Analyzing and Integrating Dependency Parsers

Ryan McDonald*
Google Inc.

Joakim Nivre**
Uppsala University

There has been a rapid increase in the volume of research on data-driven dependency parsers in the past five years. This increase has been driven by the availability of treebanks in a wide variety of languages—due in large part to the CoNLL shared tasks—as well as the straightforward mechanisms by which dependency theories of syntax can encode complex phenomena in free word order languages. In this article, our aim is to take a step back and analyze the progress that has been made through an analysis of the two predominant paradigms for data-driven dependency parsing, which are often called graph-based and transition-based dependency parsing. Our analysis covers both theoretical and empirical aspects and sheds light on the kinds of errors each type of parser makes and how they relate to theoretical expectations. Using these observations, we present an integrated system based on a stacking learning framework and show that such a system can learn to overcome the shortcomings of each non-integrated system.

1. Introduction

Syntactic dependency representations have a long history in descriptive and theoretical linguistics and many formal models have been advanced, most notably Word Grammar (Hudson 1984), Meaning-Text Theory (Mel'čuk 1988), Functional Generative Description (Sgall, Hajičová, and Panevová 1986), and Constraint Dependency Grammar (Maruyama 1990). Common to all theories is the notion of directed syntactic dependencies between the words of a sentence, an example of which is given in Figure 1 for the sentence *A hearing is scheduled on the issue today*, which has been extracted from the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993). A dependency graph of a sentence represents each word and its syntactic modifiers through labeled directed arcs, where each arc label comes from some finite set representing possible syntactic roles. Returning to our example in Figure 1, we can see multiple instances of labeled dependency relations such as the one from the finite verb *is* to *hearing* labeled *SBJ* indicating that *hearing* is the head of the syntactic subject of the finite verb. An artificial word has been inserted at the beginning of the sentence that will always serve as the single root of the graph and is primarily a means to simplify computation.

* 76 Ninth Ave., New York, NY 10011. E-mail: ryanmcd@google.com.
** Department of Linguistics and Philology, Box 635, SE-75126 Uppsala, Sweden. E-mail: joakim.nivre@lingfil.uu.se.

Submission received: 25 August 2009; revised submission received: 20 August 2010; accepted for publication: 7 October 2010.

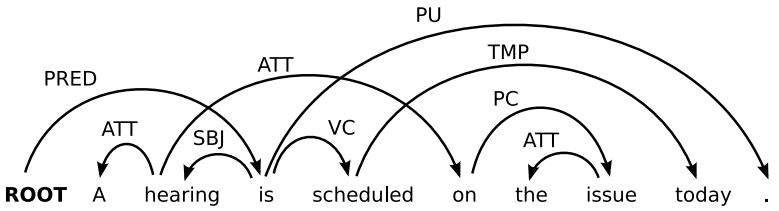

Figure 1
Dependency graph for an English sentence.

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, Jurafsky, and Ng 2005). A primary advantage of dependency representations is that they have a natural mechanism for representing discontinuous constructions, which arise due to long-distance dependencies or in languages where grammatical relations are often signaled by morphology instead of word order. This is undoubtedly one of the reasons for the emergence of dependency parsers for a wide range of languages (Buchholz and Marsi 2006; Nivre et al. 2007). Thus, the example in Figure 1 contains an instance of a discontinuous construction through the subgraph rooted at the word *hearing*. Specifically, the dependency arc from *hearing* to *on* spans the words *is* and *scheduled*, which are not nodes in this subgraph. An arc of this kind is said to be **non-projective**.

In this article we focus on a common paradigm called data-driven dependency parsing, which encompasses parsing systems that learn to produce dependency graphs for sentences from a corpus of sentences annotated with dependency graphs. The advantage of such models is that they are easily ported to any domain or language in which annotated resources exist. Many data-driven parsing systems are grammarless, in that they do not assume the existence of a grammar that defines permissible sentences of the language. Instead, the goal of most data-driven parsing systems is to discriminate good parses from bad for a given sentence, regardless of its grammaticality. Alternatively, one can view such systems as parsers for a grammar that induces the language of all strings.

The rise of statistical methods in natural language processing coupled with the availability of dependency annotated corpora for multiple languages—most notably from the 2006 and 2007 CoNLL shared tasks (Buchholz and Marsi 2006; Nivre et al. 2007)—has led to a boom in research on data-driven dependency parsing. Making sense of this work is a challenging problem, but an important one if the field is to continue to make advances. Of the many important questions to be asked, three are perhaps most crucial at this stage in the development of parsers:

1. How can we formally categorize the different approaches to data-driven dependency parsing?
2. Can we characterize the kinds of errors each category of parser makes through an empirical analysis?
3. Can we benefit from such an error analysis and build improved parsers?

The organizers of the CoNLL-X shared task on dependency parsing (Buchholz and Marsi 2006) point out that there are currently two dominant approaches for data-driven

dependency parsing. The first category parameterizes models over dependency subgraphs and learns these parameters to globally score correct graphs above incorrect ones. Inference is also global, in that systems attempt to find the highest scoring graph among the set of all graphs. We call such systems **graph-based** parsing models to reflect the fact that parameterization is over the graph. Graph-based models are mainly associated with the pioneering work of Eisner (Eisner 1996), as well as McDonald and colleagues (McDonald, Crammer, and Pereira 2005; McDonald et al. 2005; McDonald and Pereira 2006; McDonald, Lerman, and Pereira 2006) and others (Riedel, Çakıcı, and Meza-Ruiz 2006; Carreras 2007; Koo et al. 2007; Nakagawa 2007; Smith and Smith 2007). The second category of parsing systems parameterizes models over transitions from one state to another in an abstract state-machine. Parameters in these models are typically learned using standard classification techniques that learn to predict one transition from a set of permissible transitions given a state history. Inference is local, in that systems start in a fixed initial state and greedily construct the graph by taking the highest scoring transitions at each state entered until a termination condition is met. We call such systems **transition-based** parsing models to reflect the fact that parameterization is over possible state transitions. Transition-based models have been promoted by the groups of Matsumoto (Kudo and Matsumoto 2002; Yamada and Matsumoto 2003; Cheng, Asahara, and Matsumoto 2006), Nivre (Nivre, Hall, and Nilsson 2004; Nivre and Nilsson 2005; Nivre et al. 2006), and others (Attardi 2006; Attardi and Ciaramita 2007; Johansson and Nugues 2007; Duan, Zhao, and Xu 2007; Titov and Henderson 2007a, 2007b).

It is important to note that there is no a priori reason why a graph-based parameterization should require global learning and inference, and a transition-based parameterization would necessitate local learning and greedy inference. Nevertheless, as observed by Buchholz and Marsi (2006), it is striking that recent work on data-driven dependency parsing has been dominated by *global, exhaustive, graph-based models*, on the one hand, and *local, greedy, transition-based models*, on the other. Therefore, a careful comparative analysis of these model types appears highly relevant, and this is what we will try to provide in this article. For convenience, we will use the shorthand terms "graph-based" and "transition-based" for these models, although both graph-based and transition-based parameterizations can be (and have been) combined with different types of learning and inference. For example, the system described by Zhang and Clark (2008) could be characterized as a transition-based model with global learning, and the ensemble system of Zeman and Žabokrtskỳ (2005) as a graph-based model with greedy inference.

Perhaps the most interesting reason to study the canonical graph-based and transition-based models is that even though they appear to be quite different theoretically (see Section 2), recent empirical studies show that both obtain similar parsing accuracies on a variety of languages. For example, Table 1 shows the results of the two top performing systems in the CoNLL-X shared task, those of McDonald, Lerman, and Pereira (2006) (graph-based) and Nivre et al. (2006) (transition-based), which exhibit no statistically significant difference in accuracy when averaged across all languages. This naturally leads us to our Question 2, that is, can we empirically characterize the errors of these systems to understand whether, in practice, these errors are the same or distinct? Towards this end, Section 2 describes in detail the theoretical properties and expectations of these two parsing systems and Section 4 provides a fine-grained error analysis of each system on the CoNLL-X shared task data sets (Buchholz and Marsi 2006). The result of this analysis strongly suggests that (1) the two systems do make different, yet complementary, errors, which lends support to the categorization of

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

Language	Graph-based (McDonald, Lerman, and Pereira 2006)	Transition-based (Nivre et al. 2006)
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
Average	80.83	80.75

parsers as graph-based and transition-based, and (2) the errors made by each system are directly correlated with our expectations, based on their theoretical underpinnings.

This leads to our Question 3: Can we use these insights to integrate parsers and achieve improved accuracies? In Section 5 we consider a simple way of integrating graph-based and transition-based models in order to exploit their complementary strengths and thereby improve parsing accuracy beyond what is possible by either model in isolation. The method integrates the two models by allowing the output of one model to define features for the other, which is commonly called "classifier stacking." This method is simple—requiring only the definition of new features—and robust by allowing a model to learn relative to the predictions of the other. More importantly, we rerun the error analysis and show that the integrated models do indeed take advantage of the complementary strengths of both the graph-based and transition-based parsing systems.

Combining the strengths of different machine learning systems, and even parsing systems, is by no means new as there are a number of previous studies that have looked at combining phrase-structure parsers (Henderson and Brill 1999), dependency parsers (Zeman and Žabokrtskỳ 2005), or both (McDonald 2006). Of particular note is past work on combining graph-based and transition-based dependency parsers. Sagae and Lavie (2006) present a system that combines multiple transition-based parsers with a single graph-based parser by weighting each potential dependency relation by the number of parsers that predicted it. A final dependency graph is predicted by using spanning tree inference algorithms from the graph-based parsing literature (McDonald et al. 2005). Sagae and Lavie report improvements of up to 1.7 percentage points over the best single parser when combining three transition-based models and one graph-based model for unlabeled dependency parsing, evaluated on data from the Penn Treebank. The same technique was used by Hall et al. (2007) to combine six transition-based parsers in the best performing system in the CoNLL 2007 shared task.

Zhang and Clark (2008) propose a parsing system that uses global learning coupled with beam search over a transition-based backbone incorporating both graph-based

and transition-based features, that is, features over both sub-graphs and transitions. Huang and Sagae (2010) go even further and show how transition-based parsing can be tabularized to allow for dynamic programming, which in turn permits an exponentially larger search space. Martins et al. (2008) present a method for integrating graph-based and transition-based parsers based on stacking, which is similar to the approach taken in this work. Other studies have tried to overcome the weaknesses of parsing models by changing the underlying model structure directly. For example, Hall (2007), Riedel, Çakıcı, and Meza-Ruiz (2006), Nakagawa (2007), Smith and Eisner (2008), and Martins, Smith, and Xing (2009) attempt to overcome local restrictions in feature scope for graph-based parsers through both approximations and exact solutions with integer linear programming.

