

Nilsson, J., Nivre, J. and Hall, J. 2006. Graph transformations in data-driven dependency parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pp. 257–264.

Nivre, J. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 149–160.

Nivre, J. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pp. 50–57.

Nivre, J. 2006. *Inductive Dependency Parsing*. Springer.

Nivre, J. and Hall, J. 2005. MaltParser: A language-independent system for data-driven dependency parsing. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 137–148.

Nivre, J., Hall, J. and Nilsson, J. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pp. 49–56.

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G. and Marinov, S. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pp. 221–225.

Nivre, J. and Nilsson, J. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99–106.

Nivre, J. and Scholz, M. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pp. 64–70.

Oflazer, K., Say, B., Hakkani-Tür, D. Z. and Tür, G. 2003. Building a Turkish treebank. In Abeillé, A. (ed.) *Treebanks: Building and Using Parsed Corpora*, pp. 261–277. Dordrecht: Kluwer.

Ratnaparkhi, A. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1–10.

Sagae, K. and Lavie, A. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 125–132.

Simov, K., Popova, G. and Osenova, P. 2002. HPSG-based syntactic treebank of Bulgarian (BulTreeBank). In Wilson, A., Rayson, P. and McEnery, T. (eds), *A Rainbow of Corpora: Corpus Linguistics and the Languages of the World*, pp. 135–142. Lincon-Europa.

Simmons, R. F. and Yu, Y.-H. 1992. The acquisition and use of context-dependent grammars for English. *Computational Linguistics* 18, 391–418.

Skut, W., Krenn, B., Brants, T. and Uszkoreit, H. 1997. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, Washington, D.C.

Tanev, H. and Mitkov, R. 2002. Shallow language processing architecture for Bulgarian. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING)*, pp. 995–1001.

Teleman, U. 1974. *Manual för grammatisk beskrivning av talad och skriven svenska*. Lund: Studentlitteratur.

Telljohann, H., Hinrichs, E. W., Kübler, S. and Zinsmeister, H. 2005. Stylebook for the Tübingen Treebank of Written German (TüBa-D/Z). Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany.

Titov, I. and Henderson, J. 2006. Porting statistical parsers with data-defined kernels. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pp. 6–13.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. Springer.

Veenstra, J. and Daelemans, W. 2000. A memory-based alternative for connectionist shift-reduce parsing. Technical Report ILK-0012, University of Tilburg.

Voutilainen, A. 2001. Parsing Swedish. Extended Abstract for the 13th Nordic Conference of Computational Linguistics, Uppsala University, May, 20-22, 2001.

Van der Wouden, T., Hoekstra, H., Moortgat, M., Renmans, B. and Schuurman, I. 2002. Syntactic analysis in the spoken Dutch corpus. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pp. 768–773.

Xue, N., Fei Xia, F.-D. and Palmer, M. 2005. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering* 11(2), 207–238.

Yamada, H. and Matsumoto, Y. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 195–206.

Zelle, J. M. and Mooney, R. J. 1993. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference of the American Association for Artificial Intelligence (AAAI)*, pp. 817–899.

Global Inference and Learning Algorithms for Multi-Lingual Dependency Parsing

Ryan McDonald*
University of Pennsylvania
Google Research

Fernando Pereira**
University of Pennsylvania

Koby Crammer**
University of Pennsylvania

Kevin Lerman**
University of Pennsylvania

This paper gives an overview of the work of McDonald et al. (McDonald et al. 2005a, 2005b; McDonald and Pereira 2006; McDonald et al. 2006) on global inference and learning algorithms for data-driven dependency parsing. Further details can be found in the thesis of McDonald (McDonald 2006). This paper is primarily intended for the audience of the ESSLLI 2007 course on data-driven dependency parsing.

1. Introduction

In this work, we study both learning and inference algorithms for producing dependency graph representations of natural language syntactic structure. Dependency graphs represent words and their relationship to syntactic modifiers through directed edges. For example, Figure 1 shows a dependency graph for the sentence, *John hit the ball with the bat*.

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 (Tesnière 1959) which was followed by a number of studies on dependency representations and their relationships to other formalisms, most notably by Hays (Hays 1964) and Gaifman (Gaifman 1965). 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 (Hudson 1984) and Mel′čuk's Meaning Text Theory (Mel′čuk 1988). Since then, a variety of computational syntactic dependency formalisms have been proposed. Notable amongst them is the work on constraint based dependency parsing (Maruyama 1990), which treats the parsing of dependency graphs as a constraint satisfaction problem. This framework has been extended theoretically (Maruyama 1990; Harper and Helzerman 1995) as well as applied in practical evaluations (Foth et al. 2000; Wang and Harper 2004), providing some of the best empirical support for any grammar-based dependency formalism. Another important framework is Functional Generative Description (Sgall et al. 1986), which provides the core theoretical foundation for the Prague Dependency Treebank (Böhmová et al. 2003) – the largest dependency treebank currently in use. Work on context-sensitive formalisms such as those in the TAG family (Joshi 1985) or CCGs (Steedman 2000) can also be viewed as producing dependency graphs

* 76 Ninth Ave., New York, NY 10011. Email: ryanmcd@google.com
** Department of Computer and Information Science, Levine Hall, 3330 Walnut Street, Philadelphia, PA 19104. Email: {pereira,crammer,klerman}@cis.upenn.edu

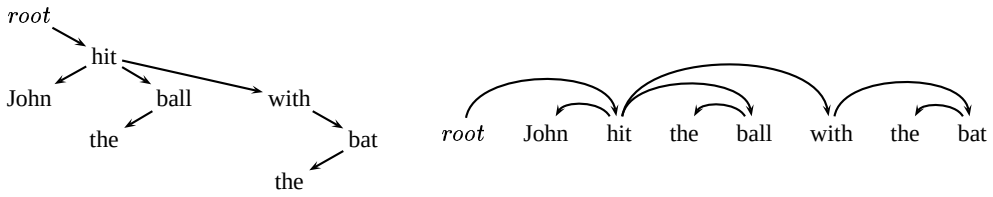


Figure 1
An example dependency graph.

Figure 2
A non-projective dependency graph.

of sentences through their derivation trees. However, these trees typically represent semantic dependencies, not syntactic ones.

The example dependency graph in Figure 1 belongs to the special class of 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 in a predefined linear order and all edges drawn on the plane above the sentence, with no edge crossing another. Figure 1 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, a large source of English dependencies is automatically generated from the Penn Treebank (Marcus et al. 1993) and is by construction exclusively projective (Yamada and Matsumoto 2003). 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 2. In languages with flexible word order, such as Czech, Dutch and German, non-projective dependencies are more frequent. In general, 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. 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.

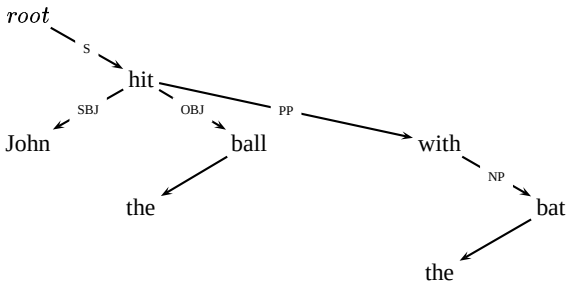


Figure 3
An example of a labeled dependency graph.

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*. In this work we will only address the problem of parsing dependency graphs that are trees, which is a common constraint (Nivre 2005). McDonald and Pereira (McDonald and Pereira 2006) show how to extend the algorithms presented here to non-tree dependency graphs.

As mentioned before, directed edges in a dependency graph represent words and their syntactic modifiers. 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. 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 to produce the graph in Figure 3.

In many cases the head-modifier relationship is 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? The distinction between various levels of dependency representation can be beneficial here. For example, 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. As a result, 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.

Recently, there has been a surge of interest in producing computational models for dependency parsing. Relative to phrase-structure formalisms such as CFGs, TAG, LTAG, or CCGs, dependencies can be considered a light-weight representation. As a result, they are much simpler to represent and analyze computationally. However, dependency graphs still encode much of the predicate-argument information relevant to many NLP problems and have been employed in a variety of applications such as relation extraction (Culotta and Sorensen 2004), machine translation (Ding and Palmer 2005), synonym generation (Shinyama et al. 2002) and lexical resource augmentation (Snow et al. 2004).

Another advantage of dependency parsers is the existence of numerous large annotated resources. The Prague Dependency Treebank (Hajič 1998; Hajič et al. 2001) contains tens of thousands of human annotated dependency representations for Czech. The Nordic Treebank Network[1] is a group of European researchers that have developed many tools for dependency parsing including treebanks for Danish (Kromann 2003) and Swedish (Einarsson 1976). There are also Turkish (Oflazer et al. 2003) and Arabic (Hajič et al. 2004) dependency treebanks available. Recently, the organizers of the shared-task at CoNLL 2006 (Buchholz et al. 2006) standardized data sets for 13 languages: Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish (Hajič et al. 2004; Simov et al. 2005; Simov and Osenova 2003; Chen et al. 2003; Böhmová et al. 2003; Kromann 2003; van der Beek et al. 2002; Brants et al. 2002; Kawata and Bartels 2000; Afonso et al. 2002; Džeroski et al. 2006; Civit Torruella and Martí Antonín 2002; Nilsson et al. 2005; Oflazer et al. 2003; Atalay et al. 2003). Furthermore, most phrase-structure treebanks typically have common tools for converting them into dependency treebanks including both the English and Chinese Penn treebanks (Marcus et al. 1993; Xue et al. 2004).

