Pseudo-Projective Dependency Parsing

Joakim Nivre and **Jens Nilsson**
School of Mathematics and Systems Engineering
Växjö University
SE-35195 Växjö, Sweden
{nivre,jni}@msi.vxu.se

Abstract

In order to realize the full potential of dependency-based syntactic parsing, it is desirable to allow non-projective dependency structures. We show how a data-driven deterministic dependency parser, in itself restricted to projective structures, can be combined with graph transformation techniques to produce non-projective structures. Experiments using data from the Prague Dependency Treebank show that the combined system can handle non-projective constructions with a precision sufficient to yield a significant improvement in overall parsing accuracy. This leads to the best reported performance for robust non-projective parsing of Czech.

1 Introduction

It is sometimes claimed that one of the advantages of dependency grammar over approaches based on constituency is that it allows a more adequate treatment of languages with variable word order, where discontinuous syntactic constructions are more common than in languages like English (Mel'čuk, 1988; Covington, 1990). However, this argument is only plausible if the formal framework allows non-projective dependency structures, i.e. structures where a head and its dependents may correspond to a discontinuous constituent. From the point of view of computational implementation this can be problematic, since the inclusion of non-projective structures makes the parsing problem more complex and therefore compromises efficiency and in practice also accuracy and robustness. Thus, most broad-coverage parsers based on dependency grammar have been restricted to projective structures. This is true of the widely used link grammar parser for English (Sleator and Temperley, 1993), which uses a dependency grammar of sorts, the probabilistic dependency parser of Eisner (1996), and more recently proposed deterministic dependency parsers (Yamada and Matsumoto, 2003; Nivre et al., 2004). It is also true of the adaptation of the Collins parser for Czech (Collins et al., 1999) and the finite-state dependency parser for Turkish by Oflazer (2003).

This is in contrast to dependency treebanks, e.g. Prague Dependency Treebank (Hajič et al., 2001b), Danish Dependency Treebank (Kromann, 2003), and the METU Treebank of Turkish (Oflazer et al., 2003), which generally allow annotations with non-projective dependency structures. The fact that projective dependency parsers can never exactly reproduce the analyses found in non-projective treebanks is often neglected because of the relative scarcity of problematic constructions. While the proportion of sentences containing non-projective dependencies is often 15–25%, the total proportion of non-projective arcs is normally only 1–2%. As long as the main evaluation metric is dependency accuracy per word, with state-of-the-art accuracy mostly below 90%, the penalty for not handling non-projective constructions is almost negligible. Still, from a theoretical point of view, projective parsing of non-projective structures has the drawback that it rules out perfect accuracy even as an asymptotic goal.

Proceedings of the 43rd Annual Meeting of the ACL, pages 99–106,
Ann Arbor, June 2005.

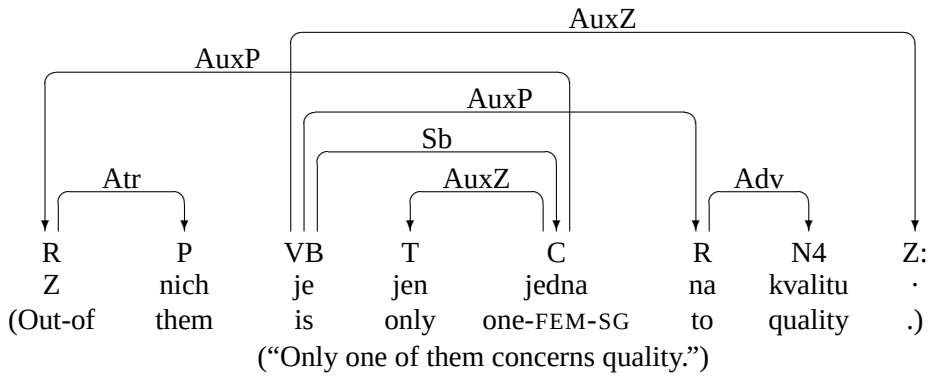

Figure 1: Dependency graph for Czech sentence from the Prague Dependency Treebank[1]

There exist a few robust broad-coverage parsers that produce non-projective dependency structures, notably Tapanainen and Järvinen (1997) and Wang and Harper (2004) for English, Foth et al. (2004) for German, and Holan (2004) for Czech. In addition, there are several approaches to non-projective dependency parsing that are still to be evaluated in the large (Covington, 1990; Kahane et al., 1998; Duchier and Debusmann, 2001; Holan et al., 2001; Hellwig, 2003). Finally, since non-projective constructions often involve long-distance dependencies, the problem is closely related to the recovery of empty categories and non-local dependencies in constituency-based parsing (Johnson, 2002; Dienes and Dubey, 2003; Jijkoun and de Rijke, 2004; Cahill et al., 2004; Levy and Manning, 2004; Campbell, 2004).