Our work differs from past studies in that we attempt to quantify exactly the types of errors these parsers make, tie them to their theoretical expectations, and show that integrating graph-based and transition-based parsers not only increases overall accuracy, but does so directly exploiting the strengths of each system. Thus, this is the first large-scale error analysis of modern data-driven dependency parsers.[1] The rest of the article is structured as follows: Section 2 describes canonical graph-based and transition-based parsing systems and discusses their theoretical benefits and limitations with respect to one another; Section 3 introduces the experimental setup based on the CoNLL-X shared task data sets that incorporate dependency treebanks from 13 diverse languages; Section 4 gives a fine-grained error analysis for the two parsers in this setup; Section 5 describes a stacking-based dependency parser combination framework; Section 6 evaluates the stacking-based parsers in comparison to the original systems with a detailed error analysis; we conclude in Section 7.

2. Two Models for Dependency Parsing

In this section we introduce central notation and define canonical graph-based and transition-based dependency parsing at an abstract level. We further compare and contrast their theoretical underpinnings with an eye to understanding the kinds of errors each system is likely to make in practice.

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 $w_0 = \text{ROOT}$. 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 $A \subseteq V \times V \times L$; that is, 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. In terms of standard linguistic dependency theory nomenclature, we say that $(i, j, l) \in A$ if there is a dependency with **head** w_i, **dependent** w_j, and **syntactic role** l.

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 $j \neq 0$.

1 This work has previously been published partially in McDonald and Nivre (2007) and Nivre and McDonald (2008).

3. If $(i, j, l) \in A$, then for all arcs $(i', j, l') \in A$, $i = i'$ and $l = l'$.
4. For all $j \in V - \{0\}$, either $(0, j, l)$ for some $l \in L$ or there is a non-empty sequence of nodes $i_1, \ldots, i_m \in V$ and labels $l_1, \ldots, l_{m+1} \in L$ such that $(0, i_1, l_1), (i_1, i_2, l_2), \ldots, (i_m, j, l_{m+1}) \in A$.

The first constraint states that the dependency graph spans the entire input. The second constraint states that the node 0 is a root. The third constraint states that each node has at most one incoming arc in the graph. The final constraint states that the graph is connected through directed paths from the node 0 to every other node in the graph. It is not difficult to show that a dependency graph satisfying these constraints is in fact a directed tree originating out of the root node 0 and we will use the term **dependency tree** to refer to any valid dependency graph. The characterization of syntactic dependency graphs as trees is consistent with most formal theories (e.g., Sgall, Hajičová, and Panevová 1986; Mel'čuk 1988). Exceptions include Word Grammar (Hudson 1984), which allows a word to modify multiple other words in the sentence, which results in directed acyclic graphs with nodes possibly having multiple incoming arcs.

We define an arc (i, j, l) connecting words w_i and w_j as **non-projective** if at least one word occurring between w_i and w_j in the input sentence is not a descendant of w_i (where "descendant" is the transitive closure of the arc relation). Alternatively, we can view non-projectivity in trees as breaking the nested property, which can be seen through the arcs that cross in the example in Figure 1. Non-projective dependencies are typically difficult to represent or parse in phrase-based models of syntax. This can either be due to nested restrictions arising in context-free formalisms or computationally expensive operations in mildly context-sensitive formalisms (e.g., adjunction in TAG frameworks).

2.2 Global, Exhaustive, Graph-Based Parsing

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

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

Let $D(G_x)$ represent the subgraphs of graph G_x that are valid dependency graphs for the sentence x, that is, dependency trees. Because G_x contains all possible labeled arcs, the set $D(G_x)$ must necessarily contain all dependency trees 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 an arc, $s(i, j, l)$ represents the likelihood of creating a dependency from head w_i to modifier w_j with the label l in a dependency tree. This score is commonly defined to be the product of a high dimensional feature representation of the arc and a

learned parameter vector, $s(i,j,l) = \mathbf{w} \cdot \mathbf{f}(i,j,l)$. If the arc score function is known, then the parsing problem can be stated as

$$G^* = \arg\max_{G \in D(G_x)} s(G) = \arg\max_{G \in D(G_x)} \sum_{(i,j,l) \in A} s(i,j,l) \quad (1)$$

An example graph G_x and the dependency tree maximizing the scoring function are given in Figure 2 for the sentence *John saw Mary*. We omit arcs into the root node for simplicity.

McDonald et al. (2005) showed that this problem is equivalent to finding the highest scoring directed spanning tree for the graph G_x originating out of the root node 0. It is not difficult to see this, because both dependency trees and spanning trees must contain all nodes of the graph and must have a tree structure with root 0. The directed spanning tree problem (also known as the r-arborescence problem) can be solved for both the labeled and unlabeled case using the Chu-Liu-Edmonds algorithm (Chu and Liu 1965; Edmonds 1967), a variant of which can be shown to have an $O(n^2)$ runtime (Tarjan 1977). Non-projective arcs are produced naturally through the inference algorithm that searches over all possible directed trees, whether projective or not.

Graph-based parsers are typically trained using structured learning algorithms (McDonald, Crammer, and Pereira 2005; Koo et al. 2007; Smith and Smith 2007), which optimize the parameters of the model to maximize the difference in score/probability between the correct dependency graph and all incorrect dependency graphs for every sentence in a training set. Such a learning procedure is global because model parameters are set relative to the classification of the entire dependency graph, and not just over single arc attachment decisions. Although a learning procedure that only optimizes the score of individual arcs is conceivable, it would not be likely to produce competitive results.

Going beyond arc-factored models, McDonald and Pereira (2006) presented a system where scores are increased in scope to include pairs of adjacent arcs in the dependency graph. In the case of projective dependency trees, polynomial time parsing algorithms were shown to exist, but non-projective trees required approximate inference that used an exhaustive projective algorithm followed by transformations to the graph that incrementally introduce non-projectivity. In general, inference and learning for graph-based dependency parsing is NP-hard when the score is factored

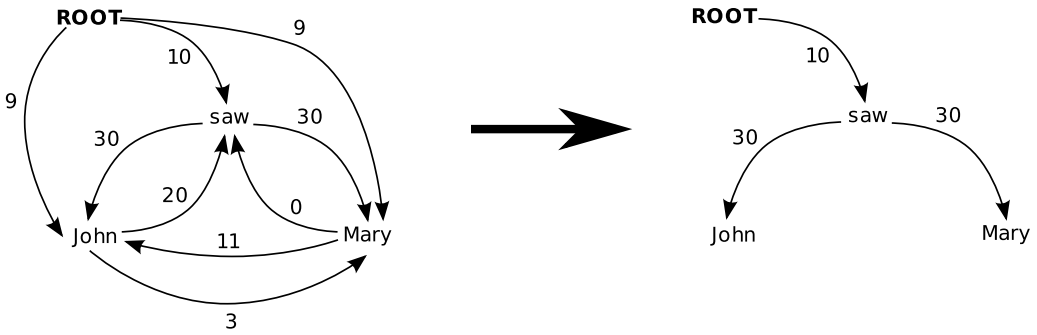

Figure 2
A graph-based parsing example. A dense graph G_x is shown on the left (arcs into the root are omitted) with corresponding arc scores. On the right is the predicted dependency tree based on Equation (1).

over anything larger than arcs (McDonald and Satta 2007). Thus, graph-based parsing systems cannot easily condition on any extended scope of the dependency graph beyond a single arc, which is their primary shortcoming relative to transition-based systems. McDonald, Crammer, and Pereira (2005) show that a rich feature set over the input space, including lexical and surface syntactic features of neighboring words, can partially alleviate this problem, and both Carreras (2007) and Koo et al. (2010) explore higher-order models for projective trees. Additionally, work has been done on approximate non-factored parsing systems (McDonald and Pereira 2006; Hall 2007; Nakagawa 2007; Smith and Eisner 2008) as well as exact solutions through integer linear programming (Riedel, Çakıcı, and Meza-Ruiz 2006; Martins, Smith, and Xing 2009).

The specific graph-based system studied in this work is that presented by McDonald, Lerman, and Pereira (2006), which uses pairwise arc scoring and approximate exhaustive search for unlabeled parsing. A separate arc label classifier is then used to label each arc. This two-stage process was adopted primarily for computational reasons and often does not affect performance significantly (see McDonald [2006] for more). Throughout the rest of this study we will refer to this system as MSTParser (or MST for short), which is also the name of the freely available implementation.[2]

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 of which is a partial 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** $C_{0,m} = (c_0, c_1, \ldots, c_m)$ for a sentence $x = w_0, w_1, \ldots, w_n$ is a sequence of configurations such that $c_0 = c_x$, $c_m \in C_x$, and, for every c_i $(1 \leq i \leq m)$, there is a transition $t \in T$ such that $c_i = t(c_{i-1})$. The dependency graph assigned to a sentence $x = w_0, w_1, \ldots, w_n$ by a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ 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 in a transition sequence leading to the optimal dependency graph for the given sentence. This score is usually defined by a classifier g taking as input a high dimensional feature representation of the configuration, $s(c, t) = g(\mathbf{f}(c), t)$.

Given a transition scoring function, 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^* out of every configuration c:

$$t^* = \arg\max_{t \in T} s(c, t)$$

2 `http://mstparser.sourceforge.net`.

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.

By way of example, we consider the transition system first presented in Nivre (2003), where a parser configuration is a triple $c = (\sigma, \beta, A)$, consisting of a stack σ of partially processed nodes, a buffer β of remaining input nodes, and a set A of labeled dependency arcs. The initial configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is $c_x = ([0], [1, \ldots, n], \emptyset)$ and the set of terminal configurations C_x contains all configurations of the form $c = (\sigma, [\,], A)$ (that is, all configurations with an empty buffer and with arbitrary σ and A). The set T of transitions for this system is specified in Figure 3. The transitions LEFT-ARC$_l$ and RIGHT-ARC$_l$ extend the arc set A with an arc (labeled l) connecting the top node i on the stack and the first node j of the buffer. In the case of LEFT-ARC$_l$, the node i becomes the dependent and is also popped from the stack; in the case of RIGHT-ARC$_l$, the node j becomes the dependent and is also pushed onto the stack. The REDUCE transition pops the stack (and presupposes that the top node has already been attached to its head in a previous RIGHT-ARC$_l$ transition), and the SHIFT transition extracts the first node of the buffer and pushes it onto the stack.

