On the Complexity of Non-Projective Data-Driven Dependency Parsing

Ryan McDonald
Google Inc.
76 Ninth Avenue
New York, NY 10028
ryanmcd@google.com

Giorgio Satta
University of Padua
via Gradenigo 6/A
I-35131 Padova, Italy
satta@dei.unipd.it

Abstract

In this paper we investigate several non-projective parsing algorithms for dependency parsing, providing novel polynomial time solutions under the assumption that each dependency decision is independent of all the others, called here the edge-factored model. We also investigate algorithms for non-projective parsing that account for non-local information, and present several hardness results. This suggests that it is unlikely that exact non-projective dependency parsing is tractable for any model richer than the edge-factored model.

1 Introduction

Dependency representations of natural language are a simple yet flexible mechanism for encoding words and their syntactic dependencies through directed graphs. These representations have been thoroughly studied in descriptive linguistics (Tesnière, 1959; Hudson, 1984; Sgall et al., 1986; Meĺčuk, 1988) and have been applied in numerous language processing tasks. Figure 1 gives an example dependency graph for the sentence *Mr. Tomash will remain as a director emeritus*, which has been extracted from the Penn Treebank (Marcus et al., 1993). Each edge in this graph represents a single syntactic dependency directed from a word to its modifier. In this representation all edges are labeled with the specific syntactic function of the dependency, e.g., SBJ for subject and NMOD for modifier of a noun. To simplify computation and some important definitions, an artificial token is inserted into the sentence as the left most word and will always represent the root of the dependency graph. We assume all dependency graphs are directed trees originating out of a single node, which is a common constraint (Nivre, 2005).

The dependency graph in Figure 1 is an example of a nested or *projective* graph. Under the assumption that the root of the graph is the left most word of the sentence, a projective graph is one where the edges can be drawn in the plane above the sentence with no two edges crossing. Conversely, a *non-projective* dependency graph does not satisfy this property. Figure 2 gives an example of a non-projective graph for a sentence that has also been extracted from the Penn Treebank. Non-projectivity arises due to long distance dependencies or in languages with flexible word order. For many languages, a significant portion of sentences require a non-projective dependency analysis (Buchholz et al., 2006). Thus, the ability to learn and infer non-projective dependency graphs is an important problem in multilingual language processing.

Syntactic dependency parsing has seen a number of new learning and inference algorithms which have raised state-of-the-art parsing accuracies for many languages. In this work we focus on *data-driven* models of dependency parsing. These models are not driven by any underlying grammar, but instead learn to predict dependency graphs based on a set of parameters learned solely from a labeled corpus. The advantage of these models is that they negate the need for the development of grammars when adapting the model to new languages.

One interesting class of data-driven models are

Proceedings of the 10th Conference on Parsing Technologies, pages 121–132,
Prague, Czech Republic, June 2007. ©2007 Association for Computational Linguistics

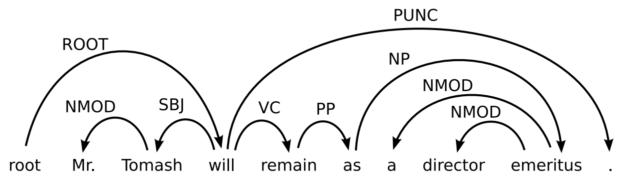

Figure 1: A projective dependency graph.

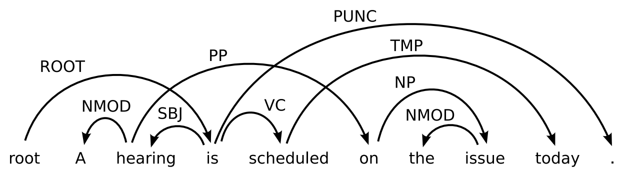

Figure 2: Non-projective dependency graph.

those that assume each dependency decision is independent modulo the global structural constraint that dependency graphs must be trees. Such models are commonly referred to as *edge-factored* since their parameters factor relative to individual edges of the graph (Paskin, 2001; McDonald et al., 2005a). Edge-factored models have many computational benefits, most notably that inference for non-projective dependency graphs can be achieved in polynomial time (McDonald et al., 2005b). The primary problem in treating each dependency as independent is that it is not a realistic assumption. Non-local information, such as arity (or valency) and neighbouring dependencies, can be crucial to obtaining high parsing accuracies (Klein and Manning, 2002; McDonald and Pereira, 2006). However, in the data-driven parsing setting this can be partially adverted by incorporating rich feature representations over the input (McDonald et al., 2005a).

The goal of this work is to further our current understanding of the computational nature of non-projective parsing algorithms for both learning and inference within the data-driven setting. We start by investigating and extending the edge-factored model of McDonald et al. (2005b). In particular, we appeal to the Matrix Tree Theorem for multi-digraphs to design polynomial-time algorithms for calculating both the partition function and edge expectations over all possible dependency graphs for a given sentence. To motivate these algorithms, we show that they can be used in many important learning and inference problems including min-risk decoding, training globally normalized log-linear models, syntactic language modeling, and unsupervised learning via the EM algorithm – none of which have previously been known to have exact non-projective implementations.

We then switch focus to models that account for non-local information, in particular arity and neighbouring parse decisions. For systems that model arity constraints we give a reduction from the Hamiltonian graph problem suggesting that the parsing problem is intractable in this case. For neighbouring parse decisions, we extend the work of McDonald and Pereira (2006) and show that modeling vertical neighbourhoods makes parsing intractable in addition to modeling horizontal neighbourhoods. A consequence of these results is that it is unlikely that exact non-projective dependency parsing is tractable for any model assumptions weaker than those made by the edge-factored models.

1.1 Related Work

There has been extensive work on data-driven dependency parsing for both projective parsing (Eisner, 1996; Paskin, 2001; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; McDonald et al., 2005a) and non-projective parsing systems (Nivre and Nilsson, 2005; Hall and Nóvák, 2005; McDonald et al., 2005b). These approaches can often be classified into two broad categories. In the first category are those methods that employ approximate inference, typically through the use of linear time shift-reduce parsing algorithms (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; Nivre and Nilsson, 2005). In the second category are those that employ exhaustive inference algorithms, usually by making strong independence assumptions, as is the case for edge-factored models (Paskin, 2001; McDonald et al., 2005a; McDonald et al., 2005b). Recently there have also been proposals for exhaustive methods that weaken the edge-factored assumption, including both approximate methods (McDonald and Pereira, 2006) and exact methods through integer linear programming (Riedel and Clarke, 2006) or branch-and-bound algorithms (Hirakawa, 2006).