In this paper, we show how non-projective dependency parsing can be achieved by combining a data-driven projective parser with special graph transformation techniques. First, the training data for the parser is projectivized by applying a minimal number of lifting operations (Kahane et al., 1998) and encoding information about these lifts in arc labels. When the parser is trained on the transformed data, it will ideally learn not only to construct projective dependency structures but also to assign arc labels that encode information about lifts. By applying an inverse transformation to the output of the parser, arcs with non-standard labels can be lowered to their proper place in the dependency graph, giving rise to non-projective structures. We call this pseudo-projective dependency parsing, since it is based on a notion of pseudo-projectivity (Kahane et al., 1998).

The rest of the paper is structured as follows. In section 2 we introduce the graph transformation techniques used to projectivize and deprojectivize dependency graphs, and in section 3 we describe the data-driven dependency parser that is the core of our system. We then evaluate the approach in two steps. First, in section 4, we evaluate the graph transformation techniques in themselves, with data from the Prague Dependency Treebank and the Danish Dependency Treebank. In section 5, we then evaluate the entire parsing system by training and evaluating on data from the Prague Dependency Treebank.

2 Dependency Graph Transformations

We assume that the goal in dependency parsing is to construct a labeled dependency graph of the kind depicted in Figure 1. Formally, we define dependency graphs as follows:

1. Let $R = \{r_1, \ldots, r_m\}$ be the set of permissible dependency types (arc labels).
2. A dependency graph for a string of words $W = w_1 \cdots w_n$ is a labeled directed graph $D = (W, A)$, where
   (a) W is the set of nodes, i.e. word tokens in the input string, ordered by a linear precedence relation $<$,
   (b) A is a set of labeled arcs (w_i, r, w_j), where $w_i, w_j \in W$, $r \in R$,
   (c) for every $w_j \in W$, there is at most one arc $(w_i, r, w_j) \in A$.

[1] The dependency graph has been modified to make the final period a dependent of the main verb instead of being a dependent of a special root node for the sentence.

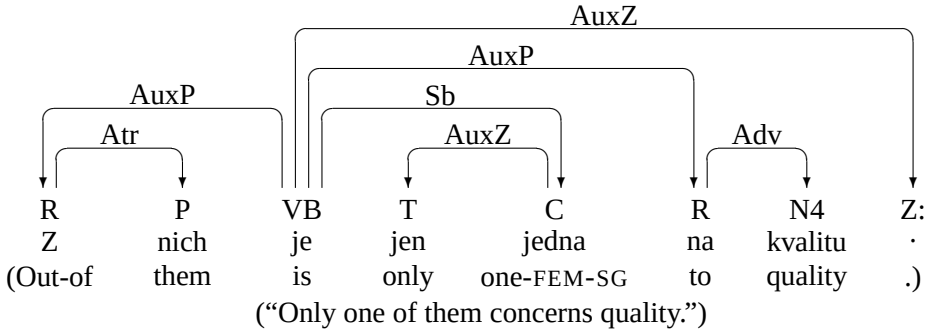

Figure 2: Projectivized dependency graph for Czech sentence

3. A graph $D = (W, A)$ is well-formed iff it is acyclic and connected.

If $(w_i, r, w_j) \in A$, we say that w_i is the head of w_j and w_j a dependent of w_i. In the following, we use the notation $w_i \xrightarrow{r} w_j$ to mean that $(w_i, r, w_j) \in A$; we also use $w_i \rightarrow w_j$ to denote an arc with unspecified label and $w_i \rightarrow^* w_j$ for the reflexive and transitive closure of the (unlabeled) arc relation.

The dependency graph in Figure 1 satisfies all the defining conditions above, but it fails to satisfy the condition of projectivity (Kahane et al., 1998):

1. An arc $w_i \rightarrow w_k$ is projective iff, for every word w_j occurring between w_i and w_k in the string ($w_i < w_j < w_k$ or $w_i > w_j > w_k$), $w_i \rightarrow^* w_j$.

2. A dependency graph $D = (W, A)$ is projective iff every arc in A is projective.

The arc connecting the head *jedna* (one) to the dependent *Z* (out-of) spans the token *je* (is), which is not dominated by *jedna*.

As observed by Kahane et al. (1998), any (non-projective) dependency graph can be transformed into a projective one by a lifting operation, which replaces each non-projective arc $w_j \rightarrow w_k$ by a projective arc $w_i \rightarrow w_k$ such that $w_i \rightarrow^* w_j$ holds in the original graph. Here we use a slightly different notion of lift, applying to individual arcs and moving their head upwards one step at a time:

$$\textsc{Lift}(w_j \rightarrow w_k) \quad = \quad \begin{cases} w_i \rightarrow w_k \text{ if } w_i \rightarrow w_j \\ \text{undefined otherwise} \end{cases}$$

Intuitively, lifting an arc makes the word w_k dependent on the head w_i of its original head w_j (which is unique in a well-formed dependency graph), unless w_j is a root in which case the operation is undefined (but then $w_j \rightarrow w_k$ is necessarily projective if the dependency graph is well-formed).

Projectivizing a dependency graph by lifting non-projective arcs is a nondeterministic operation in the general case. However, since we want to preserve as much of the original structure as possible, we are interested in finding a transformation that involves a minimal number of lifts. Even this may be nondeterministic, