Exact Inference for Generative Probabilistic Non-Projective Dependency Parsing

Shay B. Cohen
School of Computer Science
Carnegie Mellon University, USA
scohen@cs.cmu.edu

Carlos Gómez-Rodríguez
Departamento de Computación
Universidade da Coruña, Spain
cgomezr@udc.es

Giorgio Satta
Dept. of Information Engineering
University of Padua, Italy
satta@dei.unipd.it

Abstract

We describe a generative model for non-projective dependency parsing based on a simplified version of a transition system that has recently appeared in the literature. We then develop a dynamic programming parsing algorithm for our model, and derive an inside-outside algorithm that can be used for unsupervised learning of non-projective dependency trees.

1 Introduction

Dependency grammars have received considerable attention in the statistical parsing community in recent years. These grammatical formalisms offer a good balance between structural expressivity and processing efficiency. Most notably, when non-projectivity is supported, these formalisms can model crossing syntactic relations that are typical in languages with relatively free word order.

Recent work has reduced non-projective parsing to the identification of a maximum spanning tree in a graph (McDonald et al., 2005; Koo et al., 2007; McDonald and Satta, 2007; Smith and Smith, 2007). An alternative to this approach is to use transition-based parsing (Yamada and Matsumoto, 2003; Nivre and Nilsson, 2005; Attardi, 2006; Nivre, 2009; Gómez-Rodríguez and Nivre, 2010), where there is an incremental processing of a string with a model that scores transitions between parser states, conditioned on the parse history. This paper focuses on the latter approach.

The above work on transition-based parsing has focused on greedy algorithms set in a statistical framework (Nivre, 2008). More recently, dynamic programming has been successfully used for projective parsing (Huang and Sagae, 2010; Kuhlmann et al., 2011). Dynamic programming algorithms for parsing (also known as chart-based algorithms) allow polynomial space representations of all parse trees for a given input string, even in cases where the size of this set is exponential in the length of the string itself. In combination with appropriate semirings, these packed representations can be exploited to compute many values of interest for machine learning, such as best parses and feature expectations (Goodman, 1999; Li and Eisner, 2009).

In this paper we move one step forward with respect to Huang and Sagae (2010) and Kuhlmann et al. (2011) and present a polynomial dynamic programming algorithm for non-projective transition-based parsing. Our algorithm is coupled with a simplified version of the transition system from Attardi (2006), which has high coverage for the type of non-projective structures that appear in various treebanks. Instead of an additional transition operation which permits swapping of two elements in the stack (Titov et al., 2009; Nivre, 2009), Attardi's system allows reduction of elements at non-adjacent positions in the stack. We also present a *generative* probabilistic model for transition-based parsing. The implication for this, for example, is that one can now approach the problem of unsupervised learning of non-projective dependency structures within the transition-based framework.

Dynamic programming algorithms for non-projective parsing have been proposed by Kahane et al. (1998), Gómez-Rodríguez et al. (2009) and Kuhlmann and Satta (2009), but they all run in exponential time in the 'gap degree' of the parsed structures. To the best of our knowledge, this paper is the first to introduce a dynamic programming algorithm for inference with non-projective structures of unbounded gap degree.

The rest of this paper is organized as follows. In §2 and §3 we outline the transition-based model we use, together with a probabilistic generative interpretation. In §4 we give the tabular algorithm for parsing, and in §5 we discuss statistical inference using expectation maximization. We then discuss some other aspects of the work in §6 and conclude in §7.

2 Transition-based Dependency Parsing

In this section we briefly introduce the basic definitions for transition-based dependency parsing. For a more detailed presentation of this subject, we refer the reader to Nivre (2008). We then define a specific transition-based model for non-projective dependency parsing that we investigate in this paper.

2.1 General Transition Systems

Assume an input alphabet Σ with a special symbol $\$ \in \Sigma$, which we use as the root of our parse structures. Throughout this paper we denote the input string as $w = a_0 \cdots a_{n-1}$, $n \geq 1$, where $a_0 = \$$ and $a_i \in \Sigma \setminus \{\$\}$ for each i with $1 \leq i \leq n-1$.

A **dependency tree** for w is a directed tree $G_w = (V_w, A_w)$, where $V_w = \{0, \ldots, n-1\}$ is the set of nodes, and $A_w \subseteq V_w \times V_w$ is the set of arcs. The root of G_w is the node 0. The intended meaning is that each node in V_w encodes the position of a token in w. Furthermore, each arc in A_w encodes a dependency relation between two tokens. We write $i \rightarrow j$ to denote a directed arc $(i, j) \in A_w$, where node i is the head and node j is the dependent.

A **transition system** (for dependency parsing) is a tuple $S = (C, T, I, C_t)$, where C is a set of configurations, defined below, T is a finite set of **transitions**, which are partial functions $t\colon C \rightharpoonup C$, I is a total initialization function mapping each input string to a unique initial configuration, and $C_t \subseteq C$ is a set of terminal configurations.

A **configuration** is defined relative to some input string w, and is a triple (σ, β, A), where σ and β are disjoint lists called **stack** and **buffer**, respectively, and $A \subseteq V_w \times V_w$ is a set of arcs. Elements of σ and β are nodes from V_w and, in the case of the stack, a special symbol ¢ that we will use as initial stack symbol. If t is a transition and c_1, c_2 are configurations such that $t(c_1) = c_2$, we write $c_1 \vdash_t c_2$, or simply $c_1 \vdash c_2$ if t is understood from the context.

Given an input string w, a parser based on S incrementally processes w from left to right, starting in the initial configuration $I(w)$. At each step, the parser nondeterministically applies one transition, or else it stops if it has reached some terminal configuration. The dependency graph defined by the arc set associated with a terminal configuration is then returned as one possible analysis for w.

Formally, a **computation** of S is a sequence $\gamma = c_0, \ldots, c_m$, $m \geq 1$, of configurations such that, for every i with $1 \leq i \leq m$, $c_{i-1} \vdash_{t_i} c_i$ for some $t_i \in T$. In other words, each configuration in a computation is obtained as the value of the preceding configuration under some transition. A computation is called **complete** whenever $c_0 = I(w)$ for some input string w, and $c_m \in C_t$.

