Language and Linguistics Compass 4/3 (2010): 138–152, 10.1111/j.1749-818x.2010.00187.x

Dependency Parsing

Joakim Nivre*
Department of Linguistics and Philology, Uppsala University

Abstract

Dependency parsing is a form of syntactic parsing of natural language based on the theoretical tradition of dependency grammar. It has recently gained widespread interest in the computational linguistics community and has been used for applications such as information extraction, machine translation and question answering. In this article, we review the current state of the art in dependency parsing, starting with a characterization of the basic problem and some motivation for why dependency parsing may be a useful alternative to other forms of syntactic parsing. The major part of the article is devoted to an examination of the four main approaches that exist in the field today: context-free dependency parsing, constraint dependency parsing, graph-based dependency parsing, and transition-based dependency parsing. Each approach is described in detail and its strengths and weaknesses are highlighted.

1. Introduction

In syntactic parsing of natural language, we analyze sentences by automatically constructing representations of their syntactic structure. Many different types of representations have been proposed for this purpose, but in this article we focus our attention on representations based on the notion of syntactic dependency. This form of representation, which comes out of a long tradition of theoretical work in dependency grammar, has recently gained widespread interest in the computational linguistics community and has been used in a number of practical applications, such as information extraction, machine translation and question answering. In parallel with this development, we have seen the emergence of new methods for parsing with dependency representations, or *dependency parsing* for short.

In this article, we examine the *what*, *why* and *how* of dependency parsing. In Section 2, we explain what dependency parsing is, introducing the notion of dependency structure and defining the task of dependency parsing more precisely. In Section 3, we provide some motivation for why dependency parsing is a useful technique in natural language processing. In the following four sections, we show how dependency parsing can be implemented in practice, discussing four major approaches to the task: context-free dependency parsing (Section 4), constraint dependency parsing (Section 5), graph-based dependency parsing (Section 6), and transition-based dependency parsing (Section 7). We conclude in Section 8.

It is worth noting that this article only covers dependency parsing in a narrow sense, that is, systems for mapping a sentence to its dependency structure. This definition excludes, for example, parsing systems that make use of dependency relations in the derivation of a different kind of representation, such as the head-driven statistical phrase structure parsers of Collins (1999) and Charniak (2000). It also excludes parsers for richer grammatical frameworks that incorporate notions related to dependency, such as the derivation trees of

Tree Adjoining Grammar (Joshi 1985, 1997) or Combinatory Categorial Grammar (Steedman 2000), or the f-structures of Lexical Functional Grammar (Kaplan and Bresnan 1982; Bresnan 2000). Although these topics could be said to fall under the notion of dependency parsing in a broad sense, they are outside the scope of this article.

2. Dependency Structure

Dependency grammar has a long history, possibly going back all the way to Pāṇini's grammar of Sanskrit several centuries before the Common Era, and has largely developed as a form for syntactic representation used by descriptive grammarians, in particular in Europe, and especially for Classical and Slavic languages. The starting point of the modern theoretical tradition of dependency grammar is usually taken to be the work of the French linguist Lucien Tesnière, published posthumously in the late 1950s (Tesnière 1959). Since then, a number of different dependency grammar frameworks have been proposed, of which the most well known are probably the Prague School's Functional Generative Description (Sgall et al. 1986), Mel'čuk's Meaning-Text Theory (Mel'čuk 1988; Milicevic 2006) and Hudson's Word Grammar (Hudson 1984, 1990, 2007).

Different theoretical frameworks make different assumptions about the finer details of syntactic structure, but the basic assumption underlying all varieties of dependency grammar is the idea that syntactic structure essentially consists of *words* linked by binary, asymmetrical relations called *dependency relations* (or simply *dependencies*). A dependency relation holds between a syntactically subordinate word, called the *dependent*, and another word on which it depends, called the *head*.[1] This is illustrated in Figure 1, which shows a dependency structure for a simple English sentence, where dependency relations are represented by arrows pointing from the head to the dependent.[2] Moreover, each arrow has a label, indicating the *dependency type*. For example, the noun *news* is a dependent of the verb *had* with the dependency type *subject* (SBJ). By contrast, the noun *effect* is a dependent of type *object* (OBJ) with the same head verb *had*. Note also that the noun *news* is itself a syntactic head in relation to the word *Economic*, which stands in the *attribute* (ATT) relation to its head noun.

The information encoded in a dependency structure representation is different from the information captured in a *phrase structure* representation, which is the most widely used type of syntactic representation in both theoretical and computational linguistics. This can be seen by comparing the dependency structure in Figure 1 to a typical phrase structure representation for the same sentence, shown in Figure 2. While the dependency structure represents head-dependent relations between *words*, classified by *functional* categories such as subject (SBJ) and object (OBJ), the phrase structure represents the grouping of words into *phrases*, classified by *structural* categories such as noun phrase (NP) and verb phrase (VP). However, it is important to bear in mind that these differences only concern what is explicitly encoded in the respective representations. For example, phrases can be

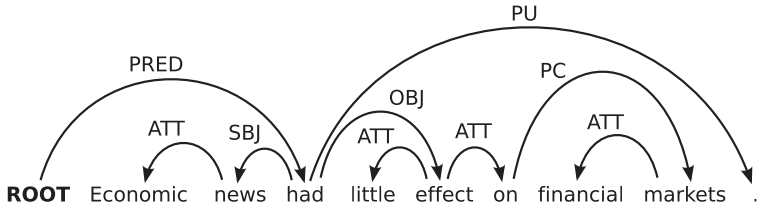

Fig 1. Dependency structure for an English sentence.