For grammar based models there has been limited work on empirical systems for non-projective parsing systems, notable exceptions include the work of Wang and Harper (2004). Theoretical studies of note include the work of Neuhaus and Böker (1997) showing that the recognition problem for a minimal dependency grammar is hard. In addition, the work of Kahane et al. (1998) provides a polynomial parsing algorithm for a constrained class of non-projective structures. Non-projective dependency parsing can be related to certain parsing problems defined for phrase structure representations, as for instance immediate dominance CFG parsing (Barton et al., 1987) and shake-and-bake translation (Brew, 1992).

Independently of this work, Koo et al. (2007) and Smith and Smith (2007) showed that the Matrix-Tree Theorem can be used to train edge-factored log-linear models of dependency parsing. Both studies constructed implementations that compare favorably with the state-of-the-art. The work of Meilă and Jaakkola (2000) is also of note. In that study they use the Matrix Tree Theorem to develop a tractable bayesian learning algorithms for tree belief networks, which in many ways are closely related to probabilistic dependency parsing formalisms and the problems we address here.

2 Preliminaries

Let $L = \{l_1, \ldots, l_{|L|}\}$ be a set of permissible syntactic edge labels and $\boldsymbol{x} = x_0 x_1 \cdots x_n$ be a sentence such that x_0=root. From this sentence we construct a complete labeled directed graph (digraph) $G_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ such that,

- $V_{\boldsymbol{x}} = \{0, 1, \ldots, n\}$
- $E_{\boldsymbol{x}} = \{(i,j)^k \mid \forall\, i,j \in V_{\boldsymbol{x}} \text{ and } 1 \leq k \leq |L|\}$

$G_{\boldsymbol{x}}$ is a graph where each word in the sentence is a node, and there is a directed edge between every pair of nodes for every possible label. By its definition, $G_{\boldsymbol{x}}$ is a multi-digraph, which is a digraph that may have more than one edge between any two nodes. Let $(i,j)^k$ represent the k^{th} edge from i to j. $G_{\boldsymbol{x}}$ encodes all possible labeled dependencies between the words of $\boldsymbol{x}$. Thus every possible dependency graph of $\boldsymbol{x}$ must be a subgraph of $G_{\boldsymbol{x}}$.

Let $i \rightarrow^+ j$ be a relation that is true if and only if there is a non-empty directed path from node i to node j in some graph under consideration. A directed spanning tree[1] of a graph G, that originates out of node 0, is any subgraph $T = (V_T, E_T)$ such that,

- $V_T = V_{\boldsymbol{x}}$ and $E_T \subseteq E_{\boldsymbol{x}}$
- $\forall j \in V_T$, $0 \rightarrow^+ j$ if and only if $j \neq 0$
- If $(i,j)^k \in E_T$, then $(i',j)^{k'} \notin E_T$, $\forall i' \neq i$ and/or $k' \neq k$.

Define $T(G)$ as the set of all directed spanning trees for a graph G. As McDonald et al. (2005b) noted, there is a one-to-one correspondence between spanning trees of $G_{\boldsymbol{x}}$ and labeled dependency graphs of $\boldsymbol{x}$, i.e., $T(G_{\boldsymbol{x}})$ is exactly the set of all possible projective and non-projective dependency graphs for sentence $\boldsymbol{x}$. Throughout the rest of this paper, we will refer to any $T \in T(G_{\boldsymbol{x}})$ as a valid dependency graph for a sentence $\boldsymbol{x}$. Thus, by definition, every valid dependency graph must be a tree.

3 Edge-factored Models

In this section we examine the class of models that assume each dependency decision is independent. Within this setting, every edge in an induced graph $G_{\boldsymbol{x}}$ for a sentence $\boldsymbol{x}$ will have an associated weight $w_{ij}^k \geq 0$ that maps the k^{th} directed edge from node i to node j to a real valued numerical weight. These weights represents the likelihood of a dependency occurring from word w_i to word w_j with label l_k. Define the weight of a spanning tree $T = (V_T, E_T)$ as the product of the edge weights

$$w(T) = \prod_{(i,j)^k \in E_T} w_{ij}^k$$

It is easily shown that this formulation includes the projective model of Paskin (2001) and the non-projective model of McDonald et al. (2005b).

The definition of w_{ij}^k depends on the context in which it is being used. For example, in the work of McDonald et al. (2005b) it is simply a linear classifier that is a function of the words in the dependency, the label of the dependency, and any contextual features of the words in the sentence. In a generative probabilistic model (such as Paskin (2001)) it could represent the conditional probability of a word w_j being generated with a label l_k given that the word being modified is w_i (possibly with some other information such as the orientation of the dependency

[1] A directed spanning tree is commonly referred to as a arborescence in the graph theory literature.

or the number of words between w_i and w_j). We will attempt to make any assumptions about the form w_{ij}^k clear when necessary.

For the remainder of this section we discuss three crucial problems for learning and inference while showing that each can be computed tractably for the non-projective case.

3.1 Finding the Argmax

The first problem of interest is finding the highest weighted tree for a given input sentence $\boldsymbol{x}$

$$T \;=\; \operatorname*{argmax}_{T \in \mathcal{T}(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} w_{ij}^k$$

McDonald et al. (2005b) showed that this can be solved in $O(n^2)$ for unlabeled parsing using the Chu-Liu-Edmonds algorithm for standard digraphs (Chu and Liu, 1965; Edmonds, 1967). Unlike most exact projective parsing algorithms, which use efficient bottom-up chart parsing algorithms, the Chu-Liu-Edmonds algorithm is greedy in nature. It begins by selecting the single best incoming dependency edge for each node j. It then post-processes the resulting graph to eliminate cycles and then continues recursively until a spanning tree (or valid dependency graph) results (see McDonald et al. (2005b) for details).

The algorithm is trivially extended to the multi-digraph case for use in labeled dependency parsing. First we note that if the maximum directed spanning tree of a multi-digraph $G_{\boldsymbol{x}}$ contains any edge $(i,j)^k$, then we must have $k = k^* = \operatorname{argmax}_k w_{ij}^k$. Otherwise we could simply substitute $(i,j)^{k^*}$ in place of $(i,j)^k$ and obtain a higher weighted tree. Therefore, without effecting the solution to the argmax problem, we can delete all edges in $G_{\boldsymbol{x}}$ that do not satisfy this property. The resulting digraph is no longer a multi-digraph and the Chu-Liu-Edmonds algorithm can be applied directly. The new runtime is $O(|L|n^2)$.