We can view a transition-based dependency parser as a device mapping strings into graphs (dependency trees). Without any restriction on transition functions in T, these functions might have an infinite domain, and could thus encode even non-recursively enumerable languages. However, in standard practice for natural language parsing, transitions are always specified by some finite mean. In particular, the definition of each transition depends on some finite window at the top of the stack and some finite window at the beginning of the buffer in each configuration. In this case, we can view a transition-based dependency parser as a notational variant of a push-down transducer (Hopcroft et al., 2000), whose computations output sequences that directly encode dependency trees. These transducers are nondeterministic, meaning that several transitions can be applied to some configurations. The transition systems we investigate in this paper follow these principles.

We close this subsection with some additional notation. We denote the stack with its topmost element to the right and the buffer with its first element to the left. We indicate concatenation in the stack and buffer by a vertical bar. For example, for $k \in V_w$, $\sigma|k$ denotes some stack with topmost element k and $k|\beta$ denotes some buffer with first element k. For $0 \leq i \leq n-1$, β_i denotes the buffer

$[i, i+1, \ldots, n-1]$; for $i \geq n$, β_i denotes $[]$ (the empty buffer).

2.2 A Non-projective Transition System

We now turn to give a description of our transition system for non-projective parsing. While a projective dependency tree satisfies the requirement that, for every arc in the tree, there is a directed path between its headword and each of the words between the two endpoints of the arc, a non-projective dependency tree may violate this condition. Even though some natural languages exhibit syntactic phenomena which require non-projective expressive power, most often such a resource is used in a limited way.

This idea is demonstrated by Attardi (2006), who proposes a transition system whose individual transitions can deal with non-projective dependencies only to a limited extent, depending on the distance in the stack of the nodes involved in the newly constructed dependency. The author defines this distance as the **degree** of the transition, with transitions of degree one being able to handle only projective dependencies. This formulation permits parsing a subset of the non-projective trees, where this subset depends on the degree of the transitions. The reported coverage in Attardi (2006) is already very high when the system is restricted to transitions of degree two or three. For instance, on training data for Czech containing 28,934 non-projective relations, 27,181 can be handled by degree two transitions, and 1,668 additional dependencies can be handled by degree three transitions. Table 1 gives additional statistics for treebanks from the CoNLL-X shared task (Buchholz and Marsi, 2006).

We now turn to describe our variant of the transition system of Attardi (2006), which is equivalent to the original system restricted to transitions of degree two. Our results are based on such a restriction. It is not difficult to extend our algorithms (§4) to higher degree transitions, but this comes at the expense of higher complexity. See §6 for more discussion on this issue.

Let $w = a_0 \cdots a_{n-1}$ be an input string over Σ defined as in §2.1, with $a_0 = \$$. Our transition system for non-projective dependency parsing is

$$S^{(\mathrm{np})} = (C, T^{(\mathrm{np})}, I^{(\mathrm{np})}, C_t^{(\mathrm{np})}),$$

Language	Deg. 2	Deg. 3	Deg. 4
Arabic	180	21	7
Bulgarian	961	41	10
Czech	27181	1668	85
Danish	876	136	53
Dutch	9072	2119	171
German	15827	2274	466
Japanese	1484	143	9
Portuguese	3104	424	37
Slovene	601	48	13
Spanish	66	7	0
Swedish	1566	226	79
Turkish	579	185	8

Table 1: The number of non-projective relations of various degrees for several treebanks (training sets), as reported by the parser of Attardi (2006). Deg. stands for 'degree.' The parser did not detect non-projective relations of degree higher than 4.

where C is the same set of configurations defined in §2.1. The initialization function $I^{(\mathrm{np})}$ maps each string w to the initial configuration $([¢], \beta_0, \emptyset)$. The set of terminal configurations $C_t^{(\mathrm{np})}$ contains all configurations of the form $([¢, 0], [], A)$, for any set of arcs A.

The set of transition functions is defined as

$$T^{(\mathrm{np})} = \{\mathsf{sh}_b \mid b \in \Sigma\} \cup \{\mathsf{la}_1, \mathsf{ra}_1, \mathsf{la}_2, \mathsf{ra}_2\},$$

where each transition is specified below. We let variables i, j, k, l range over V_w, and variable σ is a list of stack elements from $V_w \cup \{¢\}$:

$$\begin{array}{rrcl}
\mathsf{sh}_b: & (\sigma, k|\beta, A) & \vdash & (\sigma|k, \beta, A) \text{ if } a_k = b; \\
\mathsf{la}_1: & (\sigma|i|j, \beta, A) & \vdash & (\sigma|j, \beta, A \cup \{j \to i\}); \\
\mathsf{ra}_1: & (\sigma|i|j, \beta, A) & \vdash & (\sigma|i, \beta, A \cup \{i \to j\}); \\
\mathsf{la}_2: & (\sigma|i|j|k, \beta, A) & \vdash & (\sigma|j|k, \beta, A \cup \{k \to i\}); \\
\mathsf{ra}_2: & (\sigma|i|j|k, \beta, A) & \vdash & (\sigma|i|j, \beta, A \cup \{i \to k\}).
\end{array}$$

Each of the above transitions is undefined on configurations that do not match the forms specified above. As an example, transition la_2 is not defined for a configuration (σ, β, A) with $|\sigma| \leq 2$, and transition sh_b is not defined for a configuration $(\sigma, k|\beta, A)$ with $b \neq a_k$, or for a configuration $(\sigma, [], A)$.

Transition sh_b removes the first node from the buffer, in case this node represents symbol $b \in \Sigma$, and pushes it into the stack. These transitions are called **shift** transitions. The remaining four transitions are called **reduce** transitions, i.e., transitions that consume nodes from the stack. Notice that in the transition system at hand all the reduce transitions decrease the size of the stack by one element. Transition la_1 creates a new arc with the topmost node on the stack as the head and the second-topmost node as the dependent, and removes the latter from the stack. Transition ra_1 is symmetric with respect to la_1. Transitions la_1 and ra_1 have degree one, as already explained. When restricted to these three transitions, the system is equivalent to the so-called stack-based arc-standard model of Nivre (2004). Transition la_2 and transition ra_2 are very similar to la_1 and ra_1, respectively, but with the difference that they create a new arc between the topmost node in the stack and a node which is two positions below the topmost node. Hence, these transitions have degree two, and are the key components in parsing of non-projective dependencies.