1.1 Data-Driven Dependency Parsing

In this work we focus on parsing models that discriminate between better and worse parses for a given input sentence[2]. We assume no underlying grammar that generates the language. In fact, one can think of our parser using 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 (Collins 1999) 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 and 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 since we assume this information is contained implicitly in the annotated data.

The data-driven parsing framework is graphically displayed in Figure 4. 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 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 here are language independent. That is, if given training data in English, the learning algorithm will produce an accurate English parsing model. Similarly, if given training data in Spanish, it will produce an accurate Spanish parsing model.

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 $\boldsymbol{y}$. For many formalisms, the parsing model defines the inference algorithm. For example, if the model is a Probabilistic

1 http://w3.msi.vxu.se/~nivre/research/nt.html

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

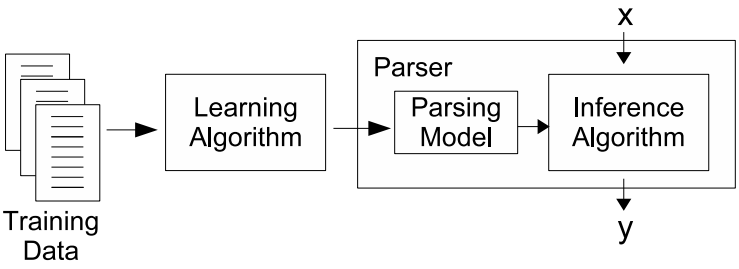


Figure 4
Outline of generic syntactic parsing framework.

Context Free Grammar, then the inference algorithm will most likely by CKY (Younger 1967) or Earley's (Early 1968), but in principle this is not necessarily true.

1.2 Previous Work

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 $\boldsymbol{y}$ (Charniak 2000; Collins 1999). 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 strong 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 (Tjong Kim Sang and De Meulder 2003) and relation extraction (Zelenko et al. 2003). The obvious question the parsing community has asked is, *can the benefits of discriminative learning be applied to parsing?*

An early work on discriminative parsing is the local decision maximum entropy model of Ratnaparkhi (Ratnaparkhi 1999), 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 (Lafferty et al. 2001). A similar system was proposed by Henderson (Henderson 2003) 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 (Clark and Curran 2004; Collins and Roark 2004; Riezler et al. 2002; Taskar et al. 2004). The most likely reason for this is that discriminative training requires repeatedly reparsing the training corpus with the current model to determine the parameter updates that will improve the training criterion. This general description applies equally for extensions to parsing of standard discriminative training techniques such as maximum entropy (Berger et al. 1996), the perceptron algorithm (Rosenblatt 1958), or support vector machines (Boser et al. 1992), which we call here *linear parsing models* because they all score a parse $\boldsymbol{y}$ 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 (Collins 1999) 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 (Charniak 2000) and Collins (Collins 1999).

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 (Collins and Duffy 2002) and the Charniak parser (Charniak and Johnson 2005) and typically results in a 10% relative reduction in error.

For data-driven dependency parsing, Eisner (Eisner 1996) 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 (Yamada and Matsumoto 2003) trained support vector machines (SVM) to make parsing decisions in a shift-reduce dependency parser for English. As in Ratnaparkhi's parser (Ratnaparkhi 1999), the classifiers are trained on individual decisions rather than on the overall quality of the parse. Nivre and Scholz (Nivre and Scholz 2004) 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 (Hajič 1998), there has also been extensive research on parsing Czech at that institution (Collins et al. 1999; Ribarov 2004; Zeman 2004).

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 (Sleator and Temperley 1993) and the system of Lin (Lin 1998). Nivre and Scholz (Nivre and Scholz 2004) provide two systems, one a pure dependency parser and the other a labeled model that labels edges with syntactic categories. Wang and Harper (Wang and Harper 2004) 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 (Yamada and Matsumoto 2003) and Nivre and Scholz (Nivre and Scholz 2004) 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 (Charniak 2000; Collins 1999; Yamada and Matsumoto 2003). These systems are based on finding phrase structure through nested chart parsing algorithms and cannot model non-projective edges tractably. However, Yamada and Matsumoto (Yamada and Matsumoto 2003) showed that these models are still very powerful since they consider much more information when making decisions then pure dependency parsers.

For non-projective dependency parsing, tractable inference algorithms have been given by Tapanainen and Järvinen (Tapanainen and Järvinen 1997) and Kahane et al. (Kahane et al. 1998). Nivre and Nilsson (Nivre and Nilsson 2005) 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 (Wang and Harper 2004) for English, which presents very good results using a constraint dependency grammar framework that is rich in lexical and syntactic information. 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 (Neuhaus and Böker 1997) 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 (Hirakawa 2001) who, like us, relates the problem of dependency parsing to finding spanning trees for Japanese text. However, that parsing algorithm uses branch and bound techniques 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 (Ribarov 2004) was developed independently of this work. 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 (Rosenblatt 1958), which is a learning algorithm related to the framework presented in Section 2. However, Ribarov's empirical evaluation on the Prague Dependency Treebank (Hajič 1998) results in an accuracy well below the state-of-the-art. This is most likely due to a very impoverished feature representation that focuses primarily on aspects of the complex Czech morphology and does not consider lexical or contextual information. We also generalize the dependency parsing as maximum spanning tree framework and consider trees with larger (and possibly intractable) feature contexts as well as apply the resulting parser to new domains and in real world applications.

2. Large-Margin Online Learning

In this section 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) (Lafferty et al. 2001), several researchers developed margin-based learning alternatives, in particular maximum margin Markov networks (M^3Ns) (Taskar et al. 2003) and the related methods of Tsochantaridis et al. (Tsochantaridis et al. 2004). These algorithms have proven successful in several real world applications including sequential classification (McCallum 2003; McDonald and Pereira 2005; Sha and Pereira 2003; Taskar et al. 2003), image labeling (He et al. 2004), natural language parsing (Taskar et al. 2004; Tsochantaridis et al. 2004) and Web page classification (Taskar et al. 2003). 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 (Rosenblatt 1958), 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 (Collins 2002), who uses an approximation to the voted perceptron algorithm (Freund and Schapire 1999), called here the averaged perceptron algorithm, for sequential classification problems. This method has since been successfully adapted to parsing (Collins and Roark 2004), language modeling (Roark et al. 2004) and more recently word alignment (Moore 2005). Perceptron-based approaches have gained a wide acceptance since they reduce learning to inference 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 (Boser et al. 1992). 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 (Crammer et al. 2003; Crammer and Singer 2003; Crammer et al. 2006)) 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.

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 5. 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 (Collins 2002).

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

2.3 Margin Infused Relaxed Algorithm (MIRA)

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

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

Figure 5
Generic online learning algorithm.

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) (Crammer et al. 2003; Crammer and Singer 2003; Crammer et al. 2006) 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 5,

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

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 (Censor and Zenios 1997). Crammer and Singer (Crammer and Singer 2003) and Crammer et al. (Crammer et al. 2003, 2006) 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 (Taskar et al. 2003). For instance, consider the correct graph in Figure 1 versus the incorrect one in Figure 6. 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 (Taskar 2004) 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 is based on the percentage of words that modify the correct head in the graph.

To use MIRA for structured classification, we follow the common method of equating structure prediction to multi-class classification, where each structure is a possible class for

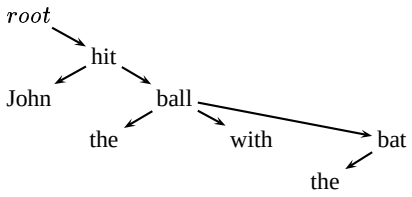


Figure 6
An example incorrect dependency parse relative to that in Figure 1. The loss of this parse is 2 since *with* and *bat* are incorrectly identified as modifiers of *ball*.

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, which is the case for dependency parsing.

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 5, line 4) would then be:

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

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. (Tsochantaridis et al. 2004). 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.

Another solution to the exponetial set of margin contraints is to factor these constraints relative to the structure of the output to produce an equivalent polynomial sized set of constraints. Taskar et al. (Taskar et al. 2003, 2004) 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 Equation (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.

3. Dependency Parsing Inference as the Maximum Spanning Tree Problem

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.

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 one can view the score of a dependency tree $\boldsymbol{y}$ for sentence $\boldsymbol{x}$ as,

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

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 (Section 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 (Tarjan 1977).

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

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

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). 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 the analysis that follows, we make the assumption that calculating $s(i,j)$ is $O(1)$. In fact, this is slightly misleading since $\mathbf{w}$ and $\mathbf{f}$ 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. This constant is typically very

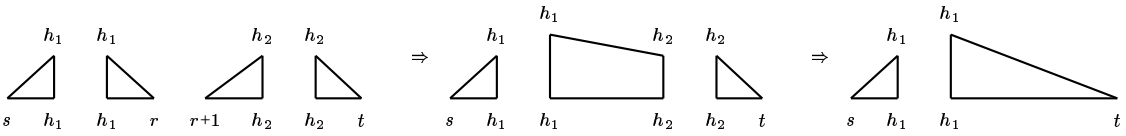


Figure 7
Cubic parsing algorithm of Eisner.

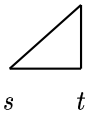
small (roughly 100), especially when compared to grammar constants in phrase-based models, which can be on the order of tens of thousands when extracted from a large treebank.