This system can derive any projective dependency tree G for an input sentence x and in doing so always adds arcs as early as possible. For this reason, the system is often referred to as **arc-eager**. When coupled with the greedy deterministic parsing strategy, the system guarantees termination after at most $2n$ transitions (for a sentence of length n), which means that the time complexity is $O(n)$ given that transitions can be performed in constant time. The dependency graph given at termination is guaranteed to be acyclic and projective and to satisfy dependency graph conditions 1–3, which means that it can always be turned into a well-formed dependency graph by 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). Whereas the initial formulation in Nivre (2003) was limited to unlabeled dependency graphs, the system was extended to labeled graphs in Nivre, Hall, and Nilsson (2004), and Nivre and Nilsson (2005) showed how the restriction to projective dependency graphs could be lifted by using graph transformation

Transitions

LEFT-ARC$_l$	$(\sigma \vert i, j \vert \beta, A) \Rightarrow (\sigma, j \vert \beta, A \cup \{(j, i, l)\})$
RIGHT-ARC$_l$	$(\sigma \vert i, j \vert \beta, A) \Rightarrow (\sigma \vert i \vert j, \beta, A \cup \{(i, j, l)\})$
REDUCE	$(\sigma \vert i, \beta, A) \Rightarrow (\sigma, \beta, A)$
SHIFT	$(\sigma, i \vert \beta, A) \Rightarrow (\sigma \vert i, \beta, A)$

Preconditions

LEFT-ARC$_l$	$\neg[i = 0]$ $\neg\exists k \exists l'[(k, i, l') \in A]$
RIGHT-ARC$_l$	$\neg\exists k \exists l'[(k, j, l') \in A]$
REDUCE	$\exists k \exists l[(k, i, l) \in A]$

Figure 3
Transitions for dependency parsing (with preconditions).

techniques to pre-process training data and post-process parser output, a technique called **pseudo-projective parsing**. Transition systems that derive non-projective trees directly have been explored by Attardi (2006) and Nivre (2007, 2009), among others.

To learn a scoring function, transition-based parsers use discriminative learning methods, such as memory-based learning or support vector machines. The training data are obtained by constructing transition sequences corresponding to gold standard parses from a treebank. The typical learning procedure is local because only single transitions are scored—not entire transition sequences—but more global optimization methods have also been proposed. The primary advantage of these models is that the feature representation is not restricted to a limited number of graph arcs but can take into account the entire dependency graph built so far, including previously assigned labels, and still support efficient inference and learning. The main disadvantage is that the greedy parsing strategy may lead to error propagation as false early predictions can eliminate valid trees due to structural constraints and are also used to create features when making future predictions. Using beam search instead of strictly deterministic parsing can to some extent alleviate this problem but does not eliminate it.

The specific transition-based system studied in this work is that presented by Nivre et al. (2006), which uses the projective, arc-eager system described here in combination with pseudo-projective parsing, which uses support vector machines to learn the scoring function for transitions and which uses greedy, deterministic one-best search at parsing time. We will refer to this system as MaltParser (or Malt for short), which is also the name of the freely available implementation.[3]

2.4 Comparison

In the previous two sections we have outlined the theoretical characteristics of canonical graph-based and transition-based dependency parsing systems. From now on, our experiments will rely on two standard implementations: MSTParser, a graph-based system, and MaltParser, a transition-based system. Here we contrast the two parsing systems with respect to how they are trained, how they produce dependency trees for new sentences, and what kinds of features they use.

Training Algorithms. Both systems use large-margin learning for linear classifiers. MSTParser uses on-line algorithms (McDonald, Crammer, and Pereira 2005; Crammer et al. 2006) and MaltParser uses support vector machines (Cortes and Vapnik 1995). The primary difference is that MaltParser trains the model to make a single classification decision (create arc, shift, reduce, etc.), whereas MSTParser trains the model to maximize the global score of correct graphs relative to incorrect graphs. It has been argued that locally trained algorithms can suffer from label bias issues (Lafferty, McCallum, and Pereira 2001). However, it is expensive to train global models since the complexity of learning is typically proportional to inference. In addition, MaltParser makes use of kernel functions, which eliminates the need for explicit conjunctions of features.

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 exhaustive search over a dense graphical representation of the

3 `http://w3.msi.vxu.se/users/nivre/research/MaltParser.html`.

sentence to find the dependency graph that maximizes the score. On the one hand, the greedy algorithm is far quicker computationally ($O(n)$ vs. $O(n^2)$ for the Chu-Liu-Edmonds algorithm and $O(n^3)$ for Eisner's algorithm). On the other hand, it may be prone to error propagation when early incorrect decisions negatively influence the parser at later stages. In particular, MaltParser uses the projective arc-eager transition system first described in Nivre (2003), which has consequences for the form of error propagation we may expect to see because the system determines the order in which arcs must be added to the graph. On the one hand, if an arc (i, m, l) covers another arc (j, k, l') (i.e., $i \leq j$ and $k \leq m$), then the smaller arc (j, k, l') has to be added to the graph first (because of projectivity). On the other hand, if two arcs (i, j, l) and (k, m, l') do not overlap, then the leftmost arc has to be added first (because of arc-eagerness). Therefore, we can expect error propagation from shorter to longer overlapping arcs and from preceding to succeeding arcs.

Feature Representation. Due to the nature of their inference and training algorithms, the feature representations of the two systems differ substantially. MaltParser can introduce a rich feature space based on the history of previous parser decisions. This is because the greedy nature of the algorithm allows it to fix the structure of the graph and use this structure to help improve future parsing decisions. By contrast, MSTParser is forced to restrict the scope of features to a single or pair of nearby parsing decisions in order to make exhaustive inference tractable. As a result, the feature representation available to the locally trained greedy models is much richer than the globally trained exhaustive models. Concisely, we can characterize MSTParser as using global training and inference with local features and MaltParser as using local training and inference with global features. (For more information about the features used in the two systems, see Sections 3.2 and 3.3.)

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. When analyzing, and ultimately explaining, the empirical difference between the systems, understanding this trade-off will be of central importance.

3. Experimental Setup

The experiments presented in this article are all based on data from the CoNLL-X shared task (Buchholz and Marsi 2006). In this section we first describe the task and the resources created there and then describe how MSTParser and MaltParser were trained for the task, including feature representations and learning algorithms.

3.1 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 data sets were quite heterogeneous, both with respect to size and with respect to linguistic annotation principles, and the best reported parsing accuracy varied from 65.7% for Turkish to 91.7% for Japanese. The official evaluation metric was the **labeled attachment score** (LAS), defined as the percentage of tokens,

Table 2
Data sets. Tok = number of tokens (×1000); Sen = number of sentences (×1000); T/S = tokens per sentence (mean); Lem = lemmatization present; CPoS = number of coarse-grained part-of-speech tags; PoS = number of (fine-grained) part-of-speech tags; MSF = number of morphosyntactic features (split into atoms); Dep = number of dependency types; NPT = proportion of non-projective dependencies/tokens (%); NPS = proportion of non-projective dependency graphs/sentences (%).

Language	Tok	Sen	T/S	Lem	CPoS	PoS	MSF	Dep	NPT	NPS
Arabic	54	1.5	37.2	yes	14	19	19	27	0.4	11.2
Bulgarian	190	14.4	14.8	no	11	53	50	18	0.4	5.4
Chinese	337	57.0	5.9	no	22	303	0	82	0.0	0.0
Czech	1,249	72.7	17.2	yes	12	63	61	78	1.9	23.2
Danish	94	5.2	18.2	no	10	24	47	52	1.0	15.6
Dutch	195	13.3	14.6	yes	13	302	81	26	5.4	36.4
German	700	39.2	17.8	no	52	52	0	46	2.3	27.8
Japanese	151	17.0	8.9	no	20	77	0	7	1.1	5.3
Portuguese	207	9.1	22.8	yes	15	21	146	55	1.3	18.9
Slovene	29	1.5	18.7	yes	11	28	51	25	1.9	22.2
Spanish	89	3.3	27.0	yes	15	38	33	21	0.1	1.7
Swedish	191	11.0	17.3	no	37	37	0	56	1.0	9.8
Turkish	58	5.0	11.5	yes	14	30	82	25	1.5	11.6

excluding punctuation, that are assigned both the correct head and the correct dependency label.[4]

The outputs of all systems that participated in the shared task are available for download and constitute a rich resource for comparative error analysis. In Section 4, we will use the outputs of MSTParser and MaltParser for all 13 languages, together with the corresponding gold standard graphs used in the evaluation, as the basis for an in-depth error analysis designed to answer Question 2 from Section 1. In Section 6, we will then evaluate our stacking-based parsers on the same data sets and repeat the error analysis. This will allow us to compare the error profiles of the new and old systems at a much finer level of detail than in standard evaluation campaigns.

Table 2 gives an overview of the training sets available for the 13 languages. First of all, we see that training set size varies from over 1.2 million words and close to 73,000 sentences for Czech to only 29,000 words and 1,500 sentences for Slovene. We also see that the average sentence length varies from close to 40 words for Arabic, using slightly different principles for sentence segmentation than the other languages, to less than 10 words for Japanese, where the data consist of transcribed spoken dialogues. Differences such as these can be expected to have a large impact on the parsing accuracy obtained for different languages, and it is probably significant that Japanese has the highest top score of all languages, whereas Arabic has the second lowest. We also see that the amount of information available in the input, in the form of lemmas (Lem), coarse and fine part-of-speech tags (CPoS, PoS), and morphosyntactic features (MSF) varies considerably, as does the granularity of the dependency label sets (Dep). Finally, the proportion of non-projective structures, whether measured on the token level (NPT) or on the sentence level (NPS), is another important source of variation.

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

The final test set for each language has been standardized in size to about 5,000 words, which makes it possible to evaluate the performance of a system over all languages by simply concatenating the system's output for all test sets and comparing this to the concatenation of the gold standard test sets. Thus, most of the statistics used in the subsequent error analysis are based on the concatenation of all test sets. Because some of the phenomena under study are relatively rare, this allows us to get more reliable estimates, even though these estimates inevitably hide important inter-language variation. Analyzing individual languages in more detail would be an interesting complementary study but is beyond the scope of this article.

3.2 Training MSTParser

MSTParser operates primarily over arc-scores, $s(i, j, l)$, which are parameterized by a linear combination of a parameter vector, $\mathbf{w}$, and a corresponding feature vector for the arc, $\mathbf{f}(i, j, l)$. We use a two-stage approach to training. The first-stage learns a model to predict unlabeled dependency trees for a sentence. Thus, arc scores do not condition on possible labels and are parameterized by features only over the head-modifier pair, $s(i, j) = \mathbf{w} \cdot \mathbf{f}(i, j)$. As a result, for the first stage of training the parser, we must define the feature representation $\mathbf{f}(i, j)$, which is outlined in Table 3(a) and Table 3(b) for a potential unlabeled arc (i, j). These features represent both information about the head and modifier in the dependency relation as well as the context of the dependency via local part-of-speech information. We include context part-of-speech features for both the fine-grained and coarse-grained tags (when available).

As mentioned in Section 2.2, the implementation of MSTParser used in our experiments also contains features over adjacent arcs, (i, j) and (i, k), which we will denote compactly as $(i, j \diamond k)$. Scores for adjacent arcs are also defined as a linear combination between weights and a feature vector $s(i, j \diamond k) = \mathbf{w} \cdot \mathbf{f}(i, j \diamond k)$, thus requiring us to define the feature representation $\mathbf{f}(i, j \diamond k)$, which is outlined in Table 3(c). These features

Table 3
Features for MSTParser. $\wedge$ indicates a conjunction of features. † indicates that all back-off versions of a conjunction feature are included as well. A back-off version of a conjunction feature is one where one or more base features are disregarded. ‡ indicates that all back-off versions are included where a single base feature is disregarded.