As a side note, the k-best argmax problem for digraphs can be solved in $O(kn^2)$ (Camerini et al., 1980). This can also be easily extended to the multi-digraph case for labeled parsing.

3.2 Partition Function

A common step in many learning algorithms is to compute the sum over the weight of all the possible outputs for a given input $\boldsymbol{x}$. This value is often referred to as the *partition function* due to its similarity with a value by the same name in statistical mechanics. We denote this value as $Z_{\boldsymbol{x}}$,

$$Z_{\boldsymbol{x}} = \sum_{T \in \mathcal{T}(G_{\boldsymbol{x}})} w(T) = \sum_{T \in \mathcal{T}(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} w_{i,j}^k$$

To compute this sum it is possible to use the Matrix Tree Theorem for multi-digraphs,

Matrix Tree Theorem (Tutte, 1984): *Let G be a multi-digraph with nodes $V = \{0, 1, \ldots, n\}$ and edges E. Define (Laplacian) matrix Q as a $(n+1) \times (n+1)$ matrix indexed from 0 to n. For all i and j, define:*

$$Q_{jj} = \sum_{i \neq j, (i,j)^k \in E} w_{ij}^k \quad \& \quad Q_{ij} = \sum_{i \neq j, (i,j)^k \in E} -w_{ij}^k$$

If the i^{th} row and column are removed from Q to produce the matrix Q^i, then the sum of the weights of all directed spanning trees rooted at node i is equal to $|Q^i|$ (the determinant of Q^i).

Thus, if we construct Q for a graph $G_{\boldsymbol{x}}$, then the determinant of the matrix Q^0 is equivalent to $Z_{\boldsymbol{x}}$. The determinant of an $n \times n$ matrix can be calculated in numerous ways, most of which take $O(n^3)$ (Cormen et al., 1990). The most efficient algorithms for calculating the determinant of a matrix use the fact that the problem is no harder than matrix multiplication (Cormen et al., 1990). Matrix multiplication currently has known $O(n^{2.38})$ implementations and it has been widely conjectured that it can be solved in $O(n^2)$ (Robinson, 2005). However, most algorithms with sub-$O(n^3)$ running times require constants that are large enough to negate any asymptotic advantage for the case of dependency parsing. As a result, in this work we use $O(n^3)$ as the runtime for computing $Z_{\boldsymbol{x}}$.

Since it takes $O(|L|n^2)$ to construct the matrix Q, the entire runtime to compute $Z_{\boldsymbol{x}}$ is $O(n^3 + |L|n^2)$.

3.3 Edge Expectations

Another important problem for various learning paradigms is to calculate the expected value of each edge for an input sentence $\boldsymbol{x}$,

$$\langle (i,j)^k \rangle_{\boldsymbol{x}} = \sum_{T \in \mathcal{T}(G_{\boldsymbol{x}})} w(T) \times I((i,j)^k, T)$$

Input: $\boldsymbol{x} = x_0 x_1 \cdots x_n$

Construct Q	$O(\|L\|n^2)$
for j : 1 .. n	$O(n)$
$Q'_{jj} = Q_{jj}$ and $Q'_{ij} = Q_{ij}, 0 \leq \forall i \leq n$	$O(n)$
$Q_{jj} = 1$ and $Q_{ij} = 0, 0 \leq \forall i \leq n$	$O(n)$
for i : 0 .. n & $i \neq j$	$O(n)$
$Q_{ij} = -1$	$O(1)$
$Z_{\boldsymbol{x}} = \|Q^0\|$	$O(n^3)$
$\langle (i,j)^k \rangle_{\boldsymbol{x}} = w_{ij}^k Z_{\boldsymbol{x}}, \forall 1 \leq k \leq \|L\|$	$O(\|L\|)$
end for	
$Q_{jj} = Q'_{jj}$ and $Q_{ij} = Q'_{ij}, 0 \leq \forall i \leq n$	$O(n)$
end for	

Figure 3: Algorithm to calculate $\langle (i,j)^k \rangle_{\boldsymbol{x}}$ in $O(n^5 + |L|n^2)$.

where $I((i,j)^k, T)$ is an indicator function that is one when the edge $(i,j)^k$ is in the tree T.

To calculate the expectation for the edge $(i,j)^k$, we can simply eliminate all edges $(i',j)^{k'} \neq (i,j)^k$ from $G_{\boldsymbol{x}}$ and calculate $Z_{\boldsymbol{x}}$. $Z_{\boldsymbol{x}}$ will now be equal to the sum of the weights of all trees that contain $(i,j)^k$. A naive implementation to compute the expectation of all $|L|n^2$ edges takes $O(|L|n^5 + |L|^2n^4)$, since calculating $Z_{\boldsymbol{x}}$ takes $O(n^3 + |L|n^2)$ for a single edge. However, we can reduce this considerably by constructing Q a single time and only making modifications to it when necessary. An algorithm is given in Figure 3.3 that has a runtime of $O(n^5 + |L|n^2)$. This algorithm works by first constructing Q. It then considers edges from the node i to the node j. Now, assume that there is only a single edge from i to j and that that edge has a weight of 1. Furthermore assume that this edge is the only edge directed into the node j. In this case Q should be modified so that $Q_{jj} = 1$, $Q_{ij} = -1$, and $Q_{i'j} = 0$, $\forall i' \neq i, j$ (by the Matrix Tree Theorem). The value of $Z_{\boldsymbol{x}}$ under this new Q will be equivalent to the weight of all trees containing the single edge from i to j with a weight of 1. For a specific edge $(i,j)^k$ its expectation is simply $w_{ij}^k Z_{\boldsymbol{x}}$, since we can factor out the weight 1 edge from i to j in all the trees that contribute to $Z_{\boldsymbol{x}}$ and multiply through the actual weight for the edge. The algorithm then reconstructs Q and continues.

Following the work of Koo et al. (2007) and Smith and Smith (2007), it is possible to compute all expectations in $O(n^3 + |L|n^2)$ through matrix inversion. To make this paper self contained, we report here their algorithm adapted to our notation. First, consider the equivalence,

$$\begin{aligned} \frac{\partial \log Z_{\boldsymbol{x}}}{\partial w_{ij}^k} &= \frac{\partial \log Z_{\boldsymbol{x}}}{\partial Z_{\boldsymbol{x}}} \frac{\partial Z_{\boldsymbol{x}}}{\partial w_{ij}^k} \\ &= \frac{1}{Z_{\boldsymbol{x}}} \sum_{T \in T(G_{\boldsymbol{x}})} \frac{w(T)}{w_{ij}^k} \times I((i,j)^k, T) \end{aligned}$$

As a result, we can re-write the edge expectations as,

$$\langle (i,j)^k \rangle = Z_{\boldsymbol{x}} w_{ij}^k \frac{\partial \log Z_{\boldsymbol{x}}}{\partial w_{ij}^k} = Z_{\boldsymbol{x}} w_{ij}^k \frac{\partial \log |Q^0|}{\partial w_{ij}^k}$$

Using the chain rule, we get,

$$\frac{\partial \log |Q^0|}{\partial w_{ij}^k} = \sum_{i',j' \geq 1} \frac{\partial \log |Q^0|}{\partial (Q^0)_{i'j'}} \frac{\partial (Q^0)_{i'j'}}{\partial w_{ij}^k}$$

We assume the rows and columns of Q^0 are indexed from 1 so that the indexes of Q and Q^0 coincide. To calculate $\langle (i,j)^k \rangle$ when $i, j > 0$, we can use the fact that $\partial \log |X| / X_{ij} = (X^{-1})_{ji}$ and that $\partial (Q^0)_{i'j'} / \partial w_{ij}^k$ is non zero only when $i' = i$ and $j' = j$ or $i' = j' = j$ to get,

$$\langle (i,j)^k \rangle = Z_{\boldsymbol{x}} w_{ij}^k [((Q^0)^{-1})_{jj} - ((Q^0)^{-1})_{ji}]$$

When $i = 0$ and $j > 0$ the only non zero term of this sum is when $i' = j' = j$ and so

$$\langle (0,j)^k \rangle = Z_{\boldsymbol{x}} w_{0j}^k ((Q^0)^{-1})_{jj}$$

$Z_{\boldsymbol{x}}$ and $(Q^0)^{-1}$ can both be calculated a single time, each taking $O(n^3)$. Using these values, each expectation is computed in $O(1)$. Coupled with with the fact that we need to construct Q and compute the expectation for all $|L|n^2$ possible edges, in total it takes $O(n^3 + |L|n^2)$ time to compute all edge expectations.

3.4 Comparison with Projective Parsing

Projective dependency parsing algorithms are well understood due to their close connection to phrase-based chart parsing algorithms. The work of Eisner (1996) showed that the argmax problem for digraphs could be solved in $O(n^3)$ using a bottom-up dynamic programming algorithm similar to CKY. Paskin (2001) presented an $O(n^3)$ inside-outside algorithm for projective dependency parsing using the Eisner algorithm as its backbone. Using this algorithm it is trivial to calculate both $Z_{\boldsymbol{x}}$ and each

	Projective	Non-Projective
argmax	$O(n^3 + \|L\|n^2)$	$O(\|L\|n^2)$
$Z_{\boldsymbol{x}}$	$O(n^3 + \|L\|n^2)$	$O(n^3 + \|L\|n^2)$
$\langle (i,j)^k \rangle_{\boldsymbol{x}}$	$O(n^3 + \|L\|n^2)$	$O(n^3 + \|L\|n^2)$

Table 1: Comparison of runtime for non-projective and projective algorithms.

edge expectation. Crucially, the nested property of projective structures allows edge expectations to be computed in $O(n^3)$ from the inside-outside values. It is straight-forward to extend the algorithms of Eisner (1996) and Paskin (2001) to the labeled case adding only a factor of $O(|L|n^2)$.

Table 1 gives an overview of the computational complexity for the three problems considered here for both the projective and non-projective case. We see that the non-projective case compares favorably for all three problems.

4 Applications

To motivate the algorithms from Section 3, we present some important situations where each calculation is required.

4.1 Inference Based Learning

Many learning paradigms can be defined as inference-based learning. These include the perceptron (Collins, 2002) and its large-margin variants (Crammer and Singer, 2003; McDonald et al., 2005a). In these settings, a models parameters are iteratively updated based on the argmax calculation for a single or set of training instances under the current parameter settings. The work of McDonald et al. (2005b) showed that it is possible to learn a highly accurate non-projective dependency parser for multiple languages using the Chu-Liu-Edmonds algorithm for unlabeled parsing.

4.2 Non-Projective Min-Risk Decoding

In min-risk decoding the goal is to find the dependency graph for an input sentence $\boldsymbol{x}$, that on average has the lowest expected risk,

$$T = \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') \mathcal{R}(T, T')$$

where $\mathcal{R}$ is a risk function measuring the error between two graphs. Min-risk decoding has been studied for both phrase-structure parsing and dependency parsing (Titov and Henderson, 2006). In that work, as is common with many min-risk decoding schemes, $T(G_{\boldsymbol{x}})$ is not the entire space of parse structures. Instead, this set is usually restricted to a small number of possible trees that have been preselected by some baseline system. In this subsection we show that when the risk function is of a specific form, this restriction can be dropped. The result is an exact min-risk decoding procedure.

Let $\mathcal{R}(T, T')$ be the Hamming distance between two dependency graphs for an input sentence $\boldsymbol{x} = x_0 x_1 \cdots x_n$,

$$\mathcal{R}(T, T') = n \; - \sum_{(i,j)^k \in E_T} I((i,j)^k, T')$$

This is a common definition of risk between two graphs as it corresponds directly to labeled dependency parsing accuracy (McDonald et al., 2005a; Buchholz et al., 2006). Some algebra reveals,

$$\begin{aligned}
T &= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') \mathcal{R}(T, T') \\
&= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') [n \; - \sum_{(i,j)^k \in E_T} I((i,j)^k, T')] \\
&= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} - \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') \sum_{(i,j)^k \in E_T} I((i,j)^k, T') \\
&= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} - \sum_{(i,j)^k \in E_T} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') I((i,j)^k, T') \\
&= \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \sum_{(i,j)^k \in E_T} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') I((i,j)^k, T') \\
&= \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} e^{\sum_{T' \in T(G_{\boldsymbol{x}})} w(T') I((i,j)^k, T')} \\
&= \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} e^{\langle (i,j)^k \rangle_{\boldsymbol{x}}}
\end{aligned}$$

By setting the edge weights to $w^k_{ij} = e^{\langle (i,j)^k \rangle_{\boldsymbol{x}}}$ we can directly solve this problem using the edge expectation algorithm described in Section 3.3 and the argmax algorithm described in Section 3.1.

4.3 Non-Projective Log-Linear Models

Conditional Random Fields (CRFs) (Lafferty et al., 2001) are global discriminative learning algorithms for problems with structured output spaces, such as dependency parsing. For dependency parsing, CRFs would define the conditional probability of a dependency graph T for a sentence $\boldsymbol{x}$ as a globally normalized log-linear model,

$$p(T|\boldsymbol{x}) = \frac{\prod_{(i,j)^k \in E_T} e^{\mathbf{w}\cdot\mathbf{f}(i,j,k)}}{\sum_{T' \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_{T'}} e^{\mathbf{w}\cdot\mathbf{f}(i,j,k)}} = \frac{\prod_{(i,j)^k \in E_T} w_{ij}^k}{\sum_{T' \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_{T'}} w_{ij}^k} = \frac{w(T)}{Z_{\boldsymbol{x}}}$$

Here, the weights w_{ij}^k are potential functions over each edge defined as an exponentiated linear classifier with weight vector $\mathbf{w} \in \mathbb{R}^N$ and feature vector $\mathbf{f}(i,j,k) \in \mathbb{R}^N$, where $f_u(i,j,k) \in \mathbb{R}$ represents a single dimension of the vector $\mathbf{f}$. The denominator, which is exactly the sum over all graph weights, is a normalization constant forcing the conditional probability distribution to sum to one.

CRFs set the parameters $\mathbf{w}$ to maximize the log-likelihood of the conditional probability over a training set of examples $\mathcal{T} = \{(\boldsymbol{x}_\alpha, T_\alpha)\}_{\alpha=1}^{|\mathcal{T}|}$,

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_\alpha \log\ p(T_\alpha|\boldsymbol{x}_\alpha)$$

This optimization can be solved through a variety of iterative gradient based techniques. Many of these require the calculation of feature expectations over the training set under model parameters for the previous iteration. First, we note that the feature functions factor over edges, i.e., $f_u(T) = \sum_{(i,j)^k \in E_T} f_u(i,j,k)$. Because of this, we can use edge expectations to compute the expectation of every feature f_u. Let $\langle f_u \rangle_{\boldsymbol{x}_\alpha}$ represent the expectation of feature f_u for the training instance $\boldsymbol{x}_\alpha$,

$$\begin{aligned}
\langle f_u \rangle_{\boldsymbol{x}_\alpha} &= \sum_{T \in T(G_{\boldsymbol{x}_\alpha})} p(T|\boldsymbol{x}_\alpha) f_u(T) \\
&= \sum_{T \in T(G_{\boldsymbol{x}_\alpha})} p(T|\boldsymbol{x}_\alpha) \sum_{(i,j)^k \in E_T} f_u(i,j,k) \\
&= \sum_{T \in T(G_{\boldsymbol{x}_\alpha})} \frac{w(T)}{Z_{\boldsymbol{x}}} \sum_{(i,j)^k \in E_T} f_u(i,j,k) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{(i,j)^k \in E_{\boldsymbol{x}_\alpha}} \sum_{T \in T(G_{\boldsymbol{x}})} w(T) I((i,j)^k, T) f_u(i,j,k) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{(i,j)^k \in E_{\boldsymbol{x}_\alpha}} \langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha} f_u(i,j,k)
\end{aligned}$$

Thus, we can calculate the feature expectation per training instance using the algorithms for computing $Z_{\boldsymbol{x}}$ and edge expectations. Using this, we can calculate feature expectations over the entire training set,

$$\langle f_u \rangle_{\mathcal{T}} = \sum_\alpha p(\boldsymbol{x}_\alpha) \langle f_u \rangle_{\boldsymbol{x}_\alpha}$$

where $p(\boldsymbol{x}_\alpha)$ is typically set to $1/|\mathcal{T}|$.

4.4 Non-projective Generative Parsing Models

A generative probabilistic dependency model over some alphabet Σ consists of parameters $p_{x,y}^k$ associated with each dependency from word $x \in \Sigma$ to word $y \in \Sigma$ with label $l_k \in L$. In addition, we impose $0 \leq p_{x,y}^k \leq 1$ and the normalization conditions $\sum_{y,k} p_{x,y}^k = 1$ for each $x \in \Sigma$. We define a generative probability model p over trees $T \in T(G_{\boldsymbol{x}})$ and a sentence $\boldsymbol{x} = x_0 x_1 \cdots x_n$ conditioned on the sentence length, which is always known,

$$\begin{aligned}
p(\boldsymbol{x}, T|n) &= p(\boldsymbol{x}|T,n) p(T|n) \\
&= \prod_{(i,j)^k \in E_T} p_{x_i,x_j}^k\ p(T|n)
\end{aligned}$$

We assume that $p(T|n) = \beta$ is uniform. This model is studied specifically by Paskin (2001). In this model, one can view the sentence as being generated recursively in a top-down process. First, a tree is generated from the distribution $p(T|n)$. Then starting at the root of the tree, every word generates all of its modifiers independently in a recursive breadth-first manner. Thus, $p_{x,y}^k$ represents the probability of the word x generating its modifier y with label l_k. This distribution is usually smoothed and is often conditioned on more information including the orientation of x relative to y (i.e., to the left/right) and distance between the two words. In the supervised setting this model can be trained with maximum likelihood estimation, which amounts to simple counts over the data. Learning in the unsupervised setting requires EM and is discussed in Section 4.4.2.

Another generative dependency model of interest is that given by Klein and Manning (2004). In this model the sentence and tree are generated jointly, which allows one to drop the assumption that $p(T|n)$ is uniform. This requires the addition to the model of parameters $p_{x,\text{STOP}}$ for each $x \in \Sigma$, with the normalization condition $p_{x,\text{STOP}} + \sum_{y,k} p_{x,y}^k = 1$. It is possible to extend the model of Klein and Manning

(2004) to the non-projective case. However, the resulting distribution will be over multisets of words from the alphabet instead of strings. The discussion in this section is stated for the model in Paskin (2001); a similar treatment can be developed for the model in Klein and Manning (2004).

4.4.1 Language Modeling

A generative model of dependency structure might be used to determine the probability of a sentence $\boldsymbol{x}$ by marginalizing out all possible dependency trees,

$$\begin{aligned} p(\boldsymbol{x}|n) &= \sum_{T \in T(G_{\boldsymbol{x}})} p(\boldsymbol{x}, T|n) \\ &= \sum_{T \in T(G_{\boldsymbol{x}})} p(\boldsymbol{x}|T, n)p(T|n) \\ &= \beta \sum_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} p^k_{x_i,x_j} = \beta Z_{\boldsymbol{x}} \end{aligned}$$

This probability can be used directly as a non-projective syntactic language model (Chelba et al., 1997) or possibly interpolated with a separate n-gram model.

4.4.2 Unsupervised Learning

In unsupervised learning we train our model on a sample of unannotated sentences $\mathcal{X} = \{\boldsymbol{x}_\alpha\}_{\alpha=1}^{|\mathcal{X}|}$. Let $|\boldsymbol{x}_\alpha| = n_\alpha$ and $p(T|n_\alpha) = \beta_\alpha$. We choose the parameters that maximize the log-likelihood

$$\begin{aligned} &\sum_{\alpha=1}^{|\mathcal{X}|} \log(p(\boldsymbol{x}_\alpha|n_\alpha)) = \\ &= \sum_{\alpha=1}^{|\mathcal{X}|} \log\Big(\sum_{T \in T(G_{\boldsymbol{x}_\alpha})} p(\boldsymbol{x}_\alpha|T, n_\alpha)\Big) + \sum_{\alpha=1}^{|\mathcal{X}|} \log(\beta_\alpha), \end{aligned}$$

viewed as a function of the parameters and subject to the normalization conditions, i.e., $\sum_{y,k} p^k_{x,y} = 1$ and $p^k_{x,y} \geq 0$.

Let $x_{\alpha i}$ be the i^{th} word of $\boldsymbol{x}_\alpha$. By solving the above constrained optimization problem with the usual Lagrange multipliers method one gets

$$\begin{aligned} &p^k_{x,y} = \\ &= \frac{\sum_{\alpha=1}^{|\mathcal{X}|} \frac{1}{Z_{\boldsymbol{x}_\alpha}} \sum_{\substack{i\,:\,x_{\alpha i} = x, \\ j\,:\,x_{\alpha j} = y}} \langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha}}{\sum_{\alpha=1}^{|\mathcal{X}|} \frac{1}{Z_{\boldsymbol{x}_\alpha}} \sum_{y',k'} \sum_{\substack{i\,:\,x_{\alpha i} = x, \\ j'\,:\,x_{\alpha j'} = y'}} \langle (i,j')^{k'} \rangle_{\boldsymbol{x}_\alpha}}, \end{aligned}$$

where for each $\boldsymbol{x}_\alpha$ the expectation $\langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha}$ is defined as in Section 3, but with the weight $w(T)$ replaced by the probability distribution $p(\boldsymbol{x}_\alpha|T, n_\alpha)$.

The above $|L| \cdot |\Sigma|^2$ relations represent a non-linear system of equations. There is no closed form solution in the general case, and one adopts the expectation maximization (EM) method, which is a specialization of the standard fixed-point iteration method for the solution of non-linear systems. We start with some initial assignment of the parameters and at each iteration we use the induced distribution $p(\boldsymbol{x}_\alpha|T, n_\alpha)$ to compute a refined value for the parameters themselves. We are always guaranteed that the Kullback-Liebler divergence between two approximated distributions computed at successive iterations does not increase, which implies the convergence of the method to some local maxima (with the exception of saddle points).

Observe that at each iteration we can compute quantities $\langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha}$ and $Z_{\boldsymbol{x}_\alpha}$ in polynomial time using the algorithms from Section 3 with $p^k_{x_{\alpha i},x_{\alpha j}}$ in place of $w^k_{i,j}$. Furthermore, under some standard conditions the fixed-point iteration method guarantees a constant number of bits of precision gain for the parameters at each iteration, resulting in overall polynomial time computation in the size of the input and in the required number of bits for the precision. As far as we know, this is the first EM learning algorithm for the model in Paskin (2001) working in the non-projective case. The projective case has been investigated in Paskin (2001).

5 Beyond Edge-factored Models

We have shown that several computational problems related to parsing can be solved in polynomial time for the class of non-projective dependency models with the assumption that dependency relations are mutually independent. These independence assumptions are unwarranted, as it has already been established that modeling non-local information such as arity and nearby parsing decisions improves the accuracy of dependency models (Klein and Manning, 2002; McDonald and Pereira, 2006).

In the spirit of our effort to understand the nature of exact non-projective algorithms, we examine dependency models that introduce arity constraints as well as permit edge decisions to be dependent on a limited neighbourhood of other edges in the graph. Both kinds of models can no longer be considered edge-factored, since the likelihood of a dependency occurring in a particular analysis is now dependent on properties beyond the edge itself.

5.1 Arity

One feature of the edge-factored models is that no restriction is imposed on the arity of the nodes in the dependency trees. As a consequence, these models can generate dependency trees of unbounded arity. We show below that this is a crucial feature in the development of the complexity results we have obtained in the previous sections.

Let us assume a graph $G_{\boldsymbol{x}}^{(\phi)} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ defined as before, but with the additional condition that each node $i \in V_{\boldsymbol{x}}$ is associated with an integer value $\phi(i) \geq 0$. $T(G_{\boldsymbol{x}}^{(\phi)})$ is now defined as the set of all directed spanning trees for $G_{\boldsymbol{x}}^{(\phi)}$ rooted in node 0, such that every node $i \in V_{\boldsymbol{x}}$ has arity smaller than or equal to $\phi(i)$. We now introduce a construction that will be used to establish several hardness results for the computational problems discussed in this paper. Recall that a Hamiltonian path in a directed graph G is a directed path that visits all of the nodes of G exactly once.

Let G be some directed graph with set of nodes $V = \{1, 2, \ldots, n\}$. We construct a target graph $G_{\boldsymbol{x}}^{(\phi)} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ with $V_{\boldsymbol{x}} = V \cup \{0\}$ (0 the root node) and $|L| = 1$. For each $i, j \in V_{\boldsymbol{x}}$ with $i \neq j$, we add an edge $(i, j)^1$ to $E_{\boldsymbol{x}}$. We set $w_{i,j}^1 = 1$ if there is an edge from i to j in G, or else if i or j is the root node 0, and $w_{i,j}^1 = 0$ otherwise. Furthermore, we set $\phi(i) = 1$ for each $i \in V_{\boldsymbol{x}}$. This construction can be clearly carried out in log-space.

Note that each $T \in T(G_{\boldsymbol{x}}^{(\phi)})$ must be a monadic tree with weight equal to either 0 or 1. It is not difficult to see that if $w(T) = 1$, then when we remove the root node 0 from T we obtain a Hamiltonian path in G. Conversely, each Hamiltonian path in G can be extended to a spanning tree $T \in T(G_{\boldsymbol{x}}^{(\phi)})$ with $w(T) = 1$, by adding the root node 0.

Using the above observations, it can be shown that the solution of the argmax problem for $G_{\boldsymbol{x}}^{(\phi)}$ provides some Hamiltonian directed path in G. The latter search problem is FNP-hard, and is unlikely to be solved in polynomial time. Furthermore, quantity $Z_{\boldsymbol{x}}$ provides the count of the Hamiltonian directed paths in G, and for each $i \in V$, the expectation $\langle (0, i)^1 \rangle_{\boldsymbol{x}}$ provides the count of the Hamiltonian directed paths in G starting from node i. Both these counting problems are #P-hard, and very unlikely to have polynomial time solutions.

This result helps to relate the hardness of data-driven models to the commonly known hardness results in the grammar-driven literature given by Neuhaus and Böker (1997). In that work, an arity constraint is included in their minimal grammar.

5.2 Vertical and Horizontal Markovization

In general, we would like to say that every dependency decision is dependent on every other edge in a graph. However, modeling dependency parsing in such a manner would be a computational nightmare. Instead, we would like to make a Markov assumption over the edges of the tree, in a similar way that a Markov assumption can be made for sequential classification problems in order to ensure tractable learning and inference.

Klein and Manning (2003) distinguish between two kinds of Markovization for unlexicalized CFG parsing. The first is vertical Markovization, which makes the generation of a non-terminal dependent on other non-terminals that have been generated at different levels in the phrase-structure tree. The second is horizontal Markovization, which makes the generation of a non-terminal dependent on other non-terminals that have been generated at the same level in the tree.

For dependency parsing there are analogous notions of vertical and horizontal Markovization for a given edge $(i, j)^k$. First, let us define the vertical and horizontal *neighbourhoods* of $(i, j)^k$. The vertical neighbourhood includes all edges in any path from the root to a leaf that passes through $(i, j)^k$. The horizontal neighbourhood contains all edges $(i, j')^{k'}$. Figure 4 graphically displays the vertical and horizontal neighbourhoods for an edge in the dependency graph from Figure 1.

Vertical and horizontal Markovization essentially allow the score of the graph to factor over a larger scope of edges, provided those edges are in the same vertical or horizontal neighbourhood. A d^{th} order factorization is one in which the score factors only over the d nearest edges in the neighbourhoods. In

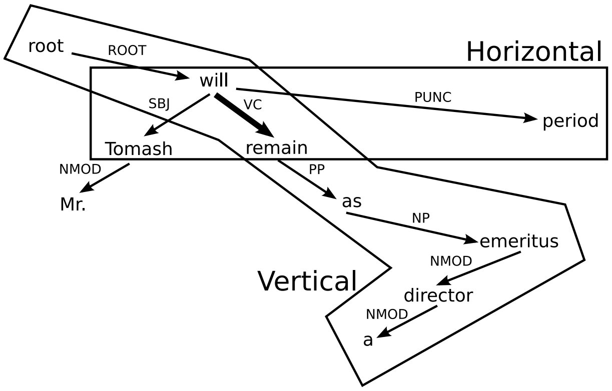

Figure 4: Vertical and Horizontal neighbourhood for the edge from *will* to *remain*.

McDonald and Pereira (2006), it was shown that non-projective dependency parsing with horizontal Markovization is FNP-hard. In this study we complete the picture and show that vertical Markovization is also FNP-hard.

Consider a first-order vertical Markovization in which the score for a dependency graph factors over pairs of vertically adjacent edges[2],

$$w(T) = \prod_{(h,i)^k,(i,j)^{k'} \in E_T} {}^{k}_{hi}w^{k'}_{ij}$$

where ${}^{k}_{hi}w^{k'}_{ij}$ is the weight of including both edges $(h,i)^k$ and $(i,j)^{k'}$ in the dependency graph. Note that this formulation does not include any contributions from dependencies that have no vertically adjacent neighbours, i.e., any edge $(0,i)^k$ such that there is no edge $(i,j)^{k'}$ in the graph. We can easily rectify this by inserting a second root node, say $0'$, and including the weights ${}^{k}_{0'0}w^{k'}_{0i}$. To ensure that only valid dependency graphs get a weight greater than zero, we can set ${}^{k}_{hi}w^{k'}_{ij} = 0$ if $i = 0'$ and ${}^{k}_{0'i}w^{k'}_{ij} = 0$ if $i \neq 0$.

Now, consider the NP-complete 3D-matching problem (3DM). As input we are given three sets of size m, call them A, B and C, and a set $S \subseteq A \times B \times C$. The 3DM problem asks if there is a set $S' \subseteq S$ such that $|S'| = m$ and for any two tuples $(a,b,c), (a',b',c') \in S'$ it is the case that $a \neq a'$, $b \neq b'$, and $c \neq c'$.

[2]McDonald and Pereira (2006) define this as a second-order Markov assumption. This is simply a difference in terminology and does not represent any meaningful distinction.

We can reduce the 3D-matching problem to the first-order vertical Markov parsing problem by constructing a graph $G = (V, E)$, such that $L = A \cup B \cup C$, $V = \{0', 0\} \cup A \cup B \cup C$ and $E = \{(i,j)^k \,|\, i,j \in V,\ k \in L\}$. The set E contains multiple edges between ever pair of nodes, each edge taking on a label representing a single element of the set $A \cup B \cup C$. Now, define ${}^{k}_{0'0}w^{k'}_{0a} = 1$, for all $a \in A$ and $k, k' \in A \cup B \cup C$, and ${}^{b}_{0a}w^{c}_{ab} = 1$, for all $a \in A$ and $b \in B$ and $c \in C$, and ${}^{c}_{ab}w^{c}_{bc} = 1$, for all $(a,b,c) \in S$. All other weights are set to zero.

We show below that there exists a bijection between the set of valid 3DMs for S and the set of non-zero weighted dependency graphs in $T(G)$. First, it is easy to show that for any 3DM S', there is a representative dependency graph that has a weight of 1. This graph simply consists of the edges $(0,a)^b$, $(a,b)^c$, and $(b,c)^c$, for all $(a,b,c) \in S'$, plus an arbitrarily labeled edge from $0'$ to 0.