We turn next to describe the equivalence between our system and the system in Attardi (2006). The transition-based parser presented by Attardi pushes back into the buffer elements that are in the top position of the stack. However, a careful analysis shows that only the first position in the buffer can be affected by this operation, in the sense that elements that are pushed back from the stack are never found in buffer positions other than the first. This means that we can consider the first element of the buffer as an additional stack element, always sitting on the top of the top-most stack symbol.

More formally, we can define a function $m_c : C \to C$ that maps configurations in the original algorithm to those in our variant as follows:

$$m_c((\sigma, k|\beta, A)) = (\sigma|k, \beta, A)$$

By applying this mapping to the source and target configuration of each transition in the original system, it is easy to check that $c_1 \vdash c_2$ in that parser if and only if $m_c(c_1) \vdash m_c(c_2)$ in our variant. We extend this and define an isomorphism between computations in both systems, such that a computation $c_0, \dots, c_m$ in the original parser is mapped to a computation $m_c(c_0), \dots, m_c(c_m)$ in the variant, with both generating the same dependency graph A. This

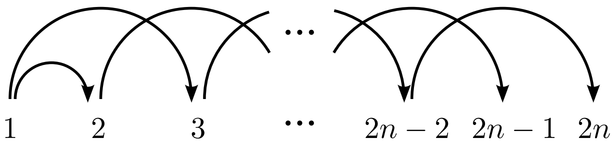

Figure 1: A dependency structure of arbitrary gap degree that can be parsed with Attardi's parser.

proves that our notational variant is in fact equivalent to Attardi's parser.

A relevant property of the set of dependency structures that can be processed by Attardi's parser, even when restricted to transitions of degree two, is that the number of discontinuities present in each of their subtrees, defined as the **gap degree** by Bodirsky et al. (2005), is not bounded. For example, the dependency graph in Figure 1 has gap degree $n - 1$, and it can be parsed by the algorithm for any arbitrary $n \geq 1$ by applying $2n$ sh_b transitions to push all the nodes into the stack, followed by $(2n - 2)$ ra_2 transitions to create the crossing arcs, and finally one ra_1 transition to create the dependency $1 \to 2$.

As mentioned in §1, the computational complexity of the dynamic programming algorithm that will be described in later sections does not depend on the gap degree, contrary to the non-projective dependency chart parsers presented by Gómez-Rodríguez et al. (2009) and by Kuhlmann and Satta (2009), whose running time is exponential in the maximum gap degree allowed by the grammar.

3 A Generative Probabilistic Model

In this section we introduce a generative probabilistic model based on the transition system of §2.2. In formal language theory, there is a standard way of giving a probabilistic interpretation to a non-deterministic parser whose computations are based on sequences of elementary operations such as transitions. The idea is to define conditional probability distributions over instances of the transition functions, and to 'combine' these probabilities to assign probabilities to computations and strings.

One difficulty we have to face with when dealing with transition systems is that the notion of computation, defined in §2.1, depends on the input string, because of the buffer component appearing in each configuration. This is a pitfall to generative modeling, where we are interested in a system whose computations lead to the *generation* of any string. To overcome this problem, we observe that each computation, defined as a sequence of stacks and buffers (the configurations) can equivalently be expressed as a sequence of stacks and transitions.

More precisely, consider a computation $\gamma = c_0, \dots, c_m$, $m \geq 1$. Let σ_i, be the stack associated with c_i, for each i with $0 \leq i \leq m$. Let also C_σ be the set of all stacks associated with configurations in C. We can make explicit the transitions that have been used in the computation by rewriting γ in the form $\sigma_0 \vdash_{t_1} \sigma_1 \cdots \sigma_{m-1} \vdash_{t_m} \sigma_m$. In this way, γ generates a string that is composed by all symbols that are pushed into the stack by transitions sh_b, in the left to right order.

We can now associate a probability to (our representation of) sequence γ by setting

$$p(\gamma) = \prod_{i=1}^{m} p(t_i \mid \sigma_{i-1}). \qquad (1)$$

To assign probabilities to complete computations we should further multiply $p(\gamma)$ by factors $p_s(\sigma_0)$ and $p_e(\sigma_m)$, where p_s and p_e are start and end probability distributions, respectively, both defined over C_σ. Note however that, as defined in §2.2, all initial configurations are associated with stack $[\cent]$ and all final configurations are associated with stack $[\cent, 0]$, thus p_s and p_e are deterministic. Note that the Markov chain represented in Eq. 1 is *homogeneous*, i.e., the probabilities of the transition operations do not depend on the time step.

As a second step we observe that, according to the definition of transition system, each $t \in T$ has an infinite domain. A commonly adopted solution is to introduce a special function, called **history** function and denoted by H, defined over the set C_σ and taking values over some finite set. For each $t \in T$ and $\sigma, \sigma' \in C_\sigma$, we then impose the condition

$$p(t \mid \sigma) = p(t \mid \sigma')$$

whenever $H(\sigma) = H(\sigma')$. Since H is finitely valued, and since T is a finite set, the above condition guarantees that there will only be a finite number of parameters $p(t \mid \sigma)$ in our model.

So far we have presented a general discussion of how to turn a transition-based parser into a generative probabilistic model, and have avoided further specification of the history function. We now turn our attention to the non-projective transition system of §2.2. To actually transform that system into a parametrized probabilistic model, and to develop an associated efficient inference procedure as well, we need to balance between the amount of information we put into the history function and the computational complexity which is required for inference.

We start the discussion with a naïve model using a history function defined by a fixed size window over the topmost portion of the stack. More precisely, each transition is conditioned on the lexical form of the three symbols at the top of the stack σ, indicated as $b_3, b_2, b_1 \in \Sigma$ below, with b_1 referring to the topmost symbol. The parameters of the model are defined as follows.

$$\begin{aligned} p(\mathsf{sh}_b \mid b_3, b_2, b_1) &= \theta^{\mathsf{sh}_b}_{b_3,b_2,b_1} \ , \quad \forall b \in \Sigma \ , \\ p(\mathsf{la}_1 \mid b_3, b_2, b_1) &= \theta^{\mathsf{la}_1}_{b_3,b_2,b_1} \ , \\ p(\mathsf{ra}_1 \mid b_3, b_2, b_1) &= \theta^{\mathsf{ra}_1}_{b_3,b_2,b_1} \ , \\ p(\mathsf{la}_2 \mid b_3, b_2, b_1) &= \theta^{\mathsf{la}_2}_{b_3,b_2,b_1} \ , \\ p(\mathsf{ra}_2 \mid b_3, b_2, b_1) &= \theta^{\mathsf{ra}_2}_{b_3,b_2,b_1} \ . \end{aligned}$$

The parameters above are subject to the following normalization conditions, for every choice of $b_3, b_2, b_1 \in \Sigma$:

$$\theta^{\mathsf{la}_1}_{b_3,b_2,b_1} + \theta^{\mathsf{ra}_1}_{b_3,b_2,b_1} + \theta^{\mathsf{la}_2}_{b_3,b_2,b_1} + \\ \theta^{\mathsf{ra}_2}_{b_3,b_2,b_1} + \sum_{b \in \Sigma} \theta^{\mathsf{sh}_b}_{b_3,b_2,b_1} = 1 \ .$$

This naïve model presents two practical problems. The first problem relates to the efficiency of an inference algorithm, which has a quite high computational complexity, as it will be discussed in §5. A second problem arises in the probabilistic setting. Using this model would require estimating many parameters which are based on trigrams. This leads to higher sample complexity to avoid sparse counts: we would need more samples to accurately estimate the model.

We therefore consider a more elaborated model, which tackles both of the above problems. Again, let $b_3, b_2, b_1 \in \Sigma$ indicate the lexical form of the three symbols at the top of the stack. We define the distributions $p(t \mid \sigma)$ as follows:

$$p(\mathsf{sh}_b \mid b_1) = \theta_{b_1}^{\mathsf{sh}_b}, \ \ \forall b \in \Sigma \ ,$$
$$p(\mathsf{la}_1 \mid b_2, b_1) = \theta_{b_1}^{\mathsf{rd}} \cdot \theta_{b_2,b_1}^{\mathsf{la}_1} \ ,$$
$$p(\mathsf{ra}_1 \mid b_2, b_1) = \theta_{b_1}^{\mathsf{rd}} \cdot \theta_{b_2,b_1}^{\mathsf{ra}_1} \ ,$$
$$p(\mathsf{la}_2 \mid b_3, b_2, b_1) = \theta_{b_1}^{\mathsf{rd}} \cdot \theta_{b_2,b_1}^{\mathsf{rd}_2} \cdot \theta_{b_3,b_2,b_1}^{\mathsf{la}_2} \ ,$$
$$p(\mathsf{ra}_2 \mid b_3, b_2, b_1) = \theta_{b_1}^{\mathsf{rd}} \cdot \theta_{b_2,b_1}^{\mathsf{rd}_2} \cdot \theta_{b_3,b_2,b_1}^{\mathsf{ra}_2} \ .$$

The parameters above are subject to the following normalization conditions, for every $b_3, b_2, b_1 \in \Sigma$:

$$\sum_{b \in \Sigma} \theta_{b_1}^{\mathsf{sh}_b} + \theta_{b_1}^{\mathsf{rd}} = 1 \ , \qquad (2)$$

$$\theta_{b_2,b_1}^{\mathsf{la}_1} + \theta_{b_2,b_1}^{\mathsf{ra}_1} + \theta_{b_2,b_1}^{\mathsf{rd}_2} = 1 \ , \qquad (3)$$

$$\theta_{b_3,b_2,b_1}^{\mathsf{la}_2} + \theta_{b_3,b_2,b_1}^{\mathsf{ra}_2} = 1 \ . \qquad (4)$$

Intuitively, parameter θ_b^{rd} denotes the probability that we perform a reduce transition instead of a shift transition, given that we have seen lexical form b at the top of the stack. Similarly, parameter $\theta_{b_2,b_1}^{\mathsf{rd}_2}$ denotes the probability that we perform a reduce transition of degree 2 (see §2.2) instead of a reduce transition of degree 1, given that we have seen lexical forms b_1 and b_2 at the top of the stack.

We observe that the above model has a number of parameters $|\Sigma| + 4 \cdot |\Sigma|^2 + 2 \cdot |\Sigma|^3$ (not all independent). This should be contrasted with the naïve model, that has a number of parameters $4 \cdot |\Sigma|^3 + |\Sigma|^4$.

4 Tabular parsing

We present here a dynamic programming algorithm for simulating the computations of the system from §2–3. Given an input string w, our algorithm produces a compact representation of the set $\Gamma(w)$, defined as the set of all possible computations of the model when processing w. In combination with the appropriate semirings, this method can provide for instance the highest probability computation in $\Gamma(w)$, or else the probability of w, defined as the sum of all probabilities of computations in $\Gamma(w)$.

We follow a standard approach in the literature on dynamic programming simulation of stack-based automata (Lang, 1974; Tomita, 1986; Billot and Lang, 1989). More recently, this approach has also been applied by Huang and Sagae (2010) and by Kuhlmann et al. (2011) to the simulation of projective transition-based parsers. The basic idea in this approach is to decompose computations of the parser into smaller parts, group them into equivalence classes and recombine to obtain larger parts of computations.

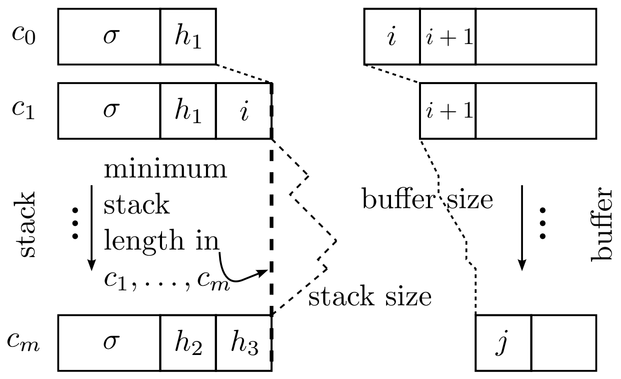

Figure 2: Schematic representation of the computations γ associated with item $[h_1, i, h_2 h_3, j]$.

Let $w = a_0 \cdots a_{n-1}$, V_w and $S^{(np)}$ be defined as in §2. We use a structure called **item**, defined as

$$[h_1, i, h_2 h_3, j],$$

where $0 \leq i < j \leq n$ and $h_1, h_2, h_3 \in V_w$ must satisfy $h_1 < i$ and $i \leq h_2 < h_3 < j$. The intended interpretation of an item can be stated as follows; see also Figure 2.

- There exists a computation γ of $S^{(\mathrm{np})}$ on w having the form $c_0, \ldots, c_m$, $m \geq 1$, with $c_0 = (\sigma | h_1, \beta_i, A)$ and $c_m = (\sigma | h_2 | h_3, \beta_j, A')$ for some stack σ and some arc sets A and A';
- For each i with $1 \leq i < m$, the stack σ_i associated with configuration c_i has the list σ at the bottom and satisfies $|\sigma_i| \geq |\sigma| + 2$.

Some comments on the above conditions are in order here. Let $t_1, \cdots, t_m$ be the sequence of transitions in $T^{(np)}$ associated with computation γ. Then we have $t_1 = \mathsf{sh}_{a_i}$, since $|\sigma_1| \geq |\sigma| + 2$. Thus we conclude that $|\sigma_1| = |\sigma| + 2$.

The most important consequence of the definition of item is that each transition t_i with $2 \leq i \leq m$ does not depend on the content of the σ portion of the stack σ_i. To see this, consider transition $c_{i-1} \vdash_{t_i} c_i$. If $t_i = \mathsf{sh}_{a_i}$, the content of σ is irrelevant at this step, since in our model sh_{a_i} is conditioned only on the topmost stack symbol of σ_{i-1}, and we have $|\sigma_{i-1}| \geq |\sigma| + 2$.

Consider now the case of $t_i = \mathsf{la}_2$. From $|\sigma_i| \geq |\sigma| + 2$ we have that $|\sigma_{i-1}| \geq |\sigma| + 3$. Again, the content of σ is irrelevant at this step, since in our model la_2 is conditioned only on the three topmost stack symbols of σ_{i-1}. A similar argument applies to the cases of $t_i \in \{\mathsf{ra}_2, \mathsf{la}_1, \mathsf{ra}_1\}$.

From the above, we conclude that if we apply the transitions $t_1, \ldots, t_m$ to stacks of the form $\sigma|h_1$, the resulting computations have all identical probabilities, independently of the choice of σ.

Each computation satisfying the two conditions above will be called an **I-computation** associated with item $[h_1, i, h_2h_3, j]$. Notice that an I-computation has the overall effect of replacing node h_1 sitting above a stack σ with nodes h_2 and h_3. This is the key property in the development of our algorithm below.

We specify our dynamic programming algorithm as a deduction system (Shieber et al., 1995). The deduction system starts with axiom $[¢, 0, ¢0, 1]$, corresponding to an initial stack $[¢]$ and to the shift of $a_0 = \$$ from the buffer into the stack. The set $\mathit{\Gamma}(w)$ is non-empty if and only if item $[¢, 0, ¢0, n]$ can be derived using the inference rules specified below. Each inference rule is annotated with the type of transition it simulates, along with the arc constructed by the transition itself, if any.

$$\frac{[h_1, i, h_2h_3, j]}{[h_3, j, h_3j, j+1]} \quad (\mathsf{sh}_{a_j})$$

$$\frac{[h_1, i, h_2h_3, k] \quad [h_3, k, h_4h_5, j]}{[h_1, i, h_2h_5, j]} \quad (\mathsf{la}_1; h_5 \rightarrow h_4)$$

$$\frac{[h_1, i, h_2h_3, k] \quad [h_3, k, h_4h_5, j]}{[h_1, i, h_2h_4, j]} \quad (\mathsf{ra}_1; h_4 \rightarrow h_5)$$

$$\frac{[h_1, i, h_2h_3, k] \quad [h_3, k, h_4h_5, j]}{[h_1, i, h_4h_5, j]} \quad (\mathsf{la}_2; h_5 \rightarrow h_2)$$

$$\frac{[h_1, i, h_2h_3, k] \quad [h_3, k, h_4h_5, j]}{[h_1, i, h_2h_4, j]} \quad (\mathsf{ra}_2; h_2 \rightarrow h_5)$$

The above deduction system infers items in a bottom-up fashion. This means that longer computations over substrings of w are built by combining shorter ones. In particular, the inference rule sh_{a_j} asserts the existence of I-computations consisting of a single sh_{a_j} transition. Such computations are represented by the consequent item $[h_3, j, h_3j, j+1]$, indicating that the index of the shifted word a_j is added to the stack by pushing it on top of h_3.

The remaining four rules implement the reduce transitions of the model. We have already observed in §2.2 that all available reduce transitions shorten the size of the stack by one unit. This allows us to combine pairs of I-computations with a reduce transition, resulting in a computation that is again an I-computation. More precisely, if we concatenate an I-computation asserted by an item $[h_1, i, h_2h_3, k]$ with an I-computation asserted by an item $[h_3, k, h_4h_5, j]$, we obtain a computation that has the overall effect of increasing the size of the stack by 2, replacing the topmost stack element h_1 with stack elements h_2, h_4 and h_5. If we now apply any of the reduce transitions from the inventory of the model, we will remove one of these three nodes from the stack, and the overall result will be again an I-computation, which can then be asserted by a certain item. For example, if we apply the reduce transition la_1, the consequent item is $[h_1, i, h_2h_5, j]$, since an la_1 transition removes the second topmost element from the stack (h_4). The other reduce transitions remove a different element, and thus their rules produce different consequent items.

The above argument shows the soundness of the deduction system, i.e., an item $I = [h_1, i, h_2h_3, j]$ is only generated if there exists an I-computation $\gamma = c_0, \ldots, c_m$ with $c_0 = (\sigma|h_1, \beta_i, A)$ and $c_m = (\sigma|h_2|h_3, \beta_j, A')$. To prove completeness, we must show the converse result, i.e., that the existence of an I-computation γ implies that item I is inferred. We first do this under the assumption that the inference rule for the shift transitions do not have an antecedent, i.e., items $[h_1, j, h_1j, j+1]$ are considered as axioms. We proceed by using strong induction on the length m of the computation γ.

For $m = 1$, γ consists of a single transition sh_{a_j}, and the corresponding item $I = [h_1, j, h_1j, j+1]$ is constructed as an axiom. For $m > 1$, let γ be as specified above. The transition that produced

c_m must have been a reduce transition, otherwise γ would not be an I-computation. Let c_k be the rightmost configuration in $c_0, \ldots, c_{m-1}$ whose stack size is $|\sigma| + 2$. Then it can be shown that the computations $\gamma_1 = c_0, \ldots, c_k$ and $\gamma_2 = c_k, \ldots, c_{m-1}$ are again I-computations. Since γ_1 and γ_2 have strictly fewer transitions than γ, by the induction hypothesis, the system constructs items $[h_1, i, h_2h_3, k]$ and $[h_3, k, h_4h_5, j]$, where h_2 and h_3 are the stack elements at the top of c_k. Applying to these items the inference rule corresponding to the reduce transition at hand, we can construct item I.

When the inference rule for the shift transition has an antecedent $[h_1, i, h_2h_3, j]$, as indicated above, we have the overall effect that I-computations consisting of a single transition shifting a_j on the top of h_3 are simulated only in case there exists a computation starting with configuration $([\text{¢}], \beta_0)$ and reaching a configuration of the form $(\sigma|h_2|h_3, \beta_j)$. This acts as a filter on the search space of the algorithm, but does not invalidate the completeness property. However, in this case the proof is considerably more involved, and we do not report it here.

An important property of the deduction system above, which will be used in the next section, is that the system is unambiguous, that is, each **I-computation** is constructed by the system in a unique way. This can be seen by observing that, in the sketch of the completeness proof reported above, there always is an unique choice of c_k that decomposes I-computation γ into I-computations γ_1 and γ_2. In fact, if we choose a configuration $c_{k'}$ other than c_k with stack size $|\sigma| + 2$, the computation $\gamma_2' = c_{k'}, \ldots, c_{m-1}$ will contain c_k as an intermediate configuration, which violates the definition of I-computation because of an intervening stack having size not larger than the size of the stack associated with the initial configuration.

As a final remark, we observe that we can keep track of all inference rules that have been applied in the computation of each item by the above algorithm, by encoding each application of a rule as a reference to the pair of items that were taken as antecedent in the inference. In this way, we obtain a parse forest structure that can be viewed as a hypergraph or as a non-recursive context-free grammar, similar to the case of parsing based on context-free grammars. See for instance Klein and Manning (2001) or Nederhof (2003). Such a parse forest encodes all valid computations in $\Gamma(w)$, as desired.

The algorithm runs in $\mathcal{O}(n^8)$ time. Using methods similar to those specified in Eisner and Satta (1999), we can reduce the running time to $\mathcal{O}(n^7)$. However, we do not further pursue this idea here, and proceed with the discussion of exact inference, found in the next section.

5 Inference

We turn next to specify exact inference with our model, for computing feature expectations. Such inference enables, for example, the derivation of an expectation-maximization algorithm for unsupervised parsing.

Here, a feature is a function over computations, providing the count of a pattern related to a parameter. We denote by $f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma)$, for instance, the number of occurrences of transition la_2 within γ with topmost stack symbols having word forms $b_3, b_2, b_1 \in \Sigma$, with b_1 associated with the topmost stack symbol.

Feature expectations are computed by using an inside-outside algorithm for the items in the tabular algorithm. More specifically, given a string w, we associate each item $[h_1, i, h_2h_3, j]$ defined as in §4 with two quantities:

$$I([h_1, i, h_2h_3, j]) \;=\; \sum_{\gamma=([h_1],\beta_i),\ldots,([h_2,h_3],\beta_j)} p(\gamma) \; ; \qquad (5)$$

$$O([h_1, i, h_2h_3, j]) \;=\; \sum_{\substack{\sigma,\gamma=([\text{¢}],\beta_0),\ldots,(\sigma|h_1,\beta_i) \\ \gamma'=(\sigma|h_2|h_3,\beta_j),\ldots,([\text{¢},0],\beta_n)}} p(\gamma) \cdot p(\gamma') \; . \qquad (6)$$

$I([h_1, i, h_2h_3, j])$ and $O([h_1, i, h_2h_3, j])$ are called the **inside** and the **outside** probabilities, respectively, of item $[h_1, i, h_2h_3, j]$. The tabular algorithm of §4 can be used to compute the inside probabilities. Using the gradient transformation (Eisner et al., 2005), a technique for deriving outside probabilities from a set of inference rules, we can also compute $O([h_1, i, h_2h_3, j])$. The use of the gradient transformation is valid in our case because the tabular algorithm is unambiguous (see §4).

Using the inside and outside probabilities, we can now efficiently compute feature expectations for our