Base features for sentence: $x = w_0, w_1, \ldots, w_n$
Lexical features: Identity of w_i, $w_i \in x$
Affix features: 3-gram lexical prefix/suffix identity of $\text{Pref}(w_i)/\text{Suff}(w_i)$, $w_i \in x$
Part-of-speech features: Identity of $\text{PoS}(w_i)$, $w_i \in x$
Morphosyntactic features: For all morphosyntactic features MSF_k for a word w_i, identity of $\text{MSF}_k(w_i)$, $w_i \in x$
Label features: Identity of l in some labeled arc (i, j, l)

(a) Head-modifier features for unlabeled arc (i, j)
$w_i \wedge \text{PoS}(w_i) \wedge w_j \wedge \text{PoS}(w_j)$ †
$\text{Pref}(w_i) \wedge \text{PoS}(w_i) \wedge \text{Pref}(w_j) \wedge \text{PoS}(w_j)$ †
$\text{Suff}(w_i) \wedge \text{PoS}(w_i) \wedge \text{Suff}(w_j) \wedge \text{PoS}(w_j)$ †
$\forall k, k' : \text{MSF}_k(w_i) \wedge \text{PoS}(w_i) \wedge \text{MSF}_{k'}(w_j) \wedge \text{PoS}(w_j)$ †

(b) PoS-context features for unlabeled arc (i, j)
$\forall k, i < k < j : \text{PoS}(w_i) \wedge \text{PoS}(w_k) \wedge \text{PoS}(w_j)$
$\text{PoS}(w_{i-1}) \wedge \text{PoS}(w_i) \wedge \text{PoS}(w_{j-1}) \wedge \text{PoS}(w_j)$ ‡
$\text{PoS}(w_{i-1}) \wedge \text{PoS}(w_i) \wedge \text{PoS}(w_j) \wedge \text{PoS}(w_{j+1})$ ‡
$\text{PoS}(w_i) \wedge \text{PoS}(w_{i+1}) \wedge \text{PoS}(w_{j-1}) \wedge \text{PoS}(w_j)$ ‡
$\text{PoS}(w_i) \wedge \text{PoS}(w_{i+1}) \wedge \text{PoS}(w_j) \wedge \text{PoS}(w_{j+1})$ ‡

(c) Head-modifier features for unlabeled arc pair $(i, j \diamond k)$
$w_j \wedge w_k$
$w_j \wedge \text{PoS}(w_k)$
$\text{PoS}(w_j) \wedge w_k$
$\text{PoS}(w_j) \wedge \text{PoS}(w_k)$
$\text{PoS}(w_i) \wedge \text{PoS}(w_j) \wedge \text{PoS}(w_k)$

(d) Arc-label features for labeled arc (i, j, l)
$w_i \wedge \text{PoS}(w_i) \wedge w_j \wedge \text{PoS}(w_j) \wedge l$ †
$\forall k, i < k < j : \text{PoS}(w_i) \wedge \text{PoS}(w_k) \wedge \text{PoS}(w_j) \wedge l$
$\text{PoS}(w_{j-1}) \wedge \text{PoS}(w_j) \wedge \text{PoS}(w_{j+1}) \wedge l$ †
$\text{PoS}(w_{i-1}) \wedge \text{PoS}(w_i) \wedge \text{PoS}(w_{i+1}) \wedge l$ †

attempt to capture likely properties about adjacent arcs in the tree via their lexical and part-of-speech information. Finally, all features in Table 3(a)–(c) contain two versions. The first is the standard feature outlined in the table, and the second is the feature conjoined with both the direction of dependency attachment (left or right) as well as the bucketed distance between the head and modifier in buckets of 0 (adjacent), 1, 2, 3, 4, 5–9, and 10+.

For the second stage label classifier we use a log-linear classifier, which is again parameterized by a vector of weights and a corresponding feature vector $s(l|i,j) = \mathbf{w} \cdot \mathbf{f}(i,j,l)$, where the score of a label l is now conditioned on a fixed dependency arc (i,j) produced from the first-stage unlabeled parser. The feature representation $\mathbf{f}(i,j,l)$ is defined in Table 3(d). These features provide the lexical and part-of-speech context for determining whether a given arc label is suitable for a given head and modifier. Each feature in Table 3(d) again has two versions, except this time the second version is only conjoined with attachment direction.

These feature representations were used to train an on-line large-margin unlabeled dependency parser (McDonald, Crammer, and Pereira 2005; McDonald and Pereira 2006) and a log-linear arc-labeler (Berger, Pietra, and Pietra 1996) regularized with a zero mean Gaussian prior with the variance hyper-parameter set to 1.0. The unlabeled dependency parser was trained for 10 iterations and the log-linear arc-labeler was trained for 100 iterations. The feature sets and model hyper-parameters were fixed for all languages. The only exception is that features containing coarse part-of-speech or morphosyntactic information were ignored if this information was not available in a corresponding treebank.

3.3 Training MaltParser

Training MaltParser amounts to estimating a function for scoring configuration-transition pairs (c,t), represented by a feature vector $\mathbf{f}(c,t) \in \mathbb{R}^k$. Features are defined in terms of arbitrary properties of the configuration c, including the state of the stack σ_c, the input buffer β_c, and the partially built dependency graph G_c (represented in the configuration by the arc set A_c). In particular, many features involve properties of the two target tokens, the token on top of the stack σ_c (denoted σ_c^0) and the first token in the input buffer β_c (denoted β_c^0), which are the two tokens that may become connected by a dependency arc through the transition out of c. The basic feature representation used for all languages in the CoNLL-X shared task included three groups of features:[5]

- Part-of-speech features: Identity of PoS(w), $w \in \{\sigma_c^0, \sigma_c^1, \beta_c^0, \beta_c^1, \beta_c^2, \beta_c^3\}$.
- Lexical features: Identity of w, $w \in \{\sigma_c^0, \beta_c^0, \beta_c^1\}$ or $(w, \sigma_c^0, l) \in G_c$.
- Arc features: Identity of l, $(w, w', l) \in G_c$ and $w \in \{\sigma_c^0, \beta_c^0\}$ or $w' \in \{\sigma_c^0\}$.

Note in particular that features can be defined with respect to the partially built dependency graph G_c. This is most obvious for the arc features, which extract the labels of particular arcs in the graph, but it is also true of the last lexical feature, which picks out the word form of the syntactic head of the word on top of the stack. This is precisely what gives transition-based parsers a richer feature space than their graph-based

5 We use the notation σ_c^i and β_c^i to denote the ith element from the top/head of the stack/buffer (with index 0 for the first element).

counterparts, even though graph-defined features are usually limited to a fairly small region of the graph around σ_c^0 and β_c^0, such as their leftmost and rightmost dependents and their syntactic head (if available).

Over and above the basic features described here, additional features were added for some languages depending on availability in the training data. This included the following:

- Coarse part-of-speech features: Identity of CPoS(w), $w \in \{\sigma_c^0, \beta_c^0, \beta_c^1\}$.
- Lemma features: Identity of Lem(w), $w \in \{\sigma_c^0, \beta_c^0, \beta_c^1\}$.
- Morphosyntactic features: Identity of MSF(w), $w \in \{\sigma_c^0, \beta_c^0, \beta_c^1\}$.

Additional feature selection experiments were carried out for each language to the extent that time permitted. Complete information about feature representations can be found in Nivre et al. (2006) and on the companion web site.[6]

The feature representations described here were used to train support vector machines as implemented in the LIBSVM library (Chang and Lin 2001), with a quadratic kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^2$ and LIBSVM's built-in one-versus-one strategy for multi-class classification, converting symbolic features to numerical ones using the standard technique of binarization.[7] One thing to note is that the quadratic kernel implicitly adds features corresponding to pairs of explicit features, thus obviating the need for explicit feature conjunctions as seen in the feature representations of MSTParser.

4. Error Analysis

A primary goal of this study is to characterize the errors made by standard data-driven dependency parsing models. To that end, this section presents a number of experiments that relate parsing errors to a set of linguistic and structural properties of the input and predicted/gold standard dependency trees. We argue throughout that the results of this analysis can be correlated to specific theoretical aspects of each model—in particular the trade-off previously highlighted in Section 2.4.

For simplicity, all experiments report labeled parsing metrics (either accuracy, precision, or recall). Identical experiments using unlabeled parsing accuracies did not reveal any additional information. Statistical significance was measured—for each metric at each point along the operating curve—by employing randomized stratified shuffling at the instance level using 10,000 iterations.[8] Furthermore, all experiments report aggregate statistics over the data from all 13 languages together, as explained in Section 3. Finally, in all figures and tables, MSTParser and MaltParser are referred to as MST and Malt, respectively, for short.

4.1 Length Factors

It is well known that parsing systems tend to have lower accuracies for longer sentences. This is primarily due to the increased presence of complex syntactic constructions

6 `http://maltparser.org/conll/conllx`.
7 For details about parameter settings, we again refer to Nivre et al. (2006) and the companion web site `http://maltparser.org/conll/conllx/`.
8 This is the method used by the CoNLL-X shared task on dependency parsing.

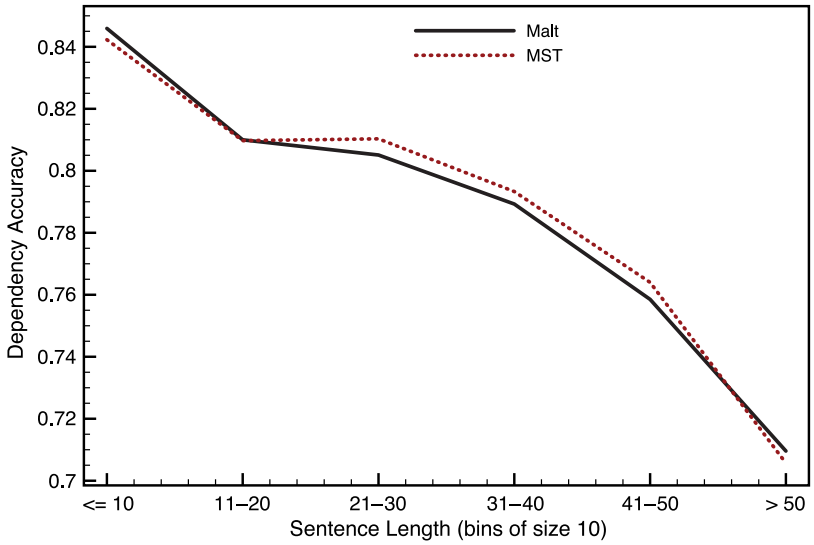

Figure 4
Accuracy relative to sentence length. Differences statistically significant ($p < 0.05$) at no positions.

involving prepositions, conjunctions, and multi-clause sentences. Figure 4 shows the accuracy of both parsing models relative to sentence length (in bins of size 10: 1–10, 11–20, etc.). System performance is almost indistinguishable, but MaltParser tends to perform better on shorter sentences, which require the greedy inference algorithm to make fewer parsing decisions. As a result, the chance of error propagation is reduced significantly when parsing these sentences. However, if this was the only difference between the two systems, we would expect them to have equal accuracy for shorter sentences. The fact that MaltParser actually has higher accuracy when the likelihood of error propagation is reduced is probably due to its richer feature space relative to MSTParser.

Another interesting property is accuracy relative to dependency length as opposed to sentence length. We define the length of a dependency from word w_i to word w_j as 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 neighbors. Figure 5 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 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 behavior can be explained using the same reasoning as above: Shorter dependency arcs are usually created first in the greedy parsing procedure of MaltParser and are less prone to error propagation. In contrast, longer dependencies are typically constructed at the later stages of the parsing algorithm and are affected more by error propagation. Theoretically, MSTParser should not perform better or worse for arcs of any length. However, due to the fact that longer dependencies are typically harder to parse, there is still a degradation in performance for MSTParser—up to 20% in the

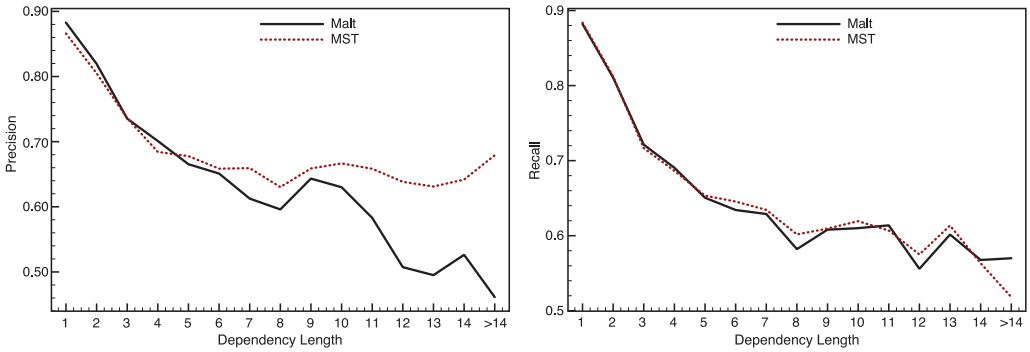

Figure 5
Dependency arc precision/recall relative to predicted/gold dependency length. Precision statistically significant ($p < 0.05$) at 1, 2, 4, 7, 8, 10 through >14. Recall statistically significant ($p < 0.05$) at >14.

extreme. However, the precision curve for MSTParser is much flatter than MaltParser, which sees a drop of up to 40% in the extreme. Note that even though the area under the curve is much larger for MSTParser, the number of dependency arcs with a length >10 is much smaller than the number with length <10, which is why the overall accuracy of the two systems is nearly identical.

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. For example, the dependency arc from ROOT to *is* in Figure 1 would have a distance of 1 and the arc from *hearing* to *A* a distance of 3. Figure 6 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

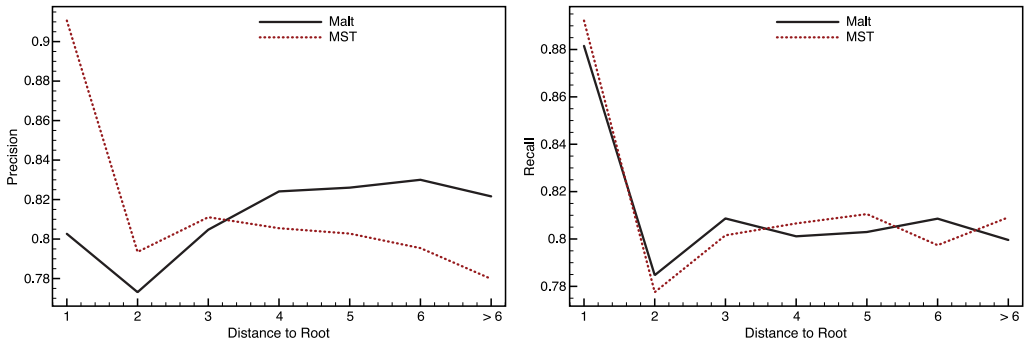

Figure 6
Dependency arc precision/recall relative to the predicted/gold distance to root. Precision statistically significant ($p < 0.05$) at 1, 2, 4 through >6. Recall statistically significant ($p < 0.05$) at 1, 2, 3.

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 6 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 because 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 over-predict root modifiers, which comes at the expense of its precision. This is 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 behavior 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 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 slight downward trend for arcs of distance greater than 3 from the root. An examination of dependency length for predicted arcs shows that MSTParser predicts many more arcs of length 1 than MaltParser, which naturally leads to over-predicting more arcs at larger distances from the root due to the presence of chains, which in turn will lower precision for these arcs. In ambiguous situations, it is not surprising that MSTParser predicts many length-1 dependencies, as this is the most common dependency length across treebanks. Thus, whereas MaltParser pushes difficult parsing decisions higher in the graph, MSTParser appears to push these decisions lower.

The final graph property we will examine aims to quantify the local neighborhood 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., $i = i'$). In Figure 1 the arcs from the word *is* to the words *hearing*, *scheduled*, and the period are all considered siblings under this definition. Figure 7 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.

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. However, given the important typological differences that exist between languages, as well as the diversity of annotation schemes used in different treebanks, it is far from straightforward to compare these categories across languages. Nevertheless, 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.

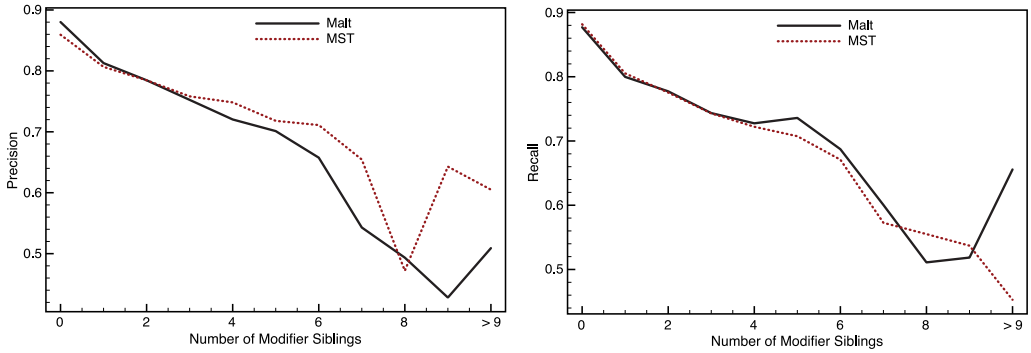

Figure 7
Dependency arc precision/recall relative to the number of predicted/gold siblings. Precision statistically significant ($p < 0.05$) at 0, 4, 6, 7, 9. Recall statistically significant ($p < 0.05$) at 5, >9.

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 have only managed to 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, and an object category (including both direct and indirect objects). Unfortunately, we had to exclude many interesting types that could not be identified with high enough precision across languages, such as adverbials, which cannot be clearly distinguished in annotation schemes that subsume them under a general modifier category, and coordinate structures, which are sometimes annotated with special dependency types, sometimes with ordinary dependency types found also in non-coordinated structures.

Table 4(a) 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, and MSTParser does better on all other categories, in particular conjunctions. This pattern is consistent with previous results insofar as verbs

Table 4
(a) Accuracy relative to dependent part of speech. (b) Precision/recall for different dependency types.

Part of Speech	MST	Malt
Verb	82.6	81.9
Noun	80.0	80.7
Pronoun	88.4	89.2
Adjective	89.1	87.9
Adverb	78.3	77.4
Adposition	69.9	68.8
Conjunction	73.1	69.8

(a)

Dependency Type	MST Precision/Recall	Malt Precision/Recall
Root	89.9 / 88.7	84.7 / 87.5
Subject	79.9 / 78.9	80.3 / 80.7
Object	76.5 / 77.7	77.2 / 77.6

(b)

and conjunctions are often involved in dependencies closer to the root that span longer distances, whereas nouns and pronouns are typically attached to verbs and therefore occur lower in the graph and with shorter distances. Thus, the average distance to the root is 3.1 for verbs and 3.8 for conjunctions, but 4.7 for nouns and 4.9 for pronouns; the average dependency length is 4.2 for verbs, 4.8 for conjunctions, 2.3 for nouns, and 1.6 for pronouns. Adverbs resemble verbs and conjunctions with respect to root distance (3.7) but group with nouns and pronouns for dependency length (2.3), so it appears that the former is more important here. Furthermore, adverb modifiers have 2.4 siblings on average, which is greater than the sibling average for conjunctions (2.1), adpositions (1.9), pronouns (1.7), verbs (1.3), nouns (1.3), and adjectives (1.2). This would be consistent with the graph in Figure 7.

Adpositions and especially adjectives constitute a puzzle. With a root distance of 4.4 and 5.2, respectively, a dependency length of 2.5/1.5 and a sibling average of 1.9/1.2, we would expect MaltParser to do better than MSTParser for these categories. 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. At present, we do not have an explanation for this behavior.

Finally, in Table 4(b), 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 MaltParser integrates labeling into the parsing process, which means that previously assigned dependency labels can be used as features. This may sometimes be important to disambiguate subjects and objects, especially in free-word order languages where a dependent's position relative to the verb does not determine its syntactic role.

4.4 Discussion

The experiments in this 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 typically 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 classes of arcs like short dependencies or subject and object modifiers. 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 majority 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. By understanding the strengths and weaknesses of each model we have gained insights towards new and better models for dependency parsing.

To get some upper bounds on the improvement that can be obtained by combining the strengths of each model, we can perform 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 labeled attachment score 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 from this section to generate ideas for new models. Below we sketch some possible new directions:

1. *Ensemble systems:* The error analysis presented in this article 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. *Integrated/Hybrid systems:* Rather than using an ensemble of several independent parsers, we may construct systems that trust different parsers in different situations, possibly based on the characteristics of the input and predicted dependency trees. The oracle results reported here show that such an approach could potentially result in substantial improvements.
3. *Novel approaches:* The theoretical analysis presented in this article reveals that the two dominant approaches are each based on a particular combination of training and inference methods, which raises the question of which other combinations can fruitfully be explored. For example, can we construct globally trained, greedy, transition-based parsers? Or graph-based parsers with global features? To some extent the former characterization fits the approach of Zhang and Clark (2008) and Huang and Sagae (2010), and the latter that of Riedel and Clarke (2006), Nakagawa (2007), and others. The analysis presented in this section explains the relative success of such approaches.

In the next two sections we explore a model that falls into category 2. The system we propose uses a two-stage stacking framework, where a second-stage parser conditions on the predictions of a first-stage parser during inference. The second-stage parser is also learned with access to the first-stage parser's decisions and thus learns when to trust the first-stage parser's predictions and when to trust its own. The method is not a traditional ensemble, because the parsers are not learned independently of one another.

5. Integrated Models

As just discussed, there are many conceivable ways of combining the two parsers, including more or less complex ensemble systems and voting schemes, which only perform the integration at parsing time. However, given that we are dealing with data-driven models, it should be possible to integrate at learning time, so that the two complementary models can learn from one another. In this article, we propose to do this by letting one model generate features for the other in a stacked learning framework.