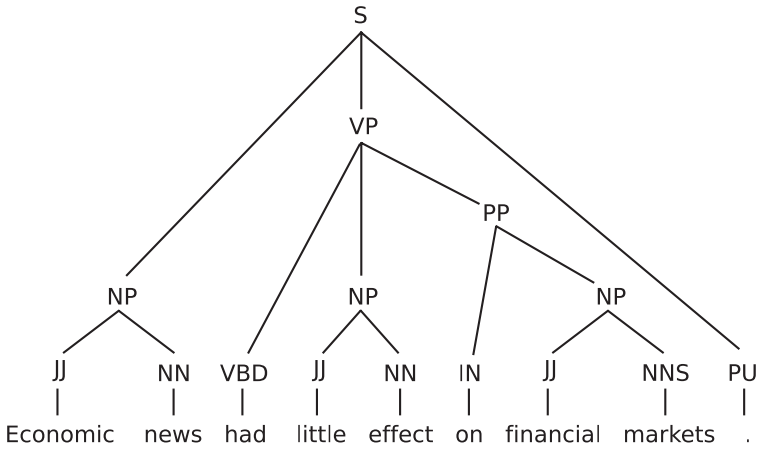

Fig 2. Phrase structure for an English sentence.

distinguished in a dependency structure by letting each head word represent a phrase consisting of the word itself and all the words that are dependent on it (possibly in several steps). Conversely, functional relations like subject and object can be identified in a phrase structure in terms of structural configurations (e.g., ''NP under S'' and ''NP under VP'').

One peculiarity of the dependency structure in Figure 1 is that there is an artificial word ROOT before the first word of the sentence. This is a mere technicality, which simplifies both formal definitions and computational implementations. In particular, we can normally assume that every real word of the sentence should have a syntactic head. Thus, instead of saying that the verb *had* lacks a syntactic head, we can say that it is a dependent of the artificial word ROOT. In this article, we therefore assume that the dependency structure of a sentence $S = w_1,\ldots,w_n$ is a tree rooted in the artificial word ROOT prefixed to the sentence. Formally speaking, such a tree is a pair $T = (V,A)$, consisting of a set V of nodes, each corresponding to a word indexed by its position in the sentence (including the word ROOT with index 0), and a set A of labeled directed arcs, that is, triples (w_i,l,w_j), where w_i and w_j are nodes in V and l is an arc label.[3] For example, the dependency tree depicted in Figure 1 corresponds to the following sets of nodes and arcs:

$$V = \{\text{root}_0, \text{Economic}_1, \text{news}_2, \text{had}_3, \text{little}_4, \text{effect}_5, \text{on}_6, \text{financial}_7, \text{markets}_8, .{}_9\}$$

$$\begin{aligned} A = \{&(\text{root}_0, \text{pred}, \text{had}_3), (\text{news}_2, \text{att}, \text{Economic}_1), (\text{had}_3, \text{sbj}, \text{news}_2),\\ &(\text{effect}_5, \text{att}, \text{little}_4), (\text{had}_3, \text{obj}, \text{effect}_5), (\text{effect}_5, \text{att}, \text{on}_6),\\ &(\text{markets}_8, \text{att}, \text{financial}_7), (\text{on}_6, \text{pc}, \text{markets}_8), (\text{had}_3, \text{pu}, .{}_9)\} \end{aligned}$$

Assuming that a dependency structure is always a tree implies that every word (except the artificial word ROOT) has a single syntactic head and that the structure is acyclic. Although this assumption is not universally accepted in the theoretical tradition of dependency grammar, it usually holds for practical parsing systems and will be taken for granted in the rest of this article. Some frameworks in addition assume that dependency trees should be *projective*, which means that all subtrees consist of nodes corresponding to a contiguous substring of the sentence. The tree in Figure 1 is projective, but the tree in Figure 3 is not, because it contains two subtrees violating the condition of projectivity, the subtree rooted at the node *hearing* (corresponding to the discontinuous noun phrase *A hearing on the issue*) and the subtree rooted at *scheduled* (corresponding to the discontinuous

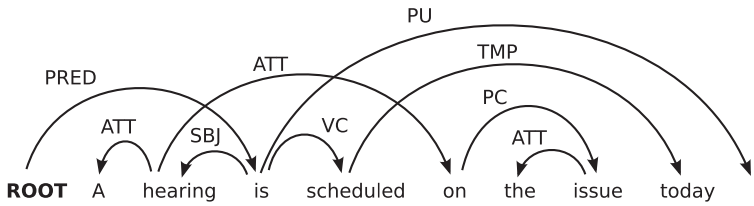

Fig 3. Non-projective dependency tree for an English sentence.

verb phrase *scheduled today*). Even though the overwhelming majority of syntactic structures are projective in most languages, most researchers today agree that strictly projective trees are insufficient for an adequate representation of syntactic structure.

We can now define the task of *dependency parsing* as the task of mapping an input sentence $S = w_1,\ldots,w_n$ to one or more dependency trees $T = (V,A)$. In the rest of the article we will examine different methods for performing this task. Some of these make use of a formal grammar specifying the relation between sentences and dependency trees, others make use of statistics from syntactically annotated corpora (treebanks), either in addition to or instead of a grammar. Issues concerning the substantive analysis of syntactic structure, for example, what should be the head in a coordinated structure such as *ships and banks*, will be decided by the specific grammar or corpus annotation guidelines, whereas the parsing methods are completely general and only assume that the output should be a dependency tree (possibly restricted to be projective). However, before we dive into the technical details of dependency parsing, we will say a few words about why dependency parsing may be an interesting alternative to other approaches to syntactic parsing.

3. The Case for Dependency Parsing

Dependency parsing has in recent years established itself as a standard technique in natural language processing, and dependency-based representations have been used in a wide range of different applications, including language modeling (Chelba et al. 1997), information extraction (Culotta and Sorensen 2004), machine translation (Ding and Palmer 2004; Quirk et al. 2005), textual entailment (Haghighi et al. 2005), ontology learning (Snow et al. 2005), and question answering (Wang et al. 2007). Assuming that all these applications can make use of the output from a syntactic parser, we may ask whether there is any reason to prefer one type of representation over another. We believe that dependency parsing has certain properties that make it particularly useful in this context.

One of the advantages of dependency representations is that they provide a transparent encoding of predicate–argument structure, which appears to be useful in many applications that make use of syntactic parsing. This can be seen by comparing the two representations in Figures 1 and 2. Whereas the head word of the grammatical subject (*news*) is connected directly to the verb (*had*) in Figure 1, the same relation is mediated by a long path of non-terminal nodes (NN–NP–S–VP–VBD) in Figure 2. In a similar fashion, there is an arc labeled OBJ connecting *had* to *effect* in Figure 1, but only an indirect connection (VBD–VP–NP–NN) in Figure 2. The direct links from predicates to their arguments also make dependency representations more suitable for languages with free or flexible word order, where the indirect encoding of grammatical functions in phrase structure representations can become very complex. This point has been made in the theoretical literature on dependency grammar (Mel'čuk 1988), but has also been demonstrated in the multilingual evaluation campaigns for dependency parsers organized

by the Conference on Computational Natural Language Learning (CoNLL) (Buchholz and Marsi 2006; Nivre et al. 2007), where the results suggest that dependency parsers are more accurate than phrase structure parsers for several languages, although it is not straightforward to compare results obtained with different representations.