3.1 Projective Parsing Algorithms

Using a slightly modified version of the CKY (Younger 1967) chart parsing algorithm, it is possible to generate and represent all projective dependency trees in a forest that is $O(n^5)$ in size and takes $O(n^5)$ time to create, which is equivalent to context-free phrase-structure parsing. However, Eisner (Eisner 1996) made the observation that if one keeps the head of each chart item to either the left or right periphery of that item, then it is possible to parse in $O(n^3)$. The idea is to parse the left and right dependents of a word independently, and combine them at a later stage. This removes the need for the additional head indices of the $O(n^5)$ algorithm and requires only two additional binary variables that specify the direction of the item (either gathering left dependents or gathering right dependents) and whether an item is complete (available to gather more dependents). Figure 7 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.

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,



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

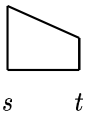
    % First: create incomplete items
    $C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s))$ (*)
    $C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$

    % 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

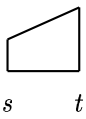
Figure 8
Pseudo-code for bottom-up Eisner cubic parsing algorithm.

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

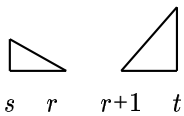


The Eisner algorithm fills in the dynamic programming table bottom-up just like the CKY parsing algorithm (Younger 1967) by finding optimal subtrees for substrings of increasing increasing length. Pseudo code for filling up the dynamic programming table is in Figure 8.

Consider the line in Figure 8 indicated by (*). This says that to find the best score for an incomplete left subtree



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



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

By forcing a unique root at the left-hand side of the sentence, the score of the best tree for the entire sentence is $C[1][n][\rightarrow][1]$. This can be shown easily by structural induction using the

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 9
Chu-Liu-Edmonds algorithm for finding maximum spanning trees in directed graphs.

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.

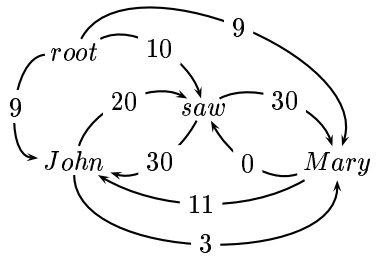
In addition to running in $O(n^3)$, the Eisner algorithm has the additional benefit that it is a bottom-up dynamic programming chart parsing algorithm allowing for k-best extensions that increase complexity by a multiplicative factor of $O(k \log k)$ (Huang and Chiang 2005).

3.2 Non-projective Parsing Algorithms

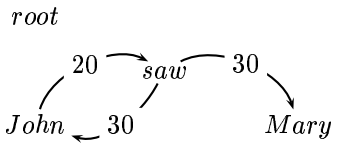
To find the highest scoring non-projective tree we simply search the entire space of spanning trees with no restrictions. Well known algorithms exist for the less general case of finding spanning trees in undirected graphs (Cormen et al. 1990), as well as k-best extensions to them (Eppstein 1990). Efficient algorithms for the directed case are less well known, but they exist. We will use here the Chu-Liu-Edmonds algorithm (Chu and Liu 1965; Edmonds 1967), sketched in Figure 9 following Georgiadis (Georgiadis 2003). Informally, the algorithm has each vertex in the graph greedily select the incoming edge with highest weight. If a tree results, it must be the maximum spanning tree. If not, there must be a cycle. The procedure identifies a cycle and contracts it into a single vertex and recalculates edge weights going into and out of the cycle. It can be shown that a maximum spanning tree on the resulting contracted graph is equivalent to a maximum spanning tree in the original graph (Georgiadis 2003). Hence the algorithm can recursively call itself on the new graph. Naively, this algorithm runs in $O(n^3)$ time since each recursive call takes $O(n^2)$ to find the highest incoming edge for each word and to contract the graph. There are at most $O(n)$ recursive calls since we cannot contract the graph more then n times. However, Tarjan (Tarjan 1977) 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 (Camerini et al. 1980) 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 $G_{\boldsymbol{x}}$,

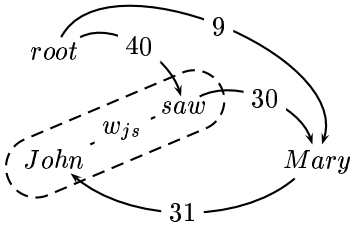


The first step of the algorithm is to find, for each word, the highest scoring incoming edge



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.

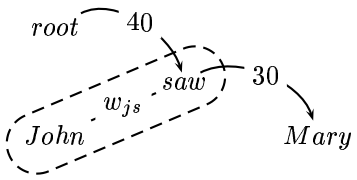
In the current example there is a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 9.



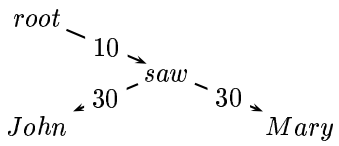
The new vertex w_{js} represents the contraction of vertices *John* and *saw*. The edge from w_{js} to *Mary* is 30 since that is the highest scoring edge from any vertex in w_{js}. The edge from *root* into w_{js} is set to 40 since this represents the score of the best spanning tree originating from *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 (Georgiadis 2003). 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.

4. Beyond Edge Factorization

Restricting scores to a single edge in a dependency tree is a very impoverished view of dependency parsing. Yamada and Matsumoto (Yamada and Matsumoto 2003) 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.

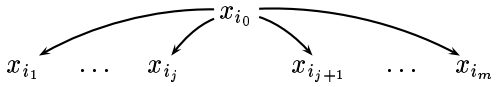
Here we will focus on methods for parsing *second-order* spanning trees. These models factor the score of the tree into the sum of adjacent edge pairs. To quantify this, consider the example from Figure 1, with words indexed: root(0) John(1) hit(2) the(3) ball(4) with(5) the(6) bat(7). Using a first-order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

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

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 efficient 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,



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 we will define the second order score to directly 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 (Charniak 2000) 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.

4.1 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 (Eisner 1996). Graphically the intuition behind the algorithm is given in Figure 10. The key

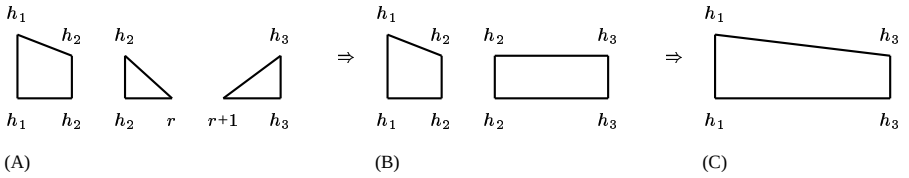


Figure 10
An extension of the Eisner algorithm to second-order dependency parsing. This figure shows how h_1 creates a dependency to h_3 with the second-order knowledge that the last dependent of h_1 was h_2. This is done through the creation of a *sibling* item in part (B).

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* type. Pseudo-code for the algorithm is given in Figure 11. 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 $O(k \log(k) n^3)$.

4.2 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. 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

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
    $C[s][t][\leftarrow][0] = C[s][t-1][\rightarrow][1] + C[t-1][t][\leftarrow][1] + s(t, \text{-}, s)$
    $C[s][t][\rightarrow][0] = C[s][s][\rightarrow][1] + C[s+1][t][\leftarrow][1] + s(s, \text{-}, t)$

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

  end for
end for

Figure 11
Pseudo-code for bottom-up second-order Eisner parsing algorithm.

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 factorization other than first-order[3]. 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 (Nivre and Nilsson 2005). 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

3 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.

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

Figure 12
Approximate second-order non-projective parsing algorithm.

tree. Pseudo-code for the algorithm is given in Figure 12. The expression $\boldsymbol{y}[i \rightarrow j]$ 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 $\boldsymbol{y}$ 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**.

We should note the similarity of this approximate dependency parsing algorithm with that of Foth et al. (Foth et al. 2000). In that work they describe an algorithm for constraint based dependency parsing (Maruyama 1990; Harper and Helzerman 1995) 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.

5. Feature Representation

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 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 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 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 easily 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.

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 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

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 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.

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.

5.1 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 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.

6. Initial Experiments

6.1 Data Sets

We performed these experiments on three sets of data, the Penn English Treebank (Marcus et al. 1993), the Czech Prague Dependency Treebank (PDT) v1.0 (Hajič 1998; Hajič et al. 2001) and the Penn Chinese Treebank (Xue et al. 2004). For the English data we extracted dependency trees using the rules of Yamada and Matsumoto (Yamada and Matsumoto 2003), which are similar, but not identical, to those used by Collins (Collins 1999) and Magerman (Magerman 1995). 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 (Yamada and Matsumoto 2003) and did not include punctuation in the calculation of accuracies. For the test set, the number of words without punctuation is 49,892. Since our system assumes part-of-speech information as input, we used the maximum entropy part-of-speech tagger of Ratnaparkhi (Ratnaparkhi 1996) 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. (Collins et al. 1999). 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 (Xue et al. 2004) 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.

6.2 Results: Unlabeled Dependencies

This section 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 (Yamada and Matsumoto 2003).

6.2.1 Projective Parsing Results. We compare five systems,

- **Y&M2003:** The Yamada and Matsumoto parser (Yamada and Matsumoto 2003) 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 (Ratnaparkhi 1999).
- **N&S2004:** The parser of Nivre and Scholz (Nivre and Scholz 2004) is a memory based parser with an approximate linear parsing algorithm.

	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 2
Unlabeled projective dependency parsing results. *Accuracy* is the percentage of words modifi fying the correct head. *Complete* is the percentage of sentences for which the entire predicted dependency graph was correct.

- **N&N2005:** The parser of (Nivre and Nilsson 2005), 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.
- **1^{st}-order-proj:** This parser uses the Eisner first-order projective parsing algorithm combined with the MIRA learning framework.
- **2^{nd}-order-proj:** This parser uses the second-order extension of the Eisner algorithm combined with the MIRA learning framework.

Results are shown in Figure 2. 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 (Nivre and Nilsson 2005; Nivre and Scholz 2004). The Yamada and Matsumoto (Yamada and Matsumoto 2003) SVM parser also has this ability. However, their locally trained model can suffer from the label bias problem (Lafferty et al. 2001) 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.

6.2.2 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 (Nivre and Nilsson 2005) is a memory based parser like (Nivre and Scholz 2004). 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).
- **1^{st}-order-proj:** The first-order projective parser from Section 6.2.1.
- **2^{nd}-order-proj:** The second-order projective parser from Section 6.2.1.
- **1^{st}-order-non-proj:** This parser uses the Chu-Liu-Edmonds MST algorithm as described in Section 3.2.

	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 3
Unlabeled non-projective dependency parsing results.

- **2^{nd}-order-non-proj:** This parser uses the approximate second-order non-projective parsing algorithm described in Section 4.2.

Results are shown in Figure 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.

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.

7. Labeled Dependency Parsing

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 3 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 (Lin 1998; Nivre and Scholz 2004; Sleator and Temperley 1993). 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 (Klein 2004), 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.

7.1 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.

7.2 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[4]. 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,

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

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.

7.3 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 always improve performance 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,

4 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.

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. Is 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.

7.4 Experiments

In this section we report results for the first-order labeled dependency models described in Section 7.1 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 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 13. Running an extraction script (we used Penn2Malt (Nivre 2004)) resulted in the set of 29 labels shown in Figure 14. 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.

7.4.1 First-Order Results. Results for the first-order labeling model (Section 7.1) are shown in Table 4. 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.

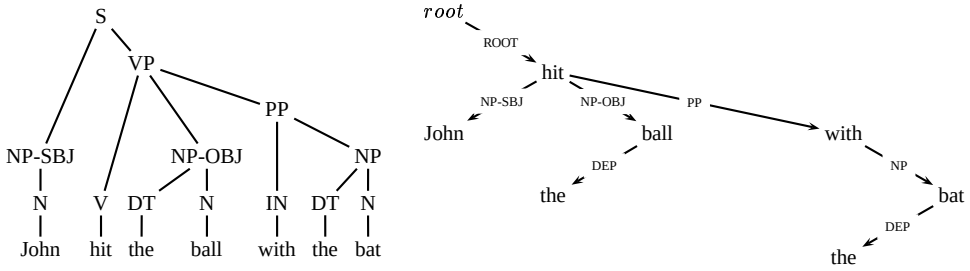


Figure 13
Converting a phrase-structure tree to a labeled dependency tree.

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 14
Labels extracted from WSJ.

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

Table 4
First-order labeling results for English.

Interestingly, unlabeled accuracy actually improves (from 90.7 to 90.9). This is consistent with previous results (Nivre and Scholz 2004) 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.

7.4.2 Two-Stage Results. Results for two-stage labeling (Section 7.3) are shown in Table 5. 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 5 and once again display the advantage of a second-order model.

	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 5
Two-stage labeling results for English.

8. Multi-lingual Dependency Parsing

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 (Buchholz et al. 2006) plus the English data set we described in Section 6.1. 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 7.3 for all experiments in this chapter.

8.1 Data Sets

We refer the reader to (Buchholz et al. 2006) for more inforamtion on the data sets used.

8.2 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?

	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 6
Dependency accuracy on 14 languages. Unlabeled (UA) and Labeled Accuracy (LA).

- **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.

8.3 Experiments

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[5]. 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 6. 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 all of its characters are unicode punctuation characters. For English we define punctuation identical to Yamada and Matsumoto (Yamada and Matsumoto 2003).

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 (Buchholz et al. 2006) and represent

5 Using the non-projective parser for all languages does not effect performance significantly.

the best reporting system for Arabic, Bulgarian, Czech, Danish, Dutch, German, Slovene and Spanish (English was not included in the shared task).

9. Summary

This paper provided an overview of the work of McDonald et al. (McDonald et al. 2005a, 2005b; McDonald and Pereira 2006; McDonald et al. 2006) on global inference and learning algorithms for data-driven dependency parsing. Further details can be found in the thesis of McDonald (McDonald 2006), which includes analysis, example feature set extractions, phrase-structure conversion head rules, applications, learning and parsing non-tree dependency graphs and more experiments not only on parsing accuracy, but also on CPU performance.

10. Acknowledgements

The following people have either contributed directly to this work, or have engaged the authors in imporant discussions related to this work: Kiril Ribarov, Jan Hajic, John Blitzer, Joakim Nivre, Nikhil Dinesh, Mark Liberman, Mitch Marcus, Aravind Joshi, Jason Eisner, Noah Smith, Hal Daume, Keith Hall, Liang Huang, Simon Corston-Oliver, Sebastian Riedel, and Fred Jelinek.

References

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

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

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

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

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2003. The PDT: a 3-level annotation scenario. In Abeillé (Abeillé 2003), chapter 7.

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

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

S. Buchholz, E. Marsi, A. Dubey, and Y. Krymolowski. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

P. M. Camerini, L. Fratta, and F. Maffioli. 1980. The k best spanning arborescences of a network. *Networks*, 10(2):91–110.

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

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics (ACL)*.

K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. 2003. Sinica Treebank: Design criteria, representational issues and implementation. In Abeillé (Abeillé 2003), chapter 13, pages 231–248.

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

M. Civit Torruella and M^{a} A. Martí Antonín. 2002. Design principles for a Spanish treebank. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*.

S. Clark and J.R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

M. Collins and N. Duffy. 2002. 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)*.
M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann. 1999. A statistical parser for Czech. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.
M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*.
T.H. Cormen, C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill.
K. Crammer and Y. Singer. 2001. On the algorithmic implementation of multiclass kernel based vector machines. *Journal of Machine Learning Research*.
K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*.
K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2003. Online passive aggressive algorithms. In *Proceedings of Neural Information Processing Systems (NIPS)*.
K. Crammer, O. Dekel, J. Keshat, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive aggressive algorithms. *Journal of Machine Learning Research*.
A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
S. Džeroski, T. Erjavec, N. Ledinek, P. Pajas, Z. Žabokrtsky, and A. Žele. 2006. Towards a Slovene dependency treebank. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC)*.
J. Early. 1968. *An Effi cient Context-Free Parsing Algorithm*. Ph.D. thesis, Carnegie Mellon University.
J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.
J. Einarsson. 1976. Talbankens skriftspråkskonkordans.
J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the International Conference on Computational Linguistics (COLING)*.
D. Eppstein. 1990. Finding the k smallest spanning trees. In *2nd Scandanavian Workshop on Algorithm Theory*.
K. Foth, W. Menzel, and I. Schröder. 2000. A transformation-based parsing technique with anytime properties. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.
Y. Freund and R.E. Schapire. 1999. Large margin classifi cation using the perceptron algorithm. *Machine Learning*, 37(3):277–296.
H. Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and Control*.
L. Georgiadis. 2003. Arborescence optimization problems solvable by Edmonds' algorithm. *Theoretical Computer Science*, 301:427 – 437.
J. Hajič, E. Hajičová, P. Pajas, J. Panevova, P. Sgall, and B. Vidova Hladka. 2001. The Prague Dependency Treebank 1.0 CDROM. Linguistics Data Consortium Cat. No. LDC2001T10.
J. Hajič, O. Smrž, P. Zemánek, J. Šnaidauf, and E. Beška. 2004. 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.
J. Hajič. 1998. Building a syntactically annotated corpus: The Prague dependency treebank. *Issues of Valency and Meaning*, pages 106–132.
M. P. Harper and R. A. Helzerman. 1995. Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language*.
D. G. Hays. 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.
X. He, R. Zemel, and M. Carreira-Perpinan. 2004. Multiscale conditional random fi elds for image labelling. In *Proceedings of Conference on Vision and Pattern Recognition*.
J. Henderson. 2003. 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)*.

H. Hirakawa. 2001. Semantic dependency analysis method for Japanese based on optimum tree search algorithm. In *Proceedings of the Pacific Association for Computational Linguistics*.
L. Huang and D. Chiang. 2005. Better k-best parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.
R. Hudson. 1984. *Word Grammar*. Blackwell.
A.K. Joshi. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? *Natural Language Parsing*.
S. Kahane, A. Nasr, and O Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
Y. Kawata and J. Bartels. 2000. Stylebook for the Japanese Treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen.
D. Klein. 2004. *The Unsupervised Learning of Natural Language Structure*. Ph.D. thesis, Stanford University.
M. T. Kromann. 2003. The Danish Dependency Treebank and the underlying linguistic theory. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*.
J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.
D. Lin. 1998. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*.
D.M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.
H. Maruyama. 1990. Structural disambiguation with constraint propagation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
A. McCallum. 2003. Efficiently inducing features of conditional random fields. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.
R. McDonald and F. Pereira. 2005. Identifying gene and protein mentions in text using conditional random fields. *BMC Bioinformatics*, 6:Supp1(S6).
R. McDonald and F. Pereira. 2006. 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)*.
R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005b. 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)*.
R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.
R. McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.
I.A. Meĺčuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.
R. Moore. 2005. 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)*.
P. Neuhaus and N. Böker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
J. Nilsson, J. Hall, and J. Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proceedings of the NODALIDA Special Session on Treebanks*.
J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of english text. In *Proceedings of the International Conference on Computational Linguistics (COLING)*.
Joakim Nivre. 2004. Penn2malt. http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html.