To prove the reverse, consider a graph with weight 1. This graph must have a weight 1 edge into the node a of the form $(0,a)^b$ since the graph must be spanning. By the definition of the weight function, in any non-zero weighted tree, a must have a single outgoing edge, and that edge must be directed into the node b. Let's say that this edge is $(a,b)^c$. Then again by the weighting function, in any non-zero weighted graph, b must have a single outgoing edge that is directed into c, in particular the edge $(b,c)^c$. Thus, for any node a, there is a single path directed out of it to a single leaf $c \in C$. We can then state that the only non-zero weighted dependency graph is one where each $a \in A$, $b \in B$ and $c \in C$ occurs in exactly one of m disjoint paths from the root of the form $0 \rightarrow a \rightarrow b \rightarrow c$. This is because the label of the single edge going into node a will determine exactly the node b that the one outgoing edge from a must go into. The label of that edge determines exactly the single outgoing edge from b into some node c. Now, since the weighting function ensures that the only non-zero weighted paths into any leaf node c correspond directly to elements of S, each of the m disjoint paths represent a single tuple in a 3DM. Thus, if there is a non-zero weighted graph in $T(G)$, then it must directly correspond to a valid 3DM, which concludes the proof.

Note that any d^{th} order Markovization can be embedded into a $d+1^{th}$ Markovization. Thus, this result also holds for any arbitrary Markovization.

6 Discussion

In this paper we have shown that many important learning and inference problems can be solved efficiently for non-projective edge-factored dependency models by appealing to the Matrix Tree Theorem for multi-digraphs. These results extend the work of McDonald et al. (2005b) and help to further our understanding of when exact non-projective algorithms can be employed. When this analysis is coupled with the projective parsing algorithms of Eisner (1996) and Paskin (2001) we begin to get a clear picture of the complexity for data-driven dependency parsing within an edge-factored framework. To further justify the algorithms presented here, we outlined a few novel learning and inference settings in which they are required.

However, for the non-projective case, moving beyond edge-factored models will almost certainly lead to intractable parsing problems. We have provided further evidence for this by proving the hardness of incorporating arity constraints and horizontal/vertical edge Markovization, both of which incorporate information unavailable to an edge-factored model. The hardness results provided here are also of interest since both arity constraints and Markovization can be incorporated efficiently in the projective case through the straight-forward augmentation of the underlying chart parsing algorithms used in the projective edge-factored models. This highlights a fundamental difference between the nature of projective parsing algorithms and non-projective parsing algorithms. On the projective side, all algorithms use a bottom-up chart parsing framework to search the space of nested constructions. On the non-projective side, algorithms are either greedy-recursive in nature (i.e., the Chu-Liu-Edmonds algorithm) or based on the calculation of the determinant of a matrix (i.e., the partition function and edge expectations).

Thus, the existence of bottom-up chart parsing algorithms for projective dependency parsing provides many advantages. As mentioned above, it permits simple augmentation techniques to incorporate non-local information such as arity constraints and Markovization. It also ensures the compatibility of projective parsing algorithms with many important natural language processing methods that work within a bottom-up chart parsing framework, including information extraction (Miller et al., 2000) and syntax-based machine translation (Wu, 1996).

The complexity results given here suggest that polynomial chart-parsing algorithms do not exist for the non-projective case. Otherwise we should be able to augment them and move beyond edge-factored models without encountering intractability – just like the projective case. An interesting line of research is to investigate classes of non-projective structures that can be parsed with chart-parsing algorithms and how these classes relate to the languages parsable by other syntactic formalisms.

Acknowledgments

Thanks to Ben Taskar for pointing out the work of Meilă and Jaakkola (2000). Thanks to David Smith, Noah Smith and Michael Collins for making drafts of their EMNLP papers available.

References

G. E. Barton, R. C. Berwick, and E. S. Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA.

C. Brew. 1992. Letting the cat out of the bag: Generation for Shake-and-Bake MT. In *Proc. COLING*.

S. Buchholz, E. Marsi, A. Dubey, and Y. Krymolowski. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. CoNLL*.

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

C. Chelba, D. Engle, F. Jelinek, V. Jimenez, S. Khudanpur, L. Mangu, H. Printz, E.S. Ristad, R. Rosenfeld, A. Stolcke, and D. Wu. 1997. Structure and performance of a dependency language model. In *Eurospeech*.

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

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

T.H. Cormen, C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*.

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

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. COLING*.

K. Hall and V. Nóvák. 2005. Corrective modeling for non-projective dependency parsing. In *Proc. IWPT*.

H. Hirakawa. 2006. Graph branch algorithm: An optimum tree search method for scored dependency graph with arc co-occurrence constraints. In *Proc. ACL*.

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

S. Kahane, A. Nasr, and O Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proc. ACL*.

D. Klein and C.D. Manning. 2002. Fast exact natural language parsing with a factored model. In *Proc. NIPS*.

D. Klein and C. Manning. 2003. Accurate unlexicalized parsing. In *Proc. ACL*.

D. Klein and C. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proc. ACL*.

T. Koo, A. Globerson, X. Carreras, and M. Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proc. EMNLP*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc EACL*.

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

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

M. Meilă and T. Jaakkola. 2000. Tractable Bayesian learning of tree belief networks. In *Proc. UAI*.

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

S. Miller, H. Fox, L.A. Ramshaw, and R.M. Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *Proc NAACL*, pages 226–233.

P. Neuhaus and N. Böker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *Proc. ACL*.

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

J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of english text. In *Proc. COLING*.

J. Nivre. 2005. Dependency grammar and dependency parsing. Technical Report MSI report 05133, Växjö University: School of Mathematics and Systems Engineering.

M.A. Paskin. 2001. Cubic-time parsing and learning algorithms for grammatical bigram models. Technical Report UCB/CSD-01-1148, Computer Science Division, University of California Berkeley.

S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proc. EMNLP*.

S. Robinson. 2005. Toward an optimal algorithm for matrix multiplication. *News Journal of the Society for Industrial and Applied Mathematics*, 38(9).

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

D.A. Smith and N.A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proc. EMNLP*.

L. Tesnière. 1959. *Éléments de syntaxe structurale*. Editions Klincksieck.

I. Titov and J. Henderson. 2006. Bayes risk minimization in natural language parsing. University of Geneva technical report.

W.T. Tutte. 1984. *Graph Theory*. Cambridge University Press.

W. Wang and M. P. Harper. 2004. A statistical constraint dependency grammar (CDG) parser. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*.

D. Wu. 1996. A polynomial-time algorithm for statistical machine translation. In *Proc. ACL*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. IWPT*.