The direct encoding of dependency relations between syntactic head words not only makes representations more transparent, it also reduces the complexity of the parsing task, which is potentially beneficial for both parsing efficiency and parsing accuracy. In essence, constructing a dependency tree for a sentence $S = w_1,\ldots,w_n$ consists in assigning to each word w_i a syntactic head $h(w_i)$ and a dependency label $d(w_i)$, which is a much more constrained task than building a phrase structure tree with (in principle) arbitrarily many non-terminal nodes. The lower complexity of dependency parsing can be exploited by constructing parsers with a better trade-off between accuracy and efficiency. For example, deterministic linear-time parsing (discussed in Section 7) is feasible with state-of-the-art accuracy for dependency parsing (Yamada and Matsumoto 2003; Nivre et al. 2006) but not for phrase structure parsing (Sagae and Lavie 2005, 2006a); and global discriminative training with dynamic programming (discussed in Section 6) is practically feasible for dependency parsing but too inefficient to be practically useful for lexicalized phrase structure parsing (McDonald et al. 2005a,b).

In sum, we may say that dependency trees provide a transparent encoding of predicate–argument structure *despite* their constrained nature, and that they support efficient and accurate parsing *thanks to* their constrained nature. The result is a good balance between expressivity and complexity, which makes dependency parsing a natural choice in applications where these representations are sufficiently informative. We will now have a look at the methods used to implement dependency parsing in practice.

4. Context-Free Dependency Parsing

The earliest work on parsing with dependency representations was intimately tied to formalizations of dependency grammar that were close to context-free grammar (CFG) (Chomsky 1956), such as the proposals of Hays (1964) and Gaifman (1965). This approach is based on the observation that any projective dependency tree can be mapped to an equivalent phrase structure tree in which non-terminal symbols are indexed by words. This is illustrated in Figure 4, which shows a projective dependency tree for an English sentence (left) and the corresponding phrase structure tree (right).

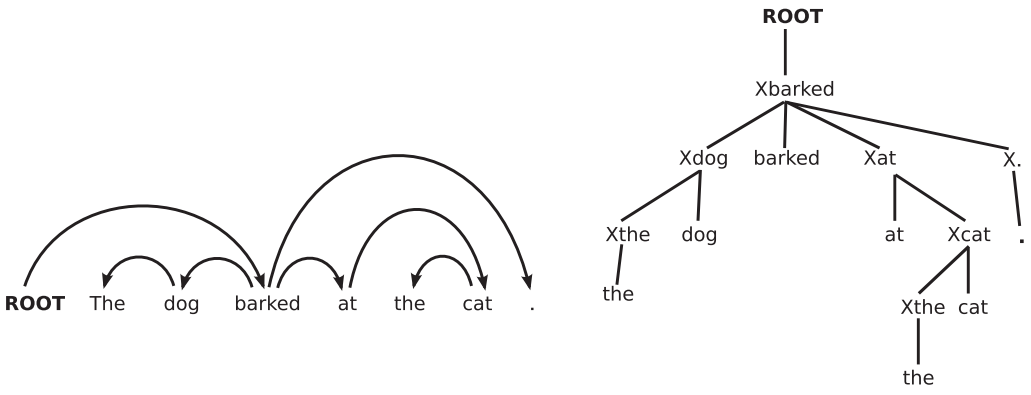

Fig 4. Projective dependency tree mapped to phrase structure tree.

The mapping is easiest described in terms of subtrees, i.e., a node together with all its transitive dependents. Every subtree rooted at a node *w* other than ROOT, with direct arcs to subtrees $t_1,\ldots,t_n$, is mapped to a subtree consisting of a non-terminal node labeled X*w* dominating a terminal node labeled *w* and subtrees corresponding to $t_1,\ldots,t_n$ (in the order given by the original word order). For example, the subtree rooted at *at* in the dependency tree, corresponds to a subtree with root labeled X*at* dominating a terminal node labeled *at* and a smaller subtree corresponding to the dependency subtree rooted at *cat*. As a special case, the root node of the phrase structure tree is labeled ROOT and dominates subtrees corresponding to dependents of ROOT in the dependency tree. We have omitted dependency labels in this example, to focus on the structural correspondence, but it is straightforward to add dependency labels, e.g., by augmenting the labels of non-head child nodes. In this scheme, the non-terminal node labeled X*dog* in Figure 4 would be extended to X*dog*:SBJ, specifying that the word *dog* has the subject relation to its head *barked*.

Given the correspondence between dependency trees and phrase structure trees, we can now use an ordinary CFG to generate phrase structure trees that represent dependency structures. For example, the following grammar generates the tree in Figure 4:[4]

ROOT → X*barked*
X*barked* → X*dog barked* X*at* X.
X*dog* → X*the dog*
X*at* → *at* X*cat*
X*cat* → X*the cat*
X*the* → *the*
X. → .

This grammar satisfies two important restrictions. First, in addition to the start symbol ROOT, the set of non-terminal symbols contains exactly one symbol (X*w*) for every terminal symbol (*w*). The grammar is therefore said to be lexicalized, as grammatical categories (non-terminal symbols) are indexed by lexical items (terminal symbols). Secondly, every production where the left-hand side is not the start symbol ROOT has exactly one terminal symbol in the right-hand side, which is the terminal *w* corresponding to the non-terminal X*w* in the left-hand side. A CFG satisfying these restrictions is sometimes referred to as a *context-free dependency grammar*, because it generates trees that are in one-to-one correspondence with dependency trees.