J. Nivre. 2005. Dependency grammar and dependency parsing. Technical Report MSI report 05133, Växjö University: School of Mathematics and Systems Engineering.
K. Oflazer, B. Say, D. Zeynep Hakkani-Tür, and G. Tür. 2003. Building a Turkish treebank. In Abeillé (Abeillé 2003), chapter 15.
A. Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142.
A. Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.
K. Ribarov. 2004. *Automatic building of a dependency tree*. Ph.D. thesis, Charles University.
S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
B. Roark, M. Saraclar, M. Collins, and M. Johnson. 2004. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 68:386–407.
P. Sgall, E. Hajičová, and J. Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.
F. Sha and F. Pereira. 2003. 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.
Y. Shinyama, S. Sekine, K. Sudo, and R. Grishman. 2002. Automatic paraphrase acquisition from news articles. In *Proceedings of the Human Language Technology Conference (HLT)*.
K. Simov and P. Osenova. 2003. Practical annotation scheme for an HPSG treebank of Bulgarian. In *Proceedings of the 4th International Workshop on Linguistically Interpreteted Corpora (LINC)*, pages 17–24.
K. Simov, P. Osenova, A. Simov, and M. Kouylekov. 2005. 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.
D. Sleator and D. Temperley. 1993. Parsing English with a link grammar. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.
R. Snow, D. Jurafsky, and A. Y. Ng. 2004. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of Neural Information Processing Systems (NIPS)*.
M. Steedman. 2000. *The Syntactic Process*. MIT Press.
P. Tapanainen and T. Järvinen. 1997. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*.
R.E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7:25–35.
B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *Proceedings of Neural Information Processing Systems (NIPS)*.
B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*.
B. Taskar. 2004. *Learning Structured Prediction Models: A Large Margin Approach*. Ph.D. thesis, Stanford.
L. Tesnière. 1959. *Éléments de syntaxe structurale*. Editions Klincksieck.
E.F. Tjong Kim Sang and F. De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.
I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning*.
L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*.
W. Wang and M. P. Harper. 2004. A statistical constraint dependency grammar (CDG) parser. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*.
N. Xue, F. Xia, F. Chiou, and M. Palmer. 2004. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*.
H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.

D.H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 12(4):361–379.
D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106.
D. Zeman. 2004. *Parsing with a Statistical Dependency Model*. Ph.D. thesis, Univerzita Karlova, Praha.

Characterizing the Errors of Data-Driven Dependency Parsing Models

Ryan McDonald
Google Inc.
76 Ninth Avenue
New York, NY 10011
ryanmcd@google.com

Joakim Nivre
Växjö University Uppsala University
35195 Växjö 75126 Uppsala
Sweden Sweden
nivre@msi.vxu.se

Abstract

We present a comparative error analysis of the two dominant approaches in data-driven dependency parsing: global, exhaustive, graph-based models, and local, greedy, transition-based models. We show that, in spite of similar performance overall, the two models produce different types of errors, in a way that can be explained by theoretical properties of the two models. This analysis leads to new directions for parser development.

1 Introduction

Syntactic dependency representations have a long history in descriptive and theoretical linguistics and many formal models have been advanced (Hudson, 1984; Mel'čuk, 1988; Sgall et al., 1986; Maruyama, 1990). A dependency graph of a sentence represents each word and its syntactic modifiers through labeled directed arcs, as shown in Figure 1, taken from the Prague Dependency Treebank (Böhmová et al., 2003). A primary advantage of dependency representations is that they have a natural mechanism for representing discontinuous constructions, arising from long distance dependencies or free word order, through *non-projective* dependency arcs, exemplified by the arc from *jedna* to *Z* in Figure 1.

Syntactic dependency graphs have recently gained a wide interest in the computational linguistics community and have been successfully employed for many problems ranging from machine translation (Ding and Palmer, 2004) to ontology construction (Snow et al., 2004). In this work we focus on a common parsing paradigm called *data-driven dependency parsing*. Unlike grammar-based parsing, data-driven approaches learn to produce dependency graphs for sentences solely from an annotated corpus. The advantage of such models is that they are easily ported to any domain or language in which annotated resources exist.

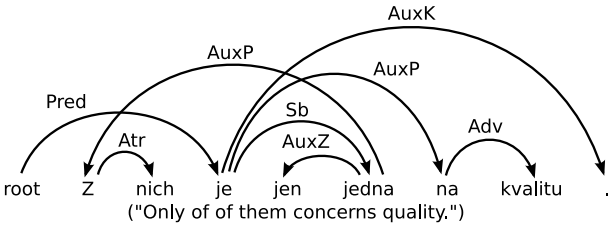


Figure 1: Example dependency graph.

As evident from the CoNLL-X shared task on dependency parsing (Buchholz and Marsi, 2006), there are currently two dominant models for data-driven dependency parsing. The first is what Buchholz and Marsi (2006) call the "all-pairs" approach, where every possible arc is considered in the construction of the optimal parse. The second is the "stepwise" approach, where the optimal parse is built stepwise and where the subset of possible arcs considered depend on previous decisions. Theoretically, these models are extremely different. The all-pairs models are globally trained, use exact (or near exact) inference algorithms, and define features over a limited history of parsing decisions. The stepwise models use local training and greedy inference algorithms, but define features over a rich history of parse decisions. However, both models obtain similar parsing accuracies

	McDonald	Nivre
Arabic	66.91	66.71
Bulgarian	87.57	87.41
Chinese	85.90	86.92
Czech	80.18	78.42
Danish	84.79	84.77
Dutch	79.19	78.59
German	87.34	85.82
Japanese	90.71	91.65
Portuguese	86.82	87.60
Slovene	73.44	70.30
Spanish	82.25	81.29
Swedish	82.55	84.58
Turkish	63.19	65.68
Overall	80.83	80.75

Table 1: Labeled parsing accuracy for top scoring systems at CoNLL-X (Buchholz and Marsi, 2006).

on a variety of languages, as seen in Table 1, which shows results for the two top performing systems in the CoNLL-X shared task, McDonald et al. (2006) ("all-pairs") and Nivre et al. (2006) ("stepwise").

Despite the similar performance in terms of overall accuracy, there are indications that the two types of models exhibit different behaviour. For example, Sagae and Lavie (2006) displayed that combining the predictions of both parsing models can lead to significantly improved accuracies. In order to pave the way for new and better methods, a much more detailed error analysis is needed to understand the strengths and weaknesses of different approaches. In this work we set out to do just that, focusing on the two top performing systems from the CoNLL-X shared task as representatives of the two dominant models in data-driven dependency parsing.

2 Two Models for Dependency Parsing

2.1 Preliminaries

Let $L = \{l_1, \ldots, l_{|L|}\}$ be a set of permissible arc labels. Let $x = w_0, w_1, \ldots, w_n$ be an input sentence where w_0=*root*. Formally, a dependency graph for an input sentence x is a labeled directed graph $G = (V, A)$ consisting of a set of nodes V and a set of labeled directed arcs $A \subseteq V \times V \times L$, i.e., if $(i, j, l) \in A$ for $i, j \in V$ and $l \in L$, then there is an arc from node i to node j with label l in the graph. A dependency graph G for sentence x must satisfy the following properties:

1. $V = \{0, 1, \ldots, n\}$
2. If $(i, j, l) \in A$, then $j \neq 0$.
3. If $(i, j, l) \in A$, then for all $i' \in V - \{i\}$ and $l' \in L$, $(i', j, l') \notin A$.
4. For all $j \in V - \{0\}$, there is a (possibly empty) sequence of nodes $i_1, \ldots, i_m \in V$ and labels $l_1, \ldots, l_m, l \in L$ such that $(0, i_1, l_1), (i_1, i_2, l_2), \ldots, (i_m, j, l) \in A$.

The constraints state that the dependency graph spans the entire input (1); that the node 0 is a root (2); that each node has at most one incoming arc in the graph (3); and that the graph is connected through directed paths from the node 0 to every other node in the graph (4). A dependency graph satisfying these constraints is a directed tree originating out of the root node 0. We say that an arc (i, j, l) is *non-projective* if not all words k occurring between i and j in the linear order are dominated by i (where dominance is the transitive closure of the arc relation).

2.2 Global, Exhaustive, Graph-Based Parsing

For an input sentence, $x = w_0, w_1, \ldots, w_n$ consider the dense graph $G_x = (V_x, A_x)$ where:

1. $V_x = \{0, 1, \ldots, n\}$
2. $A_x = \{(i, j, l) \mid \forall\, i, j \in V_x \text{ and } l \in L\}$

Let $D(G_x)$ represent the subgraphs of graph G_x that are valid dependency graphs for the sentence x. Since G_x contains all possible labeled arcs, the set $D(G_x)$ must necessarily contain all valid dependency graphs for x.

Assume that there exists a dependency arc scoring function, $s : V \times V \times L \rightarrow \mathbb{R}$. Furthermore, define the score of a graph as the sum of its arc scores,

$$s(G = (V, A)) = \sum_{(i,j,l) \in A} s(i, j, l)$$

The score of a dependency arc, $s(i, j, l)$ represents the likelihood of creating a dependency from word w_i to word w_j with the label l. If the arc score function is known a priori, then the parsing problem can be stated as,

$$G = \arg\max_{G \in D(G_x)} s(G) = \arg\max_{G \in D(G_x)} \sum_{(i,j,l) \in A} s(i,j,l)$$

This problem is equivalent to finding the highest scoring directed spanning tree in the graph G_x originating out of the root node 0, which can be solved for both the labeled and unlabeled case in $O(n^2)$ time (McDonald et al., 2005b). In this approach, non-projective arcs are produced naturally through the inference algorithm that searches over all possible directed trees, whether projective or not.