$$
\begin{aligned}
E_{p(\gamma|w)}[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma)] &= \sum_{\gamma\in\Gamma(w)} p(\gamma \mid w)\cdot f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma) = \frac{1}{p(w)}\cdot \sum_{\gamma\in\Gamma(w)} p(\gamma)\cdot f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma) \\
&= \frac{1}{p(w)}\cdot \sum_{\substack{\sigma,i,k,j,\\ h_1,h_2,h_3,h_4,h_5,\\ \text{s.t. } a_{h_2}=b_3,\\ a_{h_4}=b_2,\, a_{h_5}=b_1}} \quad \sum_{\substack{\gamma_0=([\text{¢}],\beta_0),\ldots,(\sigma|h_1,\beta_i),\\ \gamma_1=(\sigma|h_1,\beta_i),\ldots,(\sigma|h_2|h_3,\beta_k),\\ \gamma_2=(\sigma|h_2|h_3,\beta_k),\ldots,(\sigma|h_2|h_4|h_5,\beta_j),\\ \gamma_3=(\sigma|h_2|h_5,\beta_j),\ldots,([\text{¢},0],\beta_n)}} p(\gamma_0)\cdot p(\gamma_1)\cdot p(\gamma_2)\cdot p(\mathsf{la}_2 \mid b_3,b_2,b_1)\cdot p(\gamma_3) \\
&= \frac{\theta^{\mathsf{rd}}_{b_1}\cdot\theta^{\mathsf{rd}_2}_{b_2,b_1}\cdot\theta^{\mathsf{la}_2}_{b_3,b_2,b_1}}{p(w)}\cdot \sum_{\substack{\sigma,i,j,\\ h_1,h_2,h_5,\ \text{s.t.}\\ a_{h_2}=b_3,\, a_{h_5}=b_1}} \quad \sum_{\substack{\gamma_0=([\text{¢}],\beta_0),\ldots,(\sigma|h_1,\beta_i),\\ \gamma_3=(\sigma|h_2|h_5,\beta_j),\ldots,([\text{¢},0],\beta_n)}} p(\gamma_0)\cdot p(\gamma_3)\cdot \\
&\qquad \cdot \sum_{\substack{k,h_3,h_4,\\ \text{s.t. } a_{h_4}=b_2}} \quad \sum_{\gamma_1=(\sigma|h_1,\beta_i),\ldots,(\sigma|h_2|h_3,\beta_k)} p(\gamma_1)\cdot \sum_{\gamma_2=(\sigma|h_2|h_3,\beta_k),\ldots,(\sigma|h_2|h_4|h_5,\beta_j)} p(\gamma_2)
\end{aligned}
$$

Figure 3: Decomposition of the feature expectation $E_{p(\gamma|w)}[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma)]$ into a finite summation. Quantity $p(w)$ above is the sum over all probabilities of computations in $\Gamma(w)$.

model. Figure 3 shows how to express the expectation of feature $f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma)$ by means of a finite summation. Using Eq. 5 and 6 and the relation $p(w) = I([\text{¢}, 0, \text{¢}0, n])$ we can then write:

$$
\begin{aligned}
E_{p(\gamma|w)}[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma)] &= \frac{\theta^{\mathsf{rd}}_{b_1}\cdot\theta^{\mathsf{rd}_2}_{b_2,b_1}\cdot\theta^{\mathsf{la}_2}_{b_3,b_2,b_1}}{I([\text{¢}, 0, \text{¢}0, n])}\cdot \\
&\cdot \sum_{\substack{i,j,h_1,h_4,h_5,\\ \text{s.t. } a_{h_4}=b_2,\, a_{h_5}=b_1}} O([h_1, i, h_4h_5, j])\cdot \\
&\cdot \sum_{\substack{k,h_2,h_3,\\ \text{s.t. } a_{h_2}=b_3}} I([h_1, i, h_2h_3, k])\cdot I([h_3, k, h_4h_5, j])\ .
\end{aligned}
$$

Very similar expressions can be derived for the expectations for features $f^{\mathsf{ra}_2}_{b_3,b_2,b_1}(\gamma)$, $f^{\mathsf{la}_1}_{b_2,b_1}(\gamma)$, and $f^{\mathsf{ra}_1}_{b_2,b_1}(\gamma)$. As for feature $f^{\mathsf{sh}_b}_{b_1}(\gamma)$, $b\in\Sigma$, the above approach leads to

$$
\begin{aligned}
&E_{p(\gamma|w)}[f^{\mathsf{sh}_b}_{b_1}(\gamma)] = \\
&\quad = \frac{\theta^{\mathsf{sh}_b}_{b_1}}{I([\text{¢}, 0, \text{¢}0, n])}\cdot \sum_{\substack{\sigma,i,h,\ \text{s.t.}\\ a_h=b_1,\, a_i=b}} O([h, i, hi, i+1])\ .
\end{aligned}
$$

As mentioned above, these expectations can be used, for example, to derive an EM algorithm for our model. The EM algorithm in our case is not completely straightforward because of the way we parametrize the model. We give now the re-estimation steps for such an EM algorithm. We assume that all expectations below are taken with respect to a set of parameters $\boldsymbol{\theta}$ from iteration $s-1$ of the algorithm, and we are required to update these $\boldsymbol{\theta}$. To simplify notation, let us assume that there is only one string w in the training corpus. For each $b_1\in\Sigma$, we define:

$$
\begin{aligned}
Z_{b_1} &= \sum_{b_2\in\Sigma} E_{p(\gamma|w)}\left[f^{\mathsf{la}_1}_{b_2,b_1}(\gamma) + f^{\mathsf{ra}_1}_{b_2,b_1}(\gamma)\right] \\
&\quad + \sum_{b_3,b_2\in\Sigma} E_{p(\gamma|w)}\left[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma) + f^{\mathsf{ra}_2}_{b_3,b_2,b_1}(\gamma)\right]\ ; \\
Z_{b_2,b_1} &= \sum_{b_3\in\Sigma} E_{p(\gamma|w)}\left[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma) + f^{\mathsf{ra}_2}_{b_3,b_2,b_1}(\gamma)\right]\ .
\end{aligned}
$$

We then have, for every $b\in\Sigma$:

$$
\theta^{\mathsf{sh}_b}_{b_1}(s) \leftarrow \frac{E_{p(\gamma|w)}[f^{\mathsf{sh}_b}_{b_1}(\gamma)]}{Z_{b_1} + \sum_{b'\in\Sigma} E_{p(\gamma|w)}[f^{\mathsf{sh}_{b'}}_{b_1}(\gamma)]}\ .
$$

Furthermore, we have:

$$\theta^{\mathsf{la}_1}_{b_2,b_1}(s) \leftarrow \frac{E_{p(\gamma|w)}[f^{\mathsf{la}_1}_{b_2,b_1}(\gamma)]}{Z_{b_2,b_1} + E_{p(\gamma|w)}\left[f^{\mathsf{la}_1}_{b_2,b_1}(\gamma) + f^{\mathsf{ra}_1}_{b_2,b_1}(\gamma)\right]},$$

and:

$$\theta^{\mathsf{la}_2}_{b_3,b_2,b_1}(s) \leftarrow \frac{E_{p(\gamma|w)}[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma)]}{E_{p(\gamma|w)}\left[f^{\mathsf{la}_2}_{b_3,b_2,b_1}(\gamma) + f^{\mathsf{ra}_2}_{b_3,b_2,b_1}(\gamma)\right]}.$$

The rest of the parameter updates can easily be derived using the above updates because of the sum-to-1 constraints in Eq. 2–4.