One advantage of generating dependency trees indirectly using a CFG is that all the well-studied parsing algorithms for such grammars, such as CKY (Younger 1967) or Earley's algorithm (Earley 1970), can be used for dependency parsing. Moreover, adding a statistical model for ranking alternative analyses of a given sentence in terms of their probability is easily achieved by extending the grammar to a *probabilistic* CFG (PCFG) (Booth and Thompson 1973). In a PCFG, each grammar rule is assigned a probability such that the probabilities for all rules with the same left-hand side sum to 1. The PCFG model for statistical parsing has been studied extensively over the last two decades, and a wide range of different methods have been proposed both for learning rule probabilities from annotated corpora and for deriving the most probable analysis given a grammar and an input sentence. Representing dependency grammars as PCFGs makes all this research applicable to dependency parsing as well as to phrase structure parsing.

However, research has also shown that it is sometimes possible to improve methods inherited from (P)CFG parsing by exploiting the special restrictions of dependency parsing. For example, one problem with lexicalized grammars is that the worst-case time

complexity of dynamic programming algorithms like CKY and Earley's algorithm in practice becomes $O(n^5)$ instead of $O(n^3)$ with a naive implementation, where n is the length of the input sentence.[5] However, Eisner (1996, 2000) showed that, for context-free dependency grammars, the complexity can be improved to $O(n^3)$ by transforming the grammar so that left dependents and right dependents are processed independently in a so-called bilexical grammar. In addition, Eisner (1996) demonstrated how a bilexical grammar can be coupled with a history-based probability model, more powerful than the standard PCFG model and widely used in phrase structure parsing (Collins 1997, 1999; Charniak 2000), without sacrificing parsing efficiency.

In sum, by reducing dependency parsing to context-free parsing we can tap into the wealth of research results on (P)CFG parsing, both regarding efficient parsing algorithms and methods for learning statistical models from data. However, this approach to dependency parsing comes with one severe restriction – it is limited to strictly projective dependency trees. This restriction will be lifted as we move on to dependency parsing based on constraint satisfaction.

5. Constraint Dependency Parsing

A context-free dependency grammar is a set of rules for deriving a sentence $S = w_1,\ldots,w_n$ from the start symbol ROOT, and each distinct derivation of S from ROOT defines a valid dependency tree T. A *constraint dependency grammar* is instead a set of boolean constraints on well-formed dependency trees, such as the constraint that a singular count noun must have a determiner, or the constraint that a preposition must precede its complement. Parsing with such a grammar is a constraint satisfaction problem, and any dependency tree T that satisfies all constraints is a valid analysis. This is sometimes called *eliminative* or *reductionist* parsing, because it starts, at least conceptually, from the set of all possible dependency trees for a given sentence S and successively eliminates all trees that violate some constraint until only the valid analyses remain.

The notion of constraint dependency parsing was first proposed by Maruyama (1990), starting from the observation that a dependency tree T for a sentence S is equivalent to an assignment of a head node $h(w_i)$ and a dependency label $d(w_i)$ to each word w_i of S, as explained earlier in Section 2. If we regard each $h(w_i)$ as a variable with domain $V - \{w_i\}$ (i.e., the set of possible values is the set of all nodes distinct from w_i) and each $d(w_i)$ as a variable whose domain is the set of all possible dependency labels, then the parsing problem consists in finding an assignment of values to variables that respects all constraints of the grammar.

One of the advantages of the constraint satisfaction approach is that it is not inherently limited to projective dependency trees. Given our formulation of the problem, any valid variable assignment will define a tree rooted at the artificial root node, but the tree does not have to be projective unless explicit constraints are added to ensure this. For example, here is the assignment corresponding to the non-projective tree in Figure 3:

$$
\begin{array}{llllll}
h(\mathrm{A}_1) & = & \mathrm{hearing}_2 & d(A_1) & = & \textsc{det} \\
h(\mathrm{hearing}_2) & = & \mathrm{is}_3 & d(\mathrm{hearing}_2) & = & \textsc{sbj} \\
h(\mathrm{is}_3) & = & \textsc{root}_0 & d(\mathrm{is}_3) & = & \textsc{pred} \\
h(\mathrm{scheduled}_4) & = & \mathrm{is}_3 & d(\mathrm{scheduled}_4) & = & \textsc{vc} \\
h(\mathrm{on}_5) & = & \mathrm{hearing}_2 & d(\mathrm{on}_5) & = & \textsc{att} \\
h(\mathrm{the}_6) & = & \mathrm{issue}_7 & d(\mathrm{the}_6) & = & \textsc{det} \\
h(\mathrm{issue}_7) & = & \mathrm{on}_5 & d(\mathrm{issue}_7) & = & \textsc{pc}
\end{array}
$$

$h(\text{today}_8)$	=	scheduled_4	$d(\text{today}_8)$	=	ADV
$h(.9)$	=	is_3	$d(.9)$	=	PU

However, this added flexibility comes at a price because constraint satisfaction in general is an NP complete problem, which means that special care must be taken to ensure that parsing can be performed efficiently in practice. Early versions of constraint dependency parsing therefore used procedures based on local consistency, which achieve polynomial-time parsing by only considering local information in the application of constraints (Maruyama 1990; Harper and Helzerman 1995), while more recent approaches have tried to confront the problem head-on by using constraint programming to solve the satisfaction problem defined by the grammar for a given input sentence (Duchier 1999, 2003).

Another problem that is inherent in the original version of constraint dependency parsing, as in all grammar-based approaches to parsing, is the fact that, for a given input sentence, there may be no analysis satisfying all constraints, which undermines the robustness of the parser, and there may be more than one analysis, which points to the need for disambiguation or parse selection. To deal with this problem, Menzel and Schröder (1998) extended the framework of Maruyama (1990) with *weighted* constraints. The idea is that each constraint is assigned a weight k, ranging from 0.0 to 1.0, indicating how serious the violation of this constraint is. A weight of 0.0 indicates a hard constraint, that cannot be violated without making the sentence ungrammatical, e.g., a constraint enforcing subject–verb agreement in a language where agreement is obligatory, while weights closer to 1.0 indicate weaker preferences, e.g., a constraint that the subject should precede the verb in a language that does not have fixed word order. The weight of a dependency tree T is then defined as the product of the weights of all constraints that are violated by T, and the optimal tree for a given sentence is the tree with the highest total weight.

While early implementations of this system used an eliminative approach to parsing (Menzel and Schröder 1998), more recent versions use a transformation-based approach, which successively tries to improve the analysis by transforming one solution into another guided by the observed constraint violations in the current solution (Foth et al. 2004). One advantage of this heuristic approximation strategy is that it can be combined with arbitrarily complex constraints, whereas standard eliminative procedures usually require constraints to be binary for efficiency reasons. In addition, this parsing algorithm has the so-called anytime property, which means that it can be interrupted at any time and still return a valid dependency tree, which is useful for systems that operate under hard time constraints.

6. Graph-Based Dependency Parsing

The *graph-based* approach to dependency parsing is similar in spirit to weighted constraint dependency parsing in that it attempts to find the highest scoring dependency tree in the space of all possible trees for a given sentence. The essential difference is that no grammatical constraints at all are assumed. Instead, the search is guided by a function for scoring dependency trees in terms of their probability, a function that is induced from syntactically annotated corpora (or treebanks) using machine learning. In this sense, graph-based dependency parsing is a purely data-driven method, which makes no use of a formal grammar, unlike the approaches discussed so far.

At the heart of graph-based parsing systems is the notion of the score $\mathscr{F}(T)$ of a dependency tree T for a sentence S, indicating how likely it is that T is the correct analysis for S. This score can be defined in many different ways, but the fundamental property

of graph-based parsing systems is that $\mathscr{F}(T)$ is assumed to be a function of the scores $\mathscr{F}(G_i)$ of some set of subgraphs $G_1,\ldots,G_m$ of T. The most common model is to define the global score as the sum of the scores of all relevant subgraphs:

$$\mathscr{F}(T) = \sum_{i=1}^{m} \mathscr{F}(G_i)$$

This kind of model goes back to the work of Eisner (1996), who defined probability models for dependency parsing that were factored by subgraphs.[6]

One important decision in the design of a graph-based parsing model is how to factor a tree into subgraphs. The simplest approach is the so-called *arc-factored* model, where each dependency arc (w_i,l,w_j) of the tree $T = (V,A)$ is treated as a separate component in the scoring model.[7] This gives the following scoring model:

$$\mathscr{F}(T) = \mathscr{F}(V,A) = \sum_{(w_i,l,w_j)\in A} \mathscr{F}(w_i,l,w_j)$$

McDonald et al. (2005a,b) showed that, despite its simplicity, this model can achieve state-of-the-art parsing accuracy with a suitable choice of features for learning the scoring function for arcs from data. Moreover, McDonald et al. (2005a,b) showed that graph-theoretic algorithms can be used to find the best scoring dependency tree very efficiently during parsing. More precisely, finding the highest scoring dependency tree in this model is equivalent to finding the maximum spanning tree in a dense directed graph containing all possible arcs, a problem that can be solved in $O(n^2)$ time using the Chu–Liu–Edmonds algorithm (Chu and Liu 1965; Edmonds 1967). As there is nothing in this formulation that requires the spanning tree to be projective in the dependency grammar sense, this in fact constitutes an efficient parsing algorithm for arbitrary dependency trees, projective as well as non-projective.

Although the arc-factored model has given impressive results and is still widely used, later research has shown that parsing accuracy can be improved by factoring dependency trees into larger subgraphs, so-called *higher order* models (McDonald and Pereira 2006; Carreras 2007). In the model of McDonald and Pereira (2006), the score of an arc (w_i,l,w_j) is conditioned on sibling arcs (w_i,l',w_k), as in the original model of Eisner (1996), while the model of Carreras (2007) takes even larger substructures into account. Unfortunately, if we abandon the arc-factored assumption, then parsing with arbitrary dependency trees is no longer feasible in polynomial time (McDonald and Pereira 2006; McDonald and Satta 2007); so, higher order graph-based models are typically combined with Eisner's algorithm for projective dependency parsing (Eisner 1996, 2000), sometimes with a post-processing step to recover non-projective dependency arcs (McDonald and Pereira 2006).

The parsing accuracy achieved with a graph-based model depends crucially on the way the scoring function $\mathscr{F}(G)$ for subgraphs is learned from treebank data. Graph-based parsing systems typically assume that the scoring function is a linear classifier between a k-dimensional vector $\mathbf{f}(G)$ of numerical features over the subgraph G (and the input sentence S) and a corresponding real-valued weight vector $\mathbf{w}$:

$$\mathscr{F}(G) = \mathbf{w} \cdot \mathbf{f}(G) = \sum_{i=1}^{k} \mathbf{w}_i \cdot \mathbf{f}_i(G)$$

where $\mathbf{w}_i$ and $\mathbf{f}_i(G)$ are the ith component of the weight vector and feature vector, respectively. The feature vector $\mathbf{f}$ can include any relevant feature over the subgraph G

or the input sentence S, but features are typically categorical and *post hoc* converted to binary features. For example, consider the arc (hearing$_2$, ATT, on$_5$) from Figure 3. A feature representation might consist of the following categorical features:

Identity of w_i = *hearing*
Identity of w_j = *on*
Identity of part-of-speech tag for w_i = nn
Identity of part-of-speech tag for w_j = in
Identity of l = att
Identity of part-of-speech tags between w_i and w_j = vbz, vbn
Identity of part-of-speech tag for w_{i-1} = dt
Identity of part-of-speech tag for w_{i+1} = vbz
Identity of part-of-speech tag for w_{j-1} = vbn
Identity of part-of-speech tag for w_{j+1} = dt
Distance (in number of words) between w_i and w_j = 2
Direction of arc = RIGHT

These features can then be combined to create more complex categorical features such as:

Identity of w_i = *hearing* & Identity of w_j = *on* & Direction of arc = RIGHT

Given a definition of the feature vector $\mathbf{f}(G)$, the learning problem for a graph-based parser consists in estimating the weight vector $\mathbf{w}$ in order to maximize parsing accuracy on unseen sentences. This problem can be solved in many ways, but the standard approach in graph-based dependency parsing, pioneered by McDonald et al. (2005a,b), is to use *inference-based learning*, e.g., the perceptron algorithm (Rosenblatt 1958; Collins and Duffy 2002), or a large-margin version of the perceptron (McDonald et al. 2005a,b; Crammer et al. 2006). These algorithms build a linear classifier online by considering a single training sentence S in isolation, finding the highest scoring tree T for S given the current weight vector, comparing it with the treebank analysis T^*, and then, if $T \neq T^*$, adding weight to features that are present in T^* while subtracting weight from features that are present in T. In this way, it is possible to learn a weight vector that favors globally optimal dependency trees, as opposed to locally optimal substructures, which is one of the strengths of the graph-based approach to dependency parsing (McDonald and Nivre 2007).

7. *Transition-Based Dependency Parsing*

The *transition-based* approach to dependency parsing is similar to graph-based parsing in that it is a purely data-driven method that makes no use of a formal grammar but relies on machine learning from treebank data. The difference is that, whereas a graph-based parser learns to score dependency trees, factored into subgraphs, a transition-based parser learns to score possible next actions in a state machine for deriving dependency trees. The state machine is called a *transition system*, and the possible actions are called *transitions*.

A transition system for dependency parsing consists of a set C of configurations, representing partial analyses of sentences, and a set D of transitions from configurations to new configurations. For every sentence $S = w_1,\ldots,w_n$, there is a unique initial configuration $c_s(S) \in C$ and a set $C_t(S) \subseteq C$ of terminal configurations, each representing a complete analysis T of S.

For example, if we let a configuration be a triple $c = (\sigma,\beta,A)$, where σ is a stack of nodes, β is a buffer of remaining input nodes, and A is a set of labeled dependency arcs, then we can define a transition system for dependency parsing as follows:

- The initial configuration $c_i(S) = ([w_0],[w_1,\ldots,w_n],\varnothing)$
- The set of terminal configurations $C_t(S) = \{c \in C \mid c = ([w_0],[],A)\}$
- The set D of transitions include:

1. SHIFT: $(\sigma,[w_i \mid \beta],A) \Rightarrow ([\sigma \mid w_i],\beta,A)$
2. RIGHT-ARC(l): $([\sigma \mid w_i,w_j],\beta,A) \Rightarrow ([\sigma \mid w_i],\beta,A \cup \{(w_i,l,w_j)\})$
3. LEFT-ARC(l): $([\sigma \mid w_i,w_j],\beta,A) \Rightarrow ([\sigma \mid w_j],\beta,A \cup \{(w_j,l,w_i)\})$

The initial configuration has a stack containing only the special root node ($w_0 = \textsc{root}$), an empty arc set, and all the input words $w_1,\ldots,w_n$ in the buffer. A terminal configuration has the root node on the stack and an empty buffer. The SHIFT transition moves the next node in the buffer onto the stack, while the RIGHT-ARC(l) and LEFT-ARC(l) transitions add a dependency arc between the two top nodes on the stack and replace them by the head node of that arc. It is easy to show that, for any sentence $S = w_1,\ldots,w_n$ with a projective dependency tree T, there is a transition sequence that builds T in exactly $2n$ steps starting from $c_s(S)$. Over the years, a number of different transition systems have been proposed for dependency parsing, some of which are restricted to projective dependency trees (Kudo and Matsumoto 2002; Nivre 2003; Yamada and Matsumoto 2003), while others can also derive non-projective structures (Attardi 2006; Nivre 2006, 2007).

Given a scoring function $\mathscr{F}(\mathbf{f}(c), d)$, which scores possible transitions d out of a configuration c, represented by a k-dimensional feature vector $\mathbf{f}(c)$, and given a way of combining the scores of individual transitions into scores for complete sequences, parsing can be performed as search for the highest scoring transition sequence. Different search strategies are possible, but most transition-based dependency parsers implement some form of beam search, with a fixed constant beam width k, which means that parsing can be performed in $O(n)$ time for transition systems where the length of a transition sequence is linear in the length of the sentence. In fact, many systems set k to 1, which means that parsing is completely deterministic given the scoring function. Linear time parsing is feasible for projective dependency trees and subsets of non-projective trees (Attardi 2006), while transition-based parsers for arbitrary non-projective parsers run in $O(n^2)$ time (Nivre 2008).

The feature vector $\mathbf{f}(c)$ can include any relevant feature of the configuration c, but the most important features in transition-based parsing are defined by attributes of input words, identified by their position in the configuration. It is often convenient to think of these features as defined by two simpler functions, an *address function* identifying a particular word in a configuration (e.g., the word on top of the stack) and an *attribute function* selecting a specific attribute of this word (e.g., its part of speech). Typical address functions extract nodes near the top of the stack, and dependents of such nodes in the partially built dependency tree, as well as lookahead tokens near the front of the buffer. Typical attribute functions refer to linguistic properties of words, which may be given as input to the parser or computed as part of the parsing process. We can exemplify this with the word *markets* from the sentence in Figure 1:

Identity of w_i = *markets*
Identity of part-of-speech tag for w_i = nns
Identity of dependency label for w_i = pc

The first two attributes are *static* in the sense that they are constant, if available at all, in every configuration for a given sentence. By contrast, the dependency label attribute is *dynamic* in the sense that it is available only after the relevant dependency arc has been added to the arc set. Hence, such attributes can be used to define features of the transition history and the partially built dependency tree, which turns out to be one of the major advantages of the transition-based approach.

Once the feature vector $\mathbf{f}(c)$ has been defined, the scoring function $\mathscr{F}(\mathbf{f}(c), d)$ can be learned using any one of a number of machine learning algorithms, but the method of choice in recent years has been support vector machines with polynomial kernels, which give state-of-the-art accuracy in combination with different transition systems (Yamada and Matsumoto 2003; Nivre et al. 2006; Nivre 2008). Other learning methods used include memory-based learning (Nivre et al. 2004; Attardi 2006), logistic regression (Cheng et al. 2005; Attardi 2006) and perceptron learning (Ciaramita and Attardi 2007; Zhang and Clark 2008). Regardless of learning method, the main advantage of the transition-based approach is that the scoring function learned can take a very rich set of features into account—without the subgraph restriction of graph-based models—although the model is only optimized with respect to local decisions—as opposed to global structures. This illustrates a fundamental trade-off between global learning with local features, which is favored in graph-based parsing, and local learning with global features, which is typical of transition-based parsing (McDonald and Nivre 2007).