The parsing models of McDonald work primarily in this framework. To learn arc scores, these models use large-margin structured learning algorithms (McDonald et al., 2005a), which optimize the parameters of the model to maximize the score margin between the correct dependency graph and all incorrect dependency graphs for every sentence in a training set. The learning procedure is global since model parameters are set relative to the classification of the entire dependency graph, and not just over single arc attachment decisions. The primary disadvantage of these models is that the feature representation is restricted to a limited number of graph arcs. This restriction is required so that both inference and learning are tractable.

The specific model studied in this work is that presented by McDonald et al. (2006), which factors scores over pairs of arcs (instead of just single arcs) and uses near exhaustive search for unlabeled parsing coupled with a separate classifier to label each arc. We call this system MSTParser, which is also the name of the freely available implementation.[1]

2.3 Local, Greedy, Transition-Based Parsing

A *transition system* for dependency parsing defines

1. a set C of *parser configurations*, each of which defines a (partially built) dependency graph G
2. a set T of *transitions*, each a function $t : C \rightarrow C$
3. for every sentence $x = w_0, w_1, \ldots, w_n$,
   (a) a unique *initial* configuration c_x
   (b) a set C_x of *terminal* configurations

A *transition sequence* $C_{x,m} = (c_x, c_1, \ldots, c_m)$ for a sentence x is a sequence of configurations such that $c_m \in C_x$ and, for every c_i ($c_i \neq c_x$), there is a transition $t \in T$ such that $c_i = t(c_{i-1})$. The dependency graph assigned to x by $C_{x,m}$ is the graph G_m defined by the terminal configuration c_m.

Assume that there exists a transition scoring function, $s : C \times T \rightarrow \mathbb{R}$. The score of a transition t in a configuration c, $s(c,t)$, represents the likelihood of taking transition t out of configuration c. The parsing problem consists in finding a terminal configuration $c_m \in C_x$, starting from the initial configuration c_x and taking the optimal transition $t^* = \arg\max_{t \in T} s(c,t)$ out of every configuration c. This can be seen as a greedy search for the optimal dependency graph, based on a sequence of locally optimal decisions in terms of the transition system.

Many transition systems for data-driven dependency parsing are inspired by shift-reduce parsing, where configurations contain a stack for storing partially processed nodes. Transitions in such systems add arcs to the dependency graph and/or manipulate the stack. One example is the transition system defined by Nivre (2003), which parses a sentence $x = w_0, w_1, \ldots, w_n$ in $O(n)$ time, producing a projective dependency graph satisfying conditions 1–4 in section 2.1, possibly after adding arcs $(0, i, l_r)$ for every node $i \neq 0$ that is a root in the output graph (where l_r is a special label for root modifiers). Nivre and Nilsson (2005) showed how the restriction to projective dependency graphs could be lifted by using graph transformation techniques to preprocess training data and post-process parser output, so-called *pseudo-projective parsing*.

To learn transition scores, these systems use discriminative learning methods, e.g., memory-based learning or support vector machines. The learning procedure is local since only single transitions are scored, not entire transition sequences. The primary advantage of these models is that features are not restricted to a limited number of graph arcs but can take into account the entire dependency graph built so far. The main disadvantage is that the greedy parsing strategy may lead to error propagation.

The specific model studied in this work is that presented by Nivre et al. (2006), which uses labeled pseudo-projective parsing with support vector machines. We call this system MaltParser, which is also

[1] http://mstparser.sourceforge.net

the name of the freely available implementation.[2]

2.4 Comparison

These models differ primarily with respect to three important properties.

1. **Inference:** MaltParser uses a transition-based inference algorithm that greedily chooses the best parsing decision based on a trained classifier and current parser history. MSTParser instead uses near exhaustive search over a dense graphical representation of the sentence to find the dependency graph that maximizes the score.

2. **Training:** MaltParser trains a model to make a single classification decision (choose the next transition). MSTParser trains a model to maximize the global score of correct graphs.

3. **Feature Representation:** MaltParser can introduce a rich feature history based on previous parser decisions. MSTParser is forced to restrict the score of features to a single or pair of nearby parsing decisions in order to make exhaustive inference tractable.

These differences highlight an inherent trade-off between exhaustive inference algorithms plus global learning and expressiveness of feature representations. MSTParser favors the former at the expense of the latter and MaltParser the opposite.

3 The CoNLL-X Shared Task

The CoNLL-X shared task (Buchholz and Marsi, 2006) was a large-scale evaluation of data-driven dependency parsers, with data from 13 different languages and 19 participating systems. The official evaluation metric was the *labeled attachment score* (LAS), defined as the percentage of tokens, excluding punctuation, that are assigned both the correct head and the correct dependency label.[3]

The output of all systems that participated in the shared task are available for download and constitute a rich resource for comparative error analysis. The data used in the experiments below are the outputs of MSTParser and MaltParser for all 13 languages, together with the corresponding gold standard graphs used in the evaluation. We constructed the data by simply concatenating a system's output for every language. This resulted in a single output file for each system and a corresponding single gold standard file. This method is sound because the data sets for each language contain approximately the same number of tokens – 5,000. Thus, evaluating system performance over the aggregated files can be roughly viewed as measuring system performance through an equally weighted arithmetic mean over the languages.

It could be argued that a language by language comparison would be more appropriate than comparing system performance across all languages. However, as table Table 1 shows, the difference in accuracy between the two systems is typically small for all languages, and only in a few cases is this difference significant. Furthermore, by aggregating over all languages we gain better statistical estimates of parser errors, since the data set for each individual language is very small.

4 Error Analysis

The primary purpose of this study is to characterize the errors made by standard data-driven dependency parsing models. To that end, we present a large set of experiments that relate parsing errors to a set of linguistic and structural properties of the input and predicted/gold standard dependency graphs. We argue that the results can be correlated to specific theoretical aspects of each model – in particular the trade-off highlighted in Section 2.4.

For simplicity, all experiments report labeled parsing accuracies. Identical experiments using unlabeled parsing accuracies did not reveal any additional information. Furthermore, all experiments are based on the data from all 13 languages together, as explained in section 3.

4.1 Length Factors

It is well known that parsing systems tend to have lower accuracies for longer sentences. Figure 2 shows the accuracy of both parsing models relative to sentence length (in bins of size 10: 1–10, 11–20,

[2] http://w3.msi.vxu.se/users/nivre/research/MaltParser.html

[3] In addition, results were reported for *unlabeled attachment score* (UAS) (tokens with the correct head) and *label accuracy* (LA) (tokens with the correct label).

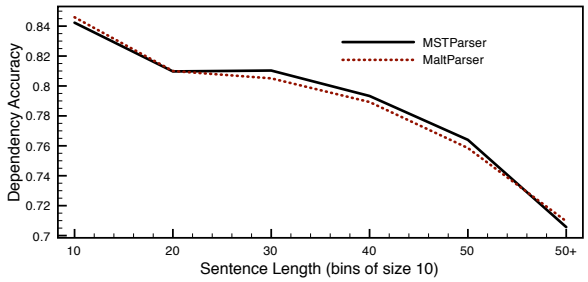


Figure 2: Accuracy relative to sentence length.

etc.). System performance is almost indistinguishable. However, MaltParser tends to perform better on shorter sentences, which require the greedy inference algorithm to make less parsing decisions. As a result, the chance of error propagation is reduced significantly when parsing these sentences. The fact that MaltParser has a higher accuracy (rather than the same accuracy) when the likelihood of error propagation is reduced comes from its richer feature representation.

Another interesting property is accuracy relative to dependency length. The length of a dependency from word w_i to word w_j is simply equal to $|i - j|$. Longer dependencies typically represent modifiers of the root or the main verb in a sentence. Shorter dependencies are often modifiers of nouns such as determiners or adjectives or pronouns modifying their direct neighbours. Figure 3 measures the precision and recall for each system relative to dependency lengths in the predicted and gold standard dependency graphs. Precision represents the percentage of predicted arcs of length d that were correct. Recall measures the percentage of gold standard arcs of length d that were correctly predicted.

Here we begin to see separation between the two systems. MSTParser is far more precise for longer dependency arcs, whereas MaltParser does better for shorter dependency arcs. This behaviour can be explained using the same reasoning as above: shorter arcs are created before longer arcs in the greedy parsing procedure of MaltParser and are less prone to error propagation. Theoretically, MSTParser should not perform better or worse for edges of any length, which appears to be the case. There is still a slight degradation, but this can be attributed to long dependencies occurring more frequently in constructions with possible ambiguity. Note that even though the area under the curve is much larger for MSTParser, the number of dependency arcs with a length greater than ten is much smaller than the number with length less than ten, which is why the overall accuracy of each system is nearly identical. For all properties considered here, bin size generally shrinks in size as the value on the x-axis increases.

4.2 Graph Factors

The structure of the predicted and gold standard dependency graphs can also provide insight into the differences between each model. For example, measuring accuracy for arcs relative to their distance to the artificial root node will detail errors at different levels of the dependency graph. For a given arc, we define this distance as the number of arcs in the reverse path from the modifier of the arc to the root. Figure 4 plots the precision and recall of each system for arcs of varying distance to the root. Precision is equal to the percentage of dependency arcs in the predicted graph that are at a distance of d and are correct. Recall is the percentage of dependency arcs in the gold standard graph that are at a distance of d and were predicted.

Figure 4 clearly shows that for arcs close to the root, MSTParser is much more precise than MaltParser, and vice-versa for arcs further away from the root. This is probably the most compelling graph given in this study since it reveals a clear distinction: MSTParser's precision degrades as the distance to the root increases whereas MaltParser's precision increases. The plots essentially run in opposite directions crossing near the middle. Dependency arcs further away from the root are usually constructed early in the parsing algorithm of MaltParser. Again a reduced likelihood of error propagation coupled with a rich feature representation benefits that parser substantially. Furthermore, MaltParser tends to overpredict root modifiers, because all words that the parser fails to attach as modifiers are automatically connected to the root, as explained in section 2.3. Hence, low precision for root modifiers (without a corresponding drop in recall) is an indication that the transition-based parser produces fragmented parses.

The behaviour of MSTParser is a little trickier to explain. One would expect that its errors should be distributed evenly over the graph. For the most part this is true, with the exception of spikes at the ends

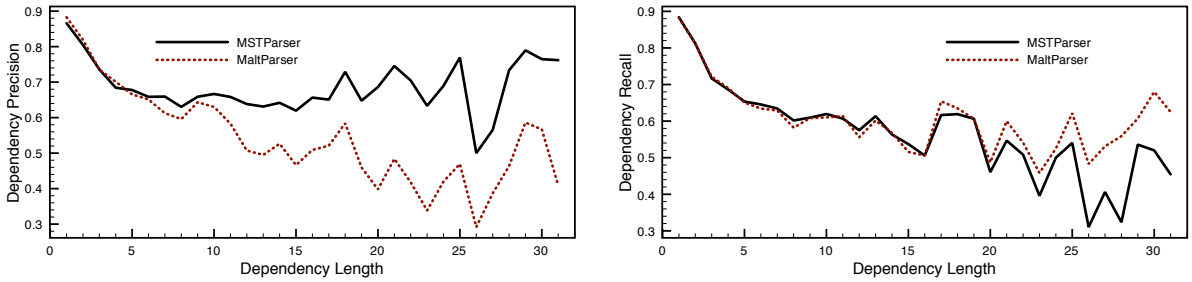


Figure 3: Dependency arc precision/recall relative to predicted/gold dependency length.

of the plot. The high performance for root modification (distance of 1) can be explained through the fact that this is typically a low entropy decision – usually the parsing algorithm has to determine the main verb from a small set of possibilities. On the other end of the plot there is a sharp downwards spike for arcs of distance greater than 10. It turns out that MSTParser over-predicts arcs near the bottom of the graph. Whereas MaltParser pushes difficult parsing decisions higher in the graph, MSTParser appears to push these decisions lower.

The next graph property we will examine aims to quantify the local neighbourhood of an arc within a dependency graph. Two dependency arcs, (i, j, l) and (i', j', l') are classified as siblings if they represent syntactic modifications of the same word, i.e., $i = i'$. Figure 5 measures the precision and recall of each system relative to the number of predicted and gold standard siblings of each arc. There is not much to distinguish between the parsers on this metric. MSTParser is slightly more precise for arcs that are predicted with more siblings, whereas MaltParser has slightly higher recall on arcs that have more siblings in the gold standard tree. Arcs closer to the root tend to have more siblings, which ties this result to the previous ones.

The final graph property we wish to look at is the degree of non-projectivity. The degree of a dependency arc from word w to word u is defined here as the number of words occurring between w and u that are not descendants of w and modify a word that does not occur between w and u (Nivre, 2006). In the example from Figure 1, the arc from *jedna* to *Z* has a degree of one, and all other arcs have a degree of zero. Figure 6 plots dependency arc precision and recall relative to arc degree in predicted and gold standard dependency graphs. MSTParser is more precise when predicting arcs with high degree and MaltParser vice-versa. Again, this can be explained by the fact that there is a tight correlation between a high degree of non-projectivity, dependency length, distance to root and number of siblings.

4.3 Linguistic Factors

It is important to relate each system's accuracy to a set of linguistic categories, such as parts of speech and dependency types. Therefore, we have made an attempt to distinguish a few broad categories that are cross-linguistically identifiable, based on the available documentation of the treebanks used in the shared task.

For parts of speech, we distinguish *verbs* (including both main verbs and auxiliaries), *nouns* (including proper names), *pronouns* (sometimes also including determiners), *adjectives*, *adverbs*, *adpositions* (prepositions, postpositions), and *conjunctions* (both coordinating and subordinating). For dependency types, we distinguish a general *root* category (for labels used on arcs from the artificial root, including either a generic label or the label assigned to predicates of main clauses, which are normally verbs), a *subject* category, an *object* category (including both direct and indirect objects), and various categories related to *coordination*.

Figure 7 shows the accuracy of the two parsers for different parts of speech. This figure measures labeled dependency accuracy relative to the part of speech of the modifier word in a dependency relation. We see that MaltParser has slightly better accuracy for nouns and pronouns, while MSTParser does better on all other categories, in particular conjunctions. This pattern is consistent with previous results insofar as verbs and conjunctions are often involved in dependencies closer to the root that span

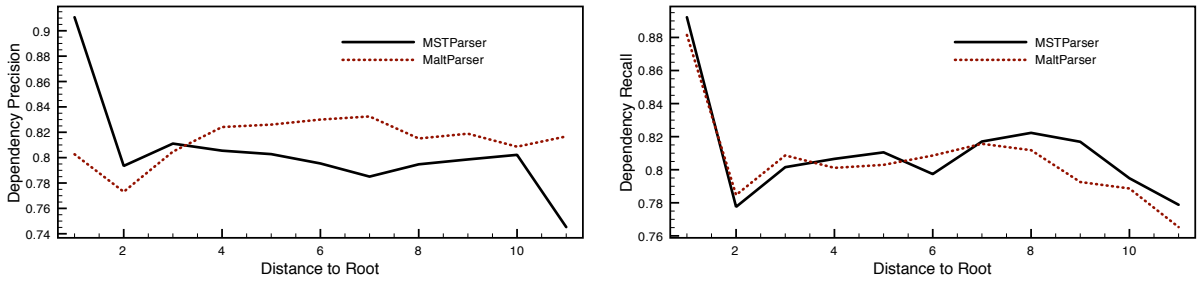


Figure 4: Dependency arc precision/recall relative to predicted/gold distance to root.

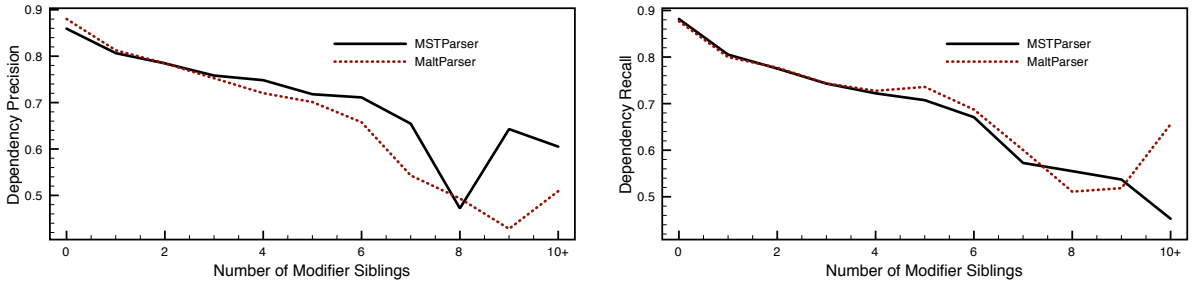


Figure 5: Dependency arc precision/recall relative to number of predicted/gold siblings.

longer distances, while nouns and pronouns are typically attached to verbs and therefore occur lower in the graph, with shorter distances. Empirically, adverbs resemble verbs and conjunctions with respect to root distance but group with nouns and pronouns for dependency length, so the former appears to be more important. In addition, both conjunctions and adverbs tend to have a high number of siblings, making the results consistent with the graph in Figure 5.

Adpositions and especially adjectives constitute a puzzle, having both high average root distance and low average dependency length. Adpositions do tend to have a high number of siblings on average, which could explain MSTParser's performance on that category. However, adjectives on average occur the furthest away from the root, have the shortest dependency length and the fewest siblings. As such, we do not have an explanation for this behaviour.

In the top half of Figure 8, we consider precision and recall for dependents of the root node (mostly verbal predicates), and for subjects and objects. As already noted, MSTParser has considerably better precision (and slightly better recall) for the root category, but MaltParser has an advantage for the nominal categories, especially subjects. A possible explanation for the latter result, in addition to the length-based and graph-based factors invoked before, is that

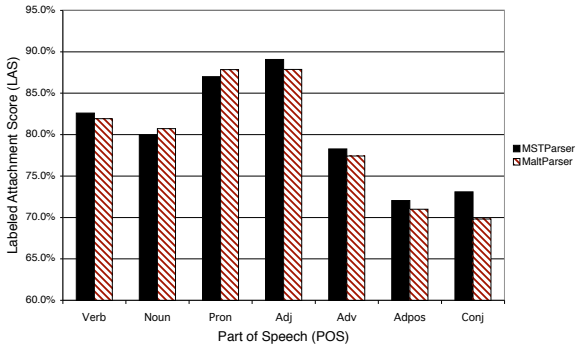


Figure 7: Accuracy for different parts of speech.

MaltParser integrates labeling into the parsing process, so that previously assigned dependency labels can be used as features, which may be important to disambiguate subjects and objects.