Feature-based integration in this sense has previously been exploited for dependency parsing by McDonald (2006), who trained an instance of MSTParser using features generated by the parsers of Collins (1999) and Charniak (2000), which improved unlabeled accuracy by 1.7 percentage points on data from the Penn Treebank. In other NLP domains, feature-based integration has been used by Taskar, Lacoste-Julien, and Klein (2005), who trained a discriminative word alignment model using features derived from the IBM models, by Florian et al. (2004), who trained classifiers on auxiliary data to guide named entity classifiers, and by others.

Feature-based integration also has points in common with co-training, which has been applied to syntactic parsing by Sarkar (2001) and Steedman et al. (2003), among others. The difference, of course, is that standard co-training is a weakly supervised method, where the first-stage parser's predictions *replace*, rather than *complement*, the gold standard annotation during training. Feature-based integration is also similar to parse reranking (Collins 2000), where one parser produces a set of candidate parses and a second-stage classifier chooses the most likely one. However, feature-based integration is not explicitly constrained to any parse decisions that the first-stage parser might make. Furthermore, as only the single most likely parse is used from the first-stage model, it is significantly more efficient than reranking, which requires both computationally and conceptually more complex parsing algorithms (Huang and Chiang 2005).

5.1 Parser Stacking with Rich Features

As explained in Section 2, both models essentially learn a scoring function $s : X \rightarrow \mathbb{R}$, where the domain X is different for the two models. For the graph-based model, X is the set of possible dependency arcs (i, j, l); for the transition-based model, X is the set of possible configuration-transition pairs (c, t). But in both cases, the input is represented by a k-dimensional feature vector $\mathbf{f} : X \rightarrow \mathbb{R}^k$. In a stacked parsing system we simply extend the feature vector for one model, called the **base model**, with a certain number of features generated by the other model, which we call the **guide model** in this context. The additional features will be referred to as **guide features**, and the version of the base model trained with the extended feature vector will be called the **guided model**. The idea is that the guided model should be able to learn in which situations to trust the guide features, in order to exploit the complementary strength of the guide model, so that performance can be improved with respect to the base model.

The exact form of the guide features depends on properties of the base model and will be discussed in Sections 5.2–5.3, but the overall scheme for the stacked parsing model can be described as follows. Assume as input a training set $T = \{(x_t, G_{x_t})\}_{t=1}^{|T|}$ of input sentences x_t and corresponding gold standard dependency trees G_{x_t}. In order to train the guide model we use a cross-validation scheme and divide T into n different disjoint subsets T_i (i.e., $T = \bigcup_{i=1}^{n} T_i$). Let $M[T]$ be the result of training the model M on T

and let $M[T](x)$ be the result of parsing a new input sentence x with $M[T]$. Now, consider a guide model C, base model B, and guided model B_C. For each x in T, define

$$G_x^C = C[T - T_i](x) \text{ if } x \in T_i$$

G_x^C is the prediction of model C on training input x when C is trained on all the subsets of T, except the one containing x. The reason for using this cross-validation scheme is that if C had been trained on all of T, then G_x^C would not be representative of the types of errors that C might make when parsing sentence x. Using cross-validation in this way is similar to how it is used in parse reranking (Collins 2000). Now, define a new training set of the form $T' = \{(\langle x_t, G_{x_t}^C \rangle, G_{x_t})\}_{t=1}^{|T|}$. That is, T' is identical to T, except that each training input x is augmented with the cross-validation prediction of model C. Finally, let

$$B_C = B[T']$$

This means that, for every sentence $x \in T$, B_C has access at training time to both the gold standard dependency graph G_x and the graph G_x^C predicted by C. Thus, B_C is able to define guide features over G_x^C, which can prove beneficial if features over G_x^C can be used to discern when parsing model C outperforms or underperforms parsing model B. When parsing a new sentence x with B_C, x is first parsed with model $C[T]$ (this time trained on the entire training set) to derive an input $\langle x, G_x^C \rangle$, so that the guide features can be extracted also at parsing time. This input is then passed through model B_C.

5.2 The Guided Graph-Based Model

The graph-based model, MSTParser, learns a scoring function $s(i, j, l) \in \mathbb{R}$ over labeled dependencies. As described in Section 3.2, dependency arcs (or pairs of arcs) are represented by a high dimensional feature vector $\mathbf{f}(i, j, l) \in \mathbb{R}^k$, where $\mathbf{f}$ is typically a binary feature vector over properties of the arc as well as the surrounding input (McDonald, Crammer, and Pereira 2005; McDonald, Lerman, and Pereira 2006). For the guided graph-based model, which we call MST_{Malt}, this feature representation is modified to include an additional argument G_x^{Malt}, which is the dependency graph predicted by MaltParser on the input sentence x. Thus, the new feature representation will map an arc *and* the entire predicted MaltParser graph to a high dimensional feature representation, $\mathbf{f}(i, j, l, G_x^{\text{Malt}}) \in \mathbb{R}^{k+m}$. These m additional features account for the guide features over the MaltParser output. The specific features used by MST_{Malt} are given in Table 5. All features are conjoined with the part-of-speech tags of the words involved in the

Table 5
Guide features for MST_{Malt} and Malt_{MST}.

MST_{Malt} – defined over (i, j, l) ($*$ = any label/node)	**Malt_{MST}** – defined over (c, t) ($*$ = any label/node)
Is $(i, j, *)$ in G_x^{Malt}?	Is $(\sigma_c^0, \beta_c^0, *)$ in G_x^{MST}?
Is (i, j, l) in G_x^{Malt}?	Is $(\beta_c^0, \sigma_c^0, *)$ in G_x^{MST}?
Is $(i, j, *)$ not in G_x^{Malt}?	Head direction for σ_c^0 in G_x^{MST} (left/right/ROOT)
Is (i, j, l) not in G_x^{Malt}?	Head direction for β_c^0 in G_x^{MST} (left/right/ROOT)
Identity of l' such that $(*, j, l')$ is in G_x^{Malt}?	Identity of l such that $(*, \sigma_c^0, l)$ is in G_x^{MST}?
Identity of l' such that (i, j, l') is in G_x^{Malt}?	Identity of l such that $(*, \beta_c^0, l)$ is in G_x^{MST}?

dependency to allow the guided model to learn weights relative to different surface syntactic environments. Features that include the arc label l are only included in the second-stage arc-labeler. Though MSTParser is capable of defining features over pairs of arcs, we restrict the guide features to single arcs as this resulted in higher accuracies during preliminary experiments.

5.3 The Guided Transition-Based Model

The transition-based model, MaltParser, learns a scoring function $s(c, t) \in \mathbb{R}$ over configurations and transitions. The set of training instances for this learning problem is the set of pairs (c, t) such that t is the correct transition out of c in the transition sequence that derives the correct dependency graph G_x for some sentence x in the training set T. As described in Section 3.3, each training instance (c, t) is represented by a feature vector $\mathbf{f}(c, t) \in \mathbb{R}^k$, where features are defined in terms of arbitrary properties of the configuration c.

For the guided transition-based model, which we call Malt_{MST}, training instances are extended to triples (c, t, G_x^{MST}), where G_x^{MST} is the dependency graph predicted by the graph-based MSTParser for the sentence x to which the configuration c belongs. We define m additional guide features, based on properties of G_x^{MST}, and extend the feature vector accordingly to $\mathbf{f}(c, t, G_x^{\text{MST}}) \in \mathbb{R}^{k+m}$. The specific features used by Malt_{MST} are given in Table 5. Unlike MSTParser, features are not explicitly defined to conjoin guide features with part-of-speech features. These features are implicitly added through the polynomial kernel used to train the SVM.

6. Integrated Parsing Experiments

In this section, we present an experimental evaluation of the two guided models followed by a comparative error analysis including both the base models and the guided models. The data sets used in these experiments are identical to those used in Section 4. The guided models were trained according to the scheme explained in Section 5, with two-fold cross-validation when parsing the training data with the guide parsers. Preliminary experiments suggested that cross-validation with more folds had a negligible impact on the results. Models are evaluated by their labeled attachment score on the test set using the evaluation software from the CoNLL-X shared task with default settings.[9] Statistical significance was assessed using Dan Bikel's randomized parsing evaluation comparator with the default setting of 10,000 iterations.[10]

6.1 Results

Table 6 shows the results, for each language and on average, for the two base models (MST, Malt) and for the two guided models (MST_{Malt}, Malt_{MST}). We also give oracle combination scores based on both by taking the best graph or the best set of arcs relative to the gold standard, as discussed in Section 4.4. First of all, we see that both guided models show a consistent increase in accuracy compared to their base model, even though the extent of the improvement varies across languages from about half a percentage point (Malt_{MST} on Chinese) up to almost four percentage points (Malt_{MST} on

9 http://nextens.uvt.nl/~conll/software.html.
10 http://www.cis.upenn.edu/~dbikel/software.html.

Table 6
Labeled attachment scores for base parsers and guided parsers (improvement in percentage points).

					Oracle	
Language	**MST**	**MST_{Malt}**	**Malt**	**$Malt_{MST}$**	**graph**	**arc**
Arabic	66.91	**68.64** (+1.73)	66.71	67.80 (+1.09)	70.3	75.8
Bulgarian	87.57	**89.05** (+1.48)	87.41	88.59 (+1.18)	90.7	92.4
Chinese	85.90	**88.43** (+2.53)	86.92	87.44 (+0.52)	90.8	91.5
Czech	80.18	**82.26** (+2.08)	78.42	81.18 (+2.76)	84.2	86.6
Danish	84.79	**86.67** (+1.88)	84.77	85.43 (+0.66)	87.9	89.6
Dutch	79.19	**81.63** (+2.44)	78.59	79.91 (+1.32)	83.5	86.4
German	87.34	**88.46** (+1.12)	85.82	87.66 (+1.84)	89.9	92.0
Japanese	90.71	91.43 (+0.72)	91.65	**92.20** (+0.55)	93.2	94.1
Portuguese	86.82	87.50 (+0.68)	87.60	**88.64** (+1.04)	90.0	91.6
Slovene	73.44	**75.94** (+2.50)	70.30	74.24 (+3.94)	77.2	80.7
Spanish	82.25	**83.99** (+1.74)	81.29	82.41 (+1.12)	85.4	88.2
Swedish	82.55	**84.66** (+2.11)	84.58	84.31 (–0.27)	86.8	88.8
Turkish	63.19	64.29 (+1.10)	65.58	**66.28** (+0.70)	69.3	72.6
Average	80.83	**82.53** (+1.70)	80.75	82.01 (+1.27)	84.5	86.9

Slovene).[11] It is thus quite clear that both models have the capacity to learn from features generated by the other model. However, it is also clear that the graph-based MST model shows a somewhat larger improvement, both on average and for all languages except Czech, German, Portuguese, and Slovene. Finally, given that the two base models had the best performance for these data sets at the CoNLL-X shared task, the guided models achieve a substantial improvement of the state of the art.[12] Although there is no statistically significant difference between the two base models, they are both outperformed by $Malt_{MST}$ ($p < 0.0001$), which in turn has significantly lower accuracy than MST_{Malt} ($p < 0.0005$).