8. Conclusion

In this article, we have presented dependency parsing as an approach to natural language parsing that analyzes syntactic structure by connecting the words of a sentence into a dependency tree. We have argued that this type of representation is useful in many practical applications of natural language processing and has the further advantage that it can be processed both efficiently and accurately. And we have examined the most important techniques that are used in the field: context-free dependency parsing, constraint dependency parsing, graph-based dependency parsing, and transition-based dependency parsing. This is hardly an exhaustive characterization. First of all, there are hybrid approaches that attempt to combine the strengths of one or more of the basic techniques. This includes recent work on integrating graph-based and transition-based parsers (Nivre and McDonald 2008; Torres Martins et al. 2008; Zhang and Clark 2008), as well as the work of Sagae and Lavie (2006b) that use the graph-based model as a framework for combining the predictions of a set of parsers that can be of any type. In addition, there are parsing systems that are very similar in spirit to dependency parsing without fitting exactly into the categories used here, such as the Link Grammar parser (Sleator and Temperley 1991, 1993) and the MINIPAR system (Lin 2003). Still, by focusing on the currently prevalent paradigms, we hope to have provided a useful overview and some insight into the challenges of dependency parsing.

Short Biography

Joakim Nivre is Professor of Computational Linguistics at Uppsala University (since 2008) and at Växjö University (since 2002). He holds a Ph.D. in General Linguistics from the University of Gothenburg and a Ph.D. in Computer Science from Växjö University. Joakim's research focuses on data-driven methods for natural language processing, in

particular for syntactic and semantic analysis. He is one of the main developers of the transition-based approach to data-driven dependency parsing, described in his 2006 book *Inductive Dependency Parsing* and implemented in the MaltParser system. Systems developed using MaltParser were tied for first place in the shared tasks on multilingual dependency parsing at the Conference on Computational Natural Language Learning in both 2006 and 2007. Joakim's current research interests include the analysis of mildly non-projective dependency structures, the integration of morphological and syntactic processing for richly inflected languages, and the modeling of human sentence processing.

Notes

* Correspondence address: Joakim Nivre, Uppsala University, Department of Linguistics and Philology. E-mail: joakim.nivre@lingfil.uu.se

[1] Other terms that are found in the literature are *modifier* or *child*, instead of *dependent*, and *governor*, *regent* or *parent*, instead of *head*.

[2] This is the notational convention that we adopt in this article, but the reader should be warned that there is a competing tradition in the literature on dependency grammar according to which arrows point from the dependent to the head.

[3] We will omit indices on words when it is clear from the context which node is referred to.

[4] We ignore the fact that the word *the* is capitalized when occurring in sentence-initial position.

[5] The worst-case complexity of these algorithms is really $O(n^3|G|)$, where $|G|$ is the number of rules in the grammar, which in a lexicalized grammar is normally much larger than n^2 for ordinary sentences. By restricting the grammar to rules that involve non-terminals indexed by words that actually occur in the sentence, the complexity can be improved to $O(n^5)$.

[6] In an ordinary probability model, the probability of T would be the *product* of the probabilities of all subgraphs G_i, rather than the *sum*, but it is easy to transform this into a summative model by taking the logarithms of probabilities.

[7] Strictly speaking, an arc is not a subgraph of the tree, but there is a one-to-one mapping from an arc (w_i,l,w_j) to a subgraph with node set $V = \{w_i,w_j\}$ and arc set $A = \{(w_i,l,w_j)\}$. We will ignore this complication in the following discussion.

Works Cited

Attardi, Giuseppe. 2006. Experiments with a multilanguage non-projective dependency parser. Proceedings of the 10th conference on computational natural language learning (CoNLL), 166–70.

Booth, Taylor L., and Richard A. Thompson. 1973. Applying probability measures to abstract languages. IEEE Transactions on Computers C-22. 442–50.

Bresnan, Joan. 2000. Lexical-functional syntax. Oxford: Blackwell.

Buchholz, Sabine, and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. Proceedings of the 10th conference on computational natural language learning (CoNLL), 149–64.

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. Proceedings of the CoNLL shared task of EMNLP-CoNLL 2007, 957–61.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. Proceedings of the first meeting of the North American chapter of the association for computational linguistics (NAACL), 138–9.

Chelba, Ciprian, David Engle, Frederick Jelinek, Victor Jimenez, Sanjeev Khudanpur, Lidia Mangu, Harry Printz, Eric Ristad, Ronald Rosenfeld, Andreas Stolcke, and Dekai Wu. 1997. Structure and performance of a dependency language model. Proceedings of eurospeech, 2775–8.

Cheng, Yuchang, Masayuki Asahara, and Yuji Matsumoto. 2005. Machine learning-based dependency analyzer for Chinese. Proceedings of international conference on Chinese computing (ICCC), 66–73.

Chomsky, Noam. 1956. Three models for the description of language. IRE Transactions on Information Theory IT-2. 113–24.

Chu, Y. J., and T. H. Liu. 1965. On the shortest arborescence of a directed graph. Science Sinica 14. 1396–400.

Ciaramita, Massimiliano, and Giuseppe Attardi. 2007. Dependency parsing with second-order feature maps and annotated semantic information. Proceedings of the tenth international conference on parsing technologies (IWPT), 133–43.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. Proceedings of the 35th annual meeting of the association for computational linguistics (ACL) and the eighth conference of the European chapter of the association for computational linguistics (EACL), 16–23.

——. 1999. Head-driven statistical models for natural language parsing. PhD thesis, University of Pennsylvania.

——, and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures and the voted perceptron. Proceedings of the 40th annual meeting of the association for computational linguistics (ACL), 263–70.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. Journal of Machine Learning Research 7. 551–85.

Culotta, Aron, and Jeffery Sorensen. 2004. Dependency tree kernels for relation extraction. Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL), 423–9.

Ding, Yuan, and Martha Palmer. 2004. Synchronous dependency insertion grammars: a grammar formalism for syntax based statistical MT. Proceedings of the workshop on recent advances in dependency grammar, 90–7.

Duchier, Denys. 1999. Axiomatizing dependency parsing using set constraints. Proceedings of the sixth meeting on mathematics of language, 115–26.

——. 2003. Configuration of labeled trees under lexicalized constraints and principles. Research on Language and Computation 1. 307–36.

Earley, Jay. 1970. En efficient context-free parsing algorithm. Communications of the ACM 13. 94–102.