Finally, in the bottom half of Figure 8, we display precision and recall for coordinate structures, divided into different groups depending on the type of analysis adopted in a particular treebank. The category CCH (coordinating conjunction as head) contains conjunctions analyzed as heads of coordinate structures, with a special dependency label that does not describe the function of the coordinate structure in the larger syntactic structure, a type of category found in the so-called Prague style analysis of coordination and used in the data sets for Arabic, Czech,

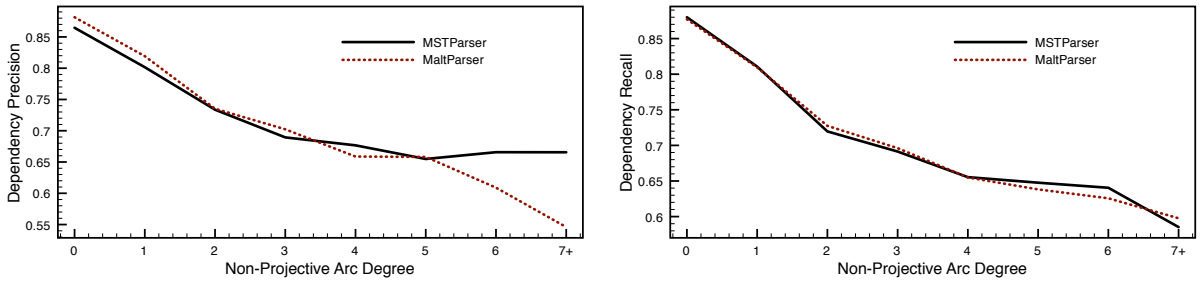


Figure 6: Dependency arc precision/recall relative to predicted/gold degree of non-projectivity.

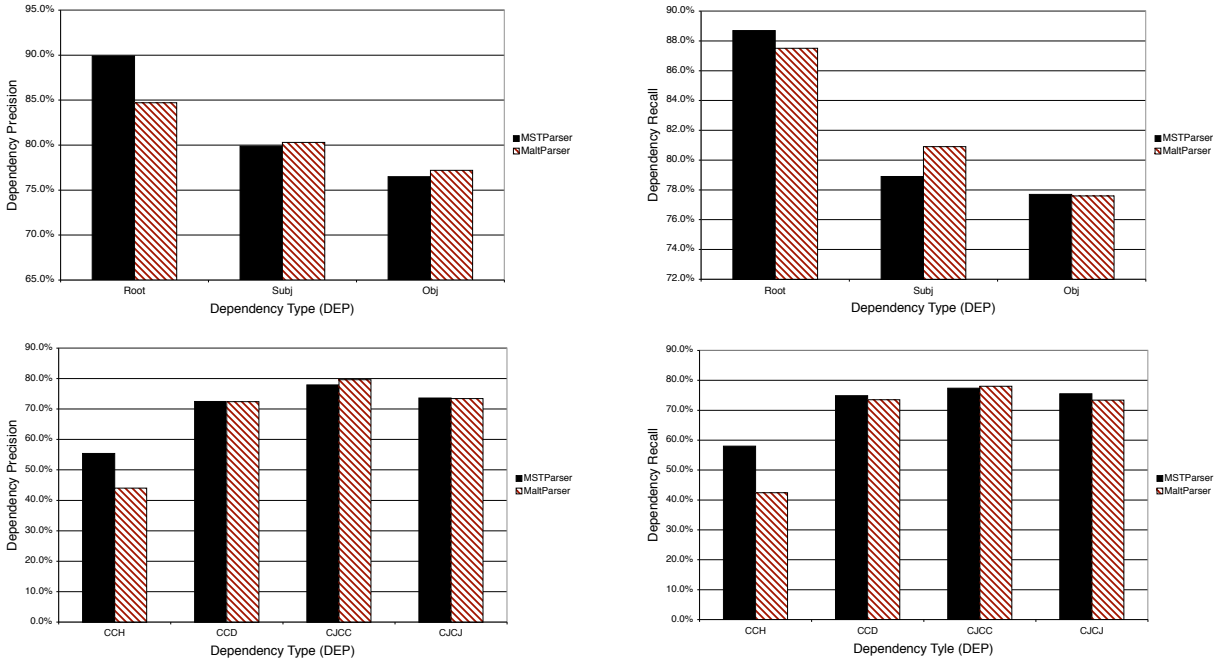


Figure 8: Precision/recall for different dependency types.

and Slovene. The category CCD (coordinating conjunction as dependent) instead denotes conjunctions that are attached as dependents of one of the conjuncts with a label that only marks them as conjunctions, a type of category found in the data sets for Bulgarian, Danish, German, Portuguese, Swedish and Turkish. The two remaining categories contain conjuncts that are assigned a dependency label that only marks them as conjuncts and that are attached either to the conjunction (CJCC) or to another conjunct (CJCJ). The former is found in Bulgarian, Danish, and German; the latter only in Portuguese and Swedish. For most of the coordination categories there is little or no difference between the two parsers, but for CCH there is a difference in both precision and recall of almost 20 percentage points to MSTParser's advantage. This can be explained by noting that, while the categories CCD, CJCC, and CJCJ denote relations that are *internal* to the coordinate structure and therefore tend to be local, the CCH relations hold between the coordinate structure and its head, which is often a relation that spans over a greater distance and is nearer the root of the dependency graph. It is likely that the difference in accuracy for this type of dependency accounts for a large part of the difference in accuracy noted earlier for conjunctions as a part of speech.

4.4 Discussion

The experiments from the previous section highlight the fundamental trade-off between global training and exhaustive inference on the one hand and expressive feature representations on the other. Error propagation is an issue for MaltParser, which typi-

cally performs worse on long sentences, long dependency arcs and arcs higher in the graphs. But this is offset by the rich feature representation available to these models that result in better decisions for frequently occurring arc types like short dependencies or subjects and objects. The errors for MSTParser are spread a little more evenly. This is expected, as the inference algorithm and feature representation should not prefer one type of arc over another.

What has been learned? It was already known that the two systems make different errors through the work of Sagae and Lavie (2006). However, in that work an arc-based voting scheme was used that took only limited account of the properties of the words connected by a dependency arc (more precisely, the overall accuracy of each parser for the part of speech of the dependent). The analysis in this work not only shows that the errors made by each system are different, but that they are different in a way that can be predicted and quantified. This is an important step in parser development.

To get some upper bounds of the improvement that can be obtained by combining the strengths of each models, we have performed two oracle experiments. Given the output of the two systems, we can envision an oracle that can optimally choose which single parse or combination of sub-parses to predict as a final parse. For the first experiment the oracle is provided with the single best parse from each system, say $G = (V, A)$ and $G' = (V', A')$. The oracle chooses a parse that has the highest number of correctly predicted labeled dependency attachments. In this situation, the oracle accuracy is 84.5%. In the second experiment the oracle chooses the tree that maximizes the number of correctly predicted dependency attachments, subject to the restriction that the tree must only contain arcs from $A \cup A'$. This can be computed by setting the weight of an arc to 1 if it is in the correct parse and in the set $A \cup A'$. All other arc weights are set to negative infinity. One can then simply find the tree that has maximal sum of arc weights using directed spanning tree algorithms. This technique is similar to the parser voting methods used by Sagae and Lavie (2006). In this situation, the oracle accuracy is 86.9%.

In both cases we see a clear increase in accuracy: 86.9% and 84.5% relative to 81% for the individual systems. This indicates that there is still potential for improvement, just by combining the two existing models. More interestingly, however, we can use the analysis to get ideas for new models. Below we sketch some possible new directions:

1. **Ensemble systems:** The error analysis presented in this paper could be used as inspiration for more refined weighting schemes for ensemble systems of the kind proposed by Sagae and Lavie (2006), making the weights depend on a range of linguistic and graph-based factors.

2. **Hybrid systems:** Rather than using an ensemble of several parsers, we may construct a single system integrating the strengths of each parser described here. This could defer to a greedy inference strategy during the early stages of the parse in order to benefit from a rich feature representation, but then default to a global exhaustive model as the likelihood for error propagation increases.

3. **Novel approaches:** The two approaches investigated are each based on a particular combination of training and inference methods. We may naturally ask what other combinations may prove fruitful. For example, what about globally trained, greedy, transition-based models? This is essentially what Daumé III et al. (2006) provide, in the form of a general search-based structured learning framework that can be directly applied to dependency parsing. The advantage of this method is that the learning can set model parameters relative to errors resulting directly from the search strategy – such as error propagation due to greedy search. When combined with MaltParser's rich feature representation, this could lead to significant improvements in performance.

5 Conclusion

We have presented a thorough study of the difference in errors made between global exhaustive graph-based parsing systems (MSTParser) and local greedy transition-based parsing systems (MaltParser). We have shown that these differences can be quantified and tied to theoretical expectations of each model, which may provide insights leading to better models in the future.

References

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2003. The PDT: A 3-level annotation scenario. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, chapter 7. Kluwer Academic Publishers.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. CoNLL*.

Hal Daumé III, John Langford, and Daniel Marcu. 2006. Search-based structured prediction. In Submission.

Y. Ding and M. Palmer. 2004. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In *Workshop on Recent Advances in Dependency Grammars (COLING)*.

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

H. Maruyama. 1990. Structural disambiguation with constraint propagation. In *Proc. ACL*.

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

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. HLT/EMNLP*.

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proc. CoNLL*.

I.A. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

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

J. Nivre, J. Hall, J. Nilsson, G. Eryigit, and S. Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proc. CoNLL*.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. IWPT*.

J. Nivre. 2006. Constraints on non-projective dependency parsing. In *Proc. EACL*.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *Proc. HLT/NAACL*.

P. Sgall, E. Hajičová, and J. Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

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