An extension to the models described so far would be to iteratively integrate the two parsers in the spirit of pipeline iteration (Hollingshead and Roark 2007). For example, one could start with a Malt model, use it to train a guided MST_{Malt} model, then use that as the guide to train a $Malt_{MST_{Malt}}$ model, and so forth. We ran such experiments, but found that accuracy did not increase significantly and in some cases decreased slightly. This was true regardless of which parser began the iterative process. In retrospect, this result is not surprising. Because the initial integration effectively incorporates knowledge from both parsing systems, there is little to be gained by adding additional parsers in the chain.

6.2 Error Analysis

The experimental results presented so far show that feature-based integration (stacking) is a viable approach for improving the accuracy of both graph-based and transition-based models for dependency parsing, but they say very little about how the integration

11 The only exception to this pattern is the result for $Malt_{MST}$ on Swedish, where we see an unexpected drop in accuracy compared to the base model.
12 Martins et al. (2008) and Martins, Smith, and Xing (2009) report additional improvements.

benefits the two models and what aspects of the parsing process are improved as a result. In order to get a better understanding of these matters, we replicate parts of the error analysis presented in Section 4, but include both integrated models into the analysis. As in Section 4, for each of the four models evaluated, we aggregate error statistics for labeled attachment over all 13 languages together.

Figure 8 shows accuracy in relation to sentence length, binned into 10-word intervals (1–10, 11-20, etc.). As mentioned earlier, Malt and MST have very similar accuracy for short sentences but Malt degrades more rapidly with increasing sentence length because of error propagation. The guided models, Malt_{MST} and MST_{Malt}, behave in a very similar fashion with respect to each other but both outperform their base parser over the entire range of sentence lengths. However, except for the two extreme data points (0–10 and 51–60) there is also a slight tendency for Malt_{MST} to improve more for longer sentences (relative to its base model) and for MST_{Malt} to improve more for short sentences (relative to its base model). Thus, whereas most of the improvement for the guided parsers seems to come from a higher accuracy in predicting arcs in general, there is also some evidence that the feature-based integration allows one parser to exploit the strength of the other.

Figure 9 plots precision (left) and recall (right) for dependency arcs of different lengths (predicted arcs for precision, gold standard arcs for recall). With respect to recall, the guided models appear to have a slight advantage over the base models for short and medium distance arcs. With respect to precision, however, there are two clear patterns. First, the graph-based models have better precision than the transition-based models when predicting long arcs, as discussed earlier. Secondly, both the guided models have better precision than their base model and, for the most part, also their guide model. In particular MST_{Malt} outperforms MST for all dependency lengths and is comparable to Malt for short arcs. More interestingly, Malt_{MST} outperforms both Malt and MST for arcs up to length 9, which provides evidence that Malt_{MST} has learned specifically to

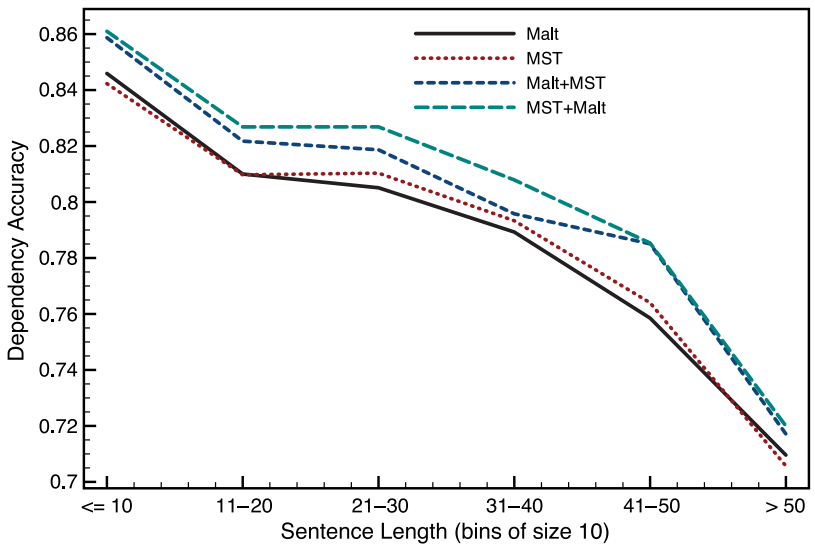

Figure 8
Accuracy relative to sentence length. Differences between MST+Malt and MST statistically significant ($p < 0.05$) at all positions. Differences between Malt+MST and Malt statistically significant ($p < 0.05$) at all positions. Differences between MST+Malt and Malt+MST statistically significant ($p < 0.05$) at 11–20, 21–30, and 31–40.

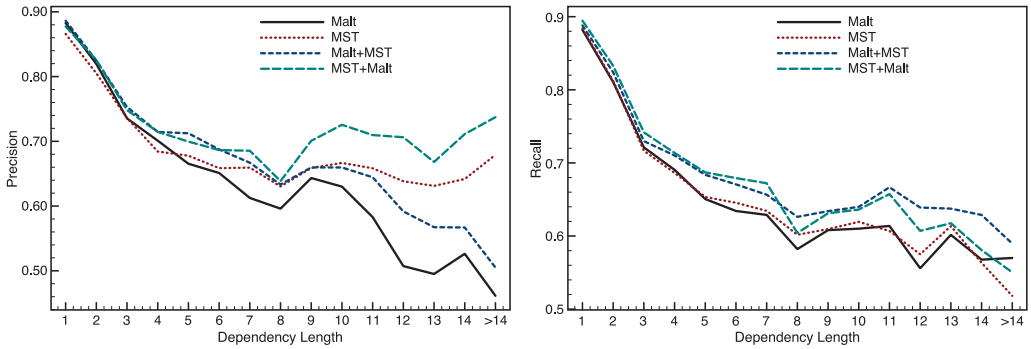

Figure 9
Dependency arc precision/recall relative to predicted/gold for dependency length. Precision between MST+Malt and MST statistically significant ($p < 0.05$) at 1–7, 9–12, 14, and >14. Recall between MST+Malt and MST statistically significant ($p < 0.05$) at 1–7, 9, 14, and >14. Precision between Malt+MST and Malt statistically significant ($p < 0.05$) at 1–8, 10–13, and > 14. Recall between Malt+MST and Malt statistically significant ($p < 0.05$) at 1–12, 14, and >14. Precision between MST+Malt and Malt+MST statistically significant ($p < 0.05$) at 1 and 9–>14. Recall between MST+Malt and Malt+MST statistically significant ($p < 0.05$) at 1, 2, 3, 14, and >14.

trust the guide features from MST for longer dependencies (those greater than length 4) and its own base features for shorter dependencies (those less than or equal to length 4). However, for dependencies of length greater than 9, the performance of Malt_{MST} begins to degrade. Because the absolute number of dependencies of length greater than 9 in the training sets is relatively small, it might be difficult for Malt_{MST} to learn from the guide parser in these situations. Interestingly, both models seem to improve most in the medium range (roughly 8–12 words), although this pattern is clearer for MSTParser than for MaltParser.

Figure 10 shows precision (left) and recall (right) for dependency arcs at different distances from the root (predicted arcs for precision, gold standard arcs for recall).

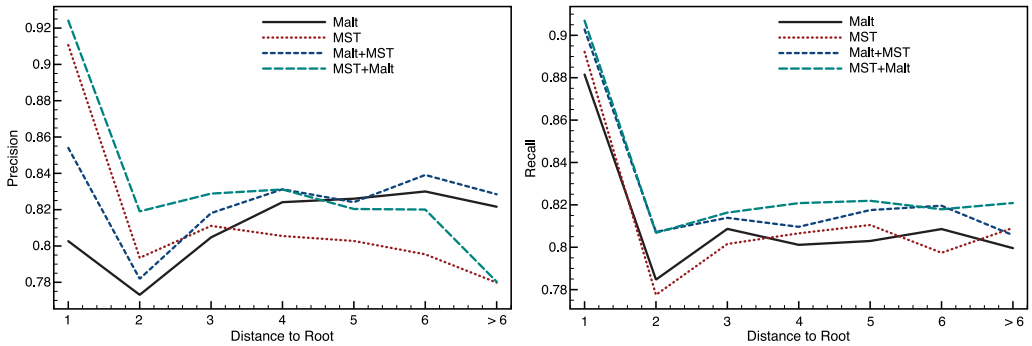

Figure 10
Dependency arc precision/recall relative to predicted/gold for distance to root. Precision between MST+Malt and MST statistically significant ($p < 0.05$) at 1–6. Recall between MST+Malt and MST statistically significant ($p < 0.05$) at all positions. Precision between Malt+MST and Malt statistically significant ($p < 0.05$) at 1–4. Recall between Malt+MST and Malt statistically significant ($p < 0.05$) at all positions. Precision between MST+Malt and Malt+MST statistically significant ($p < 0.05$) at 1, 2, 3, 6, and >6. Recall between MST+Malt and Malt+MST statistically significant ($p < 0.05$) at 4 and >6.

Again, we find the clearest patterns in the graphs for precision, where Malt has very low precision near the root but improves with increasing depth, whereas MST shows the opposite trend, as observed earlier. Considering the guided models, it is clear that Malt_{MST} improves in the direction of its guide model, with a five-point increase in precision for dependents of the root and smaller improvements for longer distances (where its base model is most accurate). Similarly, MST_{Malt} improves precision the largest in the range where its base model is inferior to Malt (roughly distances of 2–6) and is always superior to its base model. This again indicates that the guided models are learning from their guide models as they improve the most in situations where the base model has inferior accuracy.

Table 7 gives the accuracy for arcs relative to dependent part of speech. As observed earlier, we see that MST does better than Malt for all categories except nouns and pronouns. But we also see that the guided models in all cases improve over their base model and, in most cases, also over their guide model. The general trend is that MST improves more than Malt, except for adjectives and conjunctions, where Malt has a greater disadvantage from the start and therefore benefits more from the guide features. The general trend is that the parser with worse performance for a particular part-of-speech tag improves the most in terms of absolute accuracy (5 out of 7 cases), again suggesting that the guided models are learning when to trust their guide features. The exception here is verbs and adverbs, where MST has superior performance to Malt, but MST_{Malt} has a larger increase in accuracy than Malt_{MST}.

Considering the results for parts of speech, as well as those for dependency length and root distance, it is interesting to note that the guided models often improve even in situations where their base models are more accurate than their guide models. This suggests that the improvement is not a simple function of the raw accuracy of the guide model but depends on the fact that labeled dependency decisions interact in inference algorithms for both graph-based and transition-based parsing systems. Thus, if a parser can improve its accuracy on one class of dependencies (for example, longer ones), then we can expect to see improvements on all types of dependencies—as we do.

6.3 Discussion

In summation, it is clear that both guided models benefit from a higher accuracy in predicting arcs in general, which results in better performance regardless of sentence length, dependency length, or dependency depth. However, there is strong evidence that MST_{Malt} improves in the direction of Malt, with a slightly larger improvement compared to its base model for short sentences and short dependencies (but not for deep

Table 7
Accuracy relative to dependent part of speech (improvement in percentage points).