6 Discussion

We note that our model inherits spurious ambiguity from Attardi's model. More specifically, we can have different derivations, corresponding to different system computations, that result in identical dependency graphs and strings. While running our tabular algorithm with the Viterbi semiring efficiently computes the highest probability computation in $\Gamma(w)$, spurious ambiguity means that finding the highest probability dependency tree is NP-hard. This latter result can be shown using proof techniques similar to those developed by Sima'an (1996). We leave it for future work how to eliminate spurious ambiguity from the model.

While in the previous sections we have described a tabular method for the transition system of Attardi (2006) restricted to transitions of degree up to two, it is possible to generalize the model to include higher-degree transitions. In the general formulation of Attardi parser, transitions of degree d create links involving nodes located d positions beneath the topmost position in the stack:

$$\begin{aligned}
\mathsf{la}_d: &\quad (\sigma|i_1|i_2|\ldots|i_{d+1}, \beta, A) \vdash \\
&\quad (\sigma|i_2|\ldots|i_{d+1}, \beta, A \cup \{i_{d+1} \to i_1\});\\
\mathsf{ra}_d: &\quad (\sigma|i_1|i_2|\ldots|i_{d+1}, \beta, A) \vdash \\
&\quad (\sigma|i_1|i_2|\ldots|i_d, \beta, A \cup \{i_1 \to i_{d+1}\}).
\end{aligned}$$

To define a transition system that supports transitions up to degree D, we use a set of items of the form $[s_1 \ldots s_{D-1}, i, e_1 \ldots e_D, j]$, corresponding (in the sense of §4) to computations of the form $c_0, \ldots, c_m$, $m \geq 1$, with $c_0 = (\sigma|s_1|\ldots|s_{D-1}, \beta_i, A)$ and $c_m = (\sigma|e_1|\ldots|e_D, \beta_j, A')$. The deduction steps corresponding to reduce transitions in this general system have the general form

$$\frac{\begin{array}{c}[s_1 \ldots s_{D-1}, i, e_1 m_1 \ldots m_{D-1}, j]\\ [m_1 \ldots m_{D-1}, j, e_2 \ldots e_{D+1}, w]\end{array}}{[s_1 \ldots s_{D-1}, i, e_1 \ldots e_{c-1} e_{c+1} \ldots e_{D+1}, w]} \;(e_p \to e_c)$$

where the values of p and c differ for each transition: to obtain the inference rule corresponding to a la_d transition, we make $p = D + 1$ and $c = D + 1 - d$; and to obtain the rule for a ra_d transition, we make $p = D + 1 - d$ and $c = D + 1$. Note that the parser runs in time $O(n^{3D+2})$, where D stands for the maximum transition degree, so each unit increase in the transition degree adds a cubic factor to the parser's polynomial time complexity. This is in contrast to a previous tabular formulation of the Attardi parser by Gómez-Rodríguez et al. (2011), which ran in exponential time.

The model for the transition system we give in this paper is generative. It is not hard to naturally extend this model to the discriminative setting. In this case, we would condition the model on the input string to get a conditional distribution over derivations. It is perhaps more natural in this setting to use arbitrary weights for the parameter values, since the computation of a normalization constant (the probability of a string) is required in any case. Arbitrary weights in the generative setting could be more problematic, because it would require computing a normalization constant corresponding to a sum over all *strings* and derivations.

7 Conclusion

We presented in this paper a generative probabilistic model for non-projective parsing, together with the description of an efficient tabular algorithm for parsing and doing statistical inference with the model.

Acknowledgments

The authors thank Marco Kuhlmann for helpful comments on an early draft of the paper. The authors also thank Giuseppe Attardi for the help received to extract the parsing statistics. The second author has been partially supported by Ministerio de Ciencia e Innovación and FEDER (TIN2010-18552-C03-02).

References

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

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 143–151, Vancouver, Canada.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*, pages 195–203, Edinburgh, UK.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164, New York, USA.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and Head Automaton Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464, College Park, MD, USA.

Jason Eisner, Eric Goldlust, and Noah A. Smith. 2005. Compiling Comp Ling: Practical weighted dynamic programming and the Dyna language. In *Proceedings of HLT-EMNLP*, pages 281–290.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1492–1501, Uppsala, Sweden.

Carlos Gómez-Rodríguez, David J. Weir, and John Carroll. 2009. Parsing mildly non-projective dependency structures. In *Twelfth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 291–299, Athens, Greece.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics* (in press), 37(3).

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2000. *Introduction to Automata Theory*. Addison-Wesley, 2nd edition.

Liang Huang and Kenji 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, Sweden.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *36th Annual Meeting of the Association for Computational Linguistics and 18th International Conference on Computational Linguistics (COLING-ACL)*, pages 646–652, Montréal, Canada.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the IWPT*, pages 123–134.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of the EMNLP-CoNLL*, pages 141–150.

Marco Kuhlmann and Giorgio Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Twelfth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 478–486, Athens, Greece.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, Portland, Oregon, USA.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In Jacques Loecx, editor, *Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken, July 29–August 2, 1974*, number 14 in Lecture Notes in Computer Science, pages 255–269. Springer.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 40–51, Singapore.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Tenth International Conference on Parsing Technologies (IWPT)*, pages 121–132, Prague, Czech Republic.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Human Language Technology Conference (HLT) and Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–530, Vancouver, Canada.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

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

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

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

Stuart M. Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Khalil Sima'an. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of COLING*, pages 1175–1180.

David A. Smith and Noah A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of the EMNLP-CoNLL*, pages 132–140.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Proceedings of IJCAI*, pages 281–290.

Masaru Tomita. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Springer.

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