Edmonds, Jack. 1967. Optimum branchings. Journal of Research of the National Bureau of Standards 71B. 233–40.

Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: an exploration. Proceedings of the 16th international conference on computational linguistics (COLING), 340–5.

——. 2000. Bilexical grammars and their cubic-time parsing algorithms. Advances in probabilistic and other parsing technologies, ed. by Harry Bunt and Anton Nijholt, 29–62. Dordrecht: Kluwer.

Foth, Kilian, Michael Daum, and Wolfgang Menzel. 2004. A broad-coverage parser for German based on defeasible constraints. Proceedings of KONVENS 2004, 45–52.

Gaifman, Haim. 1965. Dependency systems and phrase-structure systems. Information and Control 8. 304–37.

Haghighi, Aria, Andrew Ng, and Chris D. Manning. 2005. Robust textual inference via graph matching. Proceedings of the human language technology conference and the conference on empirical methods in natural language processing (HLT/EMNLP), 387–94.

Harper, Mary P., and Randall A. Helzerman. 1995. Extensions to constraint dependency parsing for spoken language processing. Computer Speech and Language 9. 187–234.

Hays, David G. 1964. Dependency theory: a formalism and some observations. Language 40. 511–25.

Hudson, Richard A. 1984. Word grammar. Oxford: Blackwell.

——. 1990. English word grammar. Oxford: Blackwell.

——. 2007. Language networks: the new word grammar. Oxford: Oxford University Press.

Joshi, Aravind. 1985. How much context-sensitivity is necessary for characterizing structural descriptions – tree adjoining grammars. Natural language processing: psycholinguistic, computational and theoretical perspectives, ed. by David Dowty, Lauri Karttunen, and Arnold Zwicky, 206–50. Cambridge: Cambridge University Press.

——. 1997. Tree-adjoining grammars. Handbook of formal languages. Volume 3: beyond words, ed. by Grzegorz Rozenberg and Arto Salomaa, 69–123. Berlin: Springer.

Kaplan, Ron, and Joan, Bresnan. 1982. Lexical-functional grammar: a formal system for grammatical representation. The mental representation of grammatical relations, ed. by Joan Bresnan, 173–281. Cambridge, MA: MIT Press.

Kudo, Taku, and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. Proceedings of the sixth workshop on computational language learning (CoNLL), 63–9.

Lin, Dekang. 2003. Dependency-based evaluation with minipar. Treebanks: building and using parsed corpora, ed. by Anne Abeillé, 317–29. Dordrecht: Kluwer.

Maruyama, Hiroshi. 1990. Structural disambiguation with constraint propagation. Proceedings of the 28th meeting of the association for computational linguistics (ACL), 31–8.

McDonald, Ryan, and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), 122–31.

——, and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. Proceedings of the 11th conference of the European chapter of the association for computational linguistics (EACL), 81–8.

——, and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. Proceedings of the 10th international conference on parsing technologies (IWPT), 122–31.

——, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL), 91–8.

——, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. Proceedings of the human language technology conference and the conference on empirical methods in natural language technology conference and the conference on empirical methods in natural language processing (HLT/EMNLP), 523–30.

Mel'čuk, Igor. 1988. Dependency syntax: theory and practice. Albany, NY: State University of New York Press.
Menzel, Wolfgang, and Ingo Schröder. 1998. Decision procedures for dependency parsing using graded constraints. Proceedings of the workshop on processing of dependency-based grammars (ACL-COLING), 78–87.
Milicevic, J. 2006. A short guide to the meaning-text linguistic theory. Journal of Koralex 8. 187–233.
Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. Proceedings of the 8th international workshop on parsing technologies (IWPT), 149–60.
——. 2006. Constraints on non-projective dependency graphs. Proceedings of the 11th conference of the European chapter of the association for computational linguistics (EACL), 73–80.
——. 2007. Incremental non-projective dependency parsing. Proceedings of human language technologies: the annual conference of the North American chapter of the association for computational linguistics (NAACL HLT), 396–403.
——. 2008. Algorithms for deterministic incremental dependency parsing. Computational Linguistics 34. 513–53.
——, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. Proceedings of the eighth conference on computational natural language learning, 49–56.
——, ——, Jensand Nilsson, Gulsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. Proceedings of the 10th conference on computational natural language learning (CoNLL), 221–5.
——, ——, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. Proceedings of the CoNLL shared task of EMNLP-CoNLL 2007, 915–32.
——, and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. Proceedings of the 46th annual meeting of the association for computational linguistics (ACL).
Quirk, Chris, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: syntactically informed phrasal SMT. Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL), Ann Arbor, MI. 271–79.
Rosenblatt, Frank. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review 65(6). 386–408.
Sagae, Kenji, and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. Proceedings of the 9th international workshop on parsing technologies (IWPT), 125–32.
——. 2006a. A best-first probabilistic shift-reduce parser. Proceedings of the COLING/ACL 2006 main conference poster sessions, 691–8.
——. 2006b. Parser combination by reparsing. Proceedings of the human language technology conference of the NAACL, companion volume: short papers, 129–32.
Sgall, Petr, Eva Hajičová, and Jarmila Panevová. 1986. The meaning of the sentence in its pragmatic aspects. Dordrecht: Reidel.
Sleator, Daniel, and Davy Temperley. 1991. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University. Computer Science.
——. 1993. Parsing English with a link grammar. Proceedings of third international workshop on parsing technologies (IWPT), 277–92.
Snow, Rion, Dan Jurafsky, and Andrew Y. Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. Advances in Neural Information Processing Systems (NIPS).
Steedman, Mark. 2000. The syntactic process. Cambridge, MA: MIT Press.
Tesnière, Lucien. 1959. Éléments de syntaxe structurale. Paris: Editions Klincksieck.
Torres Martins, André Filipe, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking dependency parsers. Proceedings of the conference on empirical methods in natural language processing (EMNLP), 157–66.
Wang, Mengqiu, Noah A. Smith, and Teruko Mitamura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), 22–32.
Yamada, Hiroyasu, and Yuji, Matsumoto. 2003. Statistical dependency analysis with support vector machines. Proceedings of the 8th international workshop on parsing technologies (IWPT), 195–206.
Younger, Daniel H. 1967. Recognition and parsing of context-free languages in time n^3. Information and Control 10. 189–208.
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), 562–71.