Part of Speech	MST	MST_{Malt}	Malt	Malt_{MST}
Verb	82.6	**85.1** (2.5)	81.9	84.3 (2.4)
Noun	80.0	81.7 (1.7)	80.7	**81.9** (1.2)
Pronoun	88.4	**89.4** (1.0)	89.2	89.3 (0.1)
Adjective	89.1	**89.6** (0.5)	87.9	89.0 (1.1)
Adverb	78.3	**79.6** (1.3)	77.4	78.1 (0.7)
Adposition	69.9	**71.5** (1.6)	68.8	70.7 (1.9)
Conjunction	73.1	**74.9** (1.8)	69.8	72.5 (2.7)

dependencies). Conversely, Malt_{MST} improves in the direction of MST, with a larger improvement for long sentences and for dependents of the root.

The question remains why MST generally benefits more from the feature-based integration. The likely explanation is the previously mentioned interaction between different dependency decisions at inference time. Because inference in MST is exact (or nearly exact), an improvement in one type of dependency has a good chance of influencing the accuracy of other dependencies, whereas in the transition-based model, where inference is greedy, some of these additional benefits will be lost because of error propagation. This is reflected in the error analysis in the following recurrent pattern: Where Malt does well, Malt_{MST} does only slightly better. But where MST is good, MST_{Malt} is often significantly better. Furthermore, this observation easily explains the limited increases in accuracy of words with verb and adverb modifiers that is observed in Malt_{MST} relative to MST_{Malt} (Table 7) as these dependencies occur close to the root and have increased likelihood of being affected by error propagation.

Another part of the explanation may have to do with the learning algorithms used by the systems. Although both Malt and MST use discriminative algorithms, Malt uses a batch learning algorithm (SVM) and MST uses an on-line learning algorithm (MIRA). If the original rich feature representation of Malt is sufficient to separate the training data, regularization may force the weights of the guided features to be small (as they are not needed at training time). On the other hand, an on-line learning algorithm will recognize the guided features as strong indicators early in training and give them a high weight as a result. Frequent features with high weight early in training tend to have the most impact on the final classifier due to both weight regularization and averaging. This is in fact observed when inspecting the weights of MST_{Malt}.

Finally, comparing the results of the guided models to the oracle results discussed in Section 4.4, we see that there should be room for further improvement, as the best guided parser (MST_{Malt}) does not quite reach the level of the graph selection oracle, let alone that of the arc selection oracle. Further exploration of the space of possible systems, as outlined in Section 6.3, will undoubtedly be necessary to close this gap. As already noted, there are several recent developments in data-driven dependency parsing, which can be seen as targeting the specific weaknesses of traditional graph-based and transition-based models, respectively. For graph-based parsers, McDonald and Pereira (2006), Hall (2007), Nakagawa (2007), and Smith and Eisner (2008) attempt to overcome the limited feature scope of graph-based models by adding global features in conjunction with approximate inference. Additionally, Riedel and Clarke (2006) and Martins, Smith, and Xing (2009) integrate global features and maintain exact inference through integer linear programming solutions. For transition-based models, the trend is to alleviate error propagation by abandoning greedy, deterministic inference in favor of beam search with globally normalized models for scoring transition sequences, either generative (Titov and Henderson 2007a, 2007b) or conditional (Duan, Zhao, and Xu 2007; Johansson and Nugues 2007). In addition, Zhang and Clark (2008) has proposed a learning method for transition-based parsers based on global optimization similar to that traditionally used for graph-based parsers, albeit only with approximate inference through beam search, and Huang and Sagae (2010) has shown how a subclass of transition-based parsers can be tabularized to permit the use of dynamic programming.

One question that can be asked, given the correlation provided here between observed errors and algorithmic expectations, is whether it is possible to characterize the errors of a *new* parsing system simply by analyzing its theoretical properties. This is a difficult question to answer. Consider a parsing system that uses greedy inference. One

can speculate that it will result in error propagation and, as a result, a large number of parsing errors on long dependencies as well as those close to the root. However, if the algorithm is run on data that contains only deterministic local decisions and complex global decisions, such a system might not suffer from error propagation. This is because the early local decisions are made correctly. Furthermore, saying something about specific linguistic constructions is also difficult, due to the wide spectrum of difficulty when parsing certain phenomena across languages. Ultimately, this is an empirical question. What we have shown here is that, on a number of data sets, our algorithmic expectations about two widely used dependency parsing paradigms are confirmed.

7. Conclusion

In this article, we have shown that the two dominant approaches to data-driven dependency parsing—global, exhaustive, graph-based models and local, greedy, transition-based models—have distinctive error distributions despite often having very similar parsing accuracy overall. We have demonstrated that these error distributions can be explained by theoretical properties of the two models, in particular related to the fundamental tradeoff between global learning and inference, traditionally favored by graph-based parsers, and a rich feature space, typically found in transition-based parsers. Based on this analysis, we have proposed new directions of research on data-driven dependency parsing, some of which are already beginning to be explored.

We have also demonstrated how graph-based and transition-based models can be integrated by letting one model learn from features generated by the other, using the technique known as stacking in the machine learning community. Our experimental results show that both models consistently improve their accuracy when given access to features generated by the other model, which leads to a significant advancement of the state of the art in data-driven dependency parsing. Moreover, a comparative error analysis reveals that the improvements are predictable from the same theoretical properties identified in the initial error analysis, such as the tradeoff between global learning and inference, on the one hand, and rich feature representations, on the other. On a more general note, we believe that this shows the importance of careful error analysis, informed by theoretical predictions, for the further advancement of data-driven methods in natural language processing.

Acknowledgments

We want to thank our collaborators for great support in developing the parsing technology, the organizers of the CoNLL-X shared task for creating the data, and three anonymous reviewers for their feedback that substantially improved the article.

References

Attardi, Giuseppe. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170, New York, NY.

Attardi, Giuseppe and Massimiliano Ciaramita. 2007. Tree revision learning for dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 388–395, Rochester, NY.

Berger, Adam L., Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–71.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the*

10th Conference on Computational Natural Language Learning (CoNLL), pages 149–164, New York, NY.

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 957–961, Prague.

Chang, Chih-Chung and Chih-Jen Lin. 2001. LIBSVM: A Library for Support Vector Machines. Software available at `www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

Cheng, Yuchang, Masayuki Asahara, and Yuji Matsumoto. 2006. Multi-lingual dependency parsing at NAIST. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 191–195, New York, NY.

Chu, Yoeng-Jin and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

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

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 175–182, Stanford, CA.

Cortes, Corinna and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

Ding, Yuan and Martha Palmer. 2004. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In *Workshop on Recent Advances in Dependency Grammars (COLING)*, pages 90–97, Geneva.

Duan, Xiangyu, Jun Zhao, and Bo Xu. 2007. Probabilistic parsing action models for multi-lingual dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 940–946, Prague.

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

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

Florian, Radu, Hany Hassan, Abraham Ittycheriah, Hongyan Jing, Nanda Kambhatla, Xiaoqiang Luo, Nicolas Nicolov, and Salim Roukos. 2004. A statistical model for multilingual entity detection and tracking. In *Proceedings of Human Language Technology and the Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 1–8, Boston, MA.

Hall, Keith. 2007. K-best spanning tree parsing. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 392–399, Prague.

Hall, Johan, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 933–939, Prague.

Henderson, John C. and Eric Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 188–194, College Park, MD.

Hollingshead, Kristy and Brian Roark. 2007. Pipeline iteration. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 952–959, Prague.

Huang, Liang and David Chiang. 2005. Better *k*-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pages 53–64, Vancouver.

Huang, Liang and Kenjie Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1077–1086, Uppsala.

Hudson, Richard A. 1984. *Word Grammar*. Blackwell, Oxford.

Johansson, Richard and Pierre Nugues. 2007. Incremental dependency parsing using

online learning. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 1134–1138, Prague.
Koo, Terry, Amir Globerson, Xavier Carreras, and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11, Uppsala.
Koo, Terry, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, Prague.
Kudo, Taku and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pages 63–69, Edmonton.
Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 282–289, Williamstown, MA.
Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.
Martins, Andre F. T., Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking dependency parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 157–166, Honolulu, HI.
Martins, Andre F. T., Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pages 342–350, Singapore.
Maruyama, Hiroshi. 1990. Structural disambiguation with constraint propagation. In *Proceedings of the 28th Meeting of the Association for Computational Linguistics (ACL)*, pages 31–38, Pittsburgh, PA.
McDonald, Ryan. 2006. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.
McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98, Ann Arbor, MI.
McDonald, Ryan, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 216–220, New York, NY.
McDonald, Ryan and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, Prague.
McDonald, Ryan and Fernando Pereira. 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)*, pages 81–88, Trento.
McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530, Vancouver.
McDonald, Ryan and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 122–131, Prague.
Mel'čuk, Igor. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.
Nakagawa, Tetsuji. 2007. Multilingual dependency parsing using global features. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 952–956, Prague.
Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160, Nancy.

Nivre, Joakim. 2007. Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 396–403, Rochester, NY.

Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pages 351–359, Singapore.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932, Prague.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning*, pages 49–56, Boston, MA.

Nivre, Joakim, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225, New York, NY.

Nivre, Joakim and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 950–958, Columbus, OH.

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106, Ann Arbor, MI.

Riedel, Sebastian and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 129–137, Sydney.

Riedel, Sebastian, Ruket Çakıcı, and Ivan Meza-Ruiz. 2006. Multi-lingual dependency parsing with incremental integer linear programming. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 226–230, New York, NY.

Sagae, Kenji and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of NAACL: Short Papers*, pages 129–132, New York, NY.

Sarkar, Anoop. 2001. Applying co-training methods to statistical parsing. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 175–182, Pittsburgh, PA.

Sgall, Petr, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel, Dordrecht.

Smith, David A. and Jason Eisner. 2008. Dependency parsing by belief propagation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 145–156, Honolulu, HI.

Smith, David A. and Noah A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, Prague.

Snow, Rion, Dan Jurafsky, and Andrew Y. Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1297–1304, Vancouver.

Steedman, Mark, Rebecca Hwa, Miles Osborne, and Anoop Sarkar. 2003. Corrected co-training for statistical parsers. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 95–102, Washington, DC.

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

Taskar, Ben, Simon Lacoste-Julien, and Dan Klein. 2005. A discriminative matching approach to word alignment. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 73–80, Vancouver.

Titov, Ivan and James Henderson. 2007a. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 947–951, Prague.

Titov, Ivan and James Henderson. 2007b. A latent variable model for generative

dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 144–155, Prague.

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

Zeman, Daniel and Zdeněk Žabokrtskỳ. 2005. Improving parsing accuracy by combining diverse dependency parsers. *Proceedings of the International Workshop on Parsing Technologies*, pages 171–178, Vancouver.

Zhang, Yue and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–571, Honolulu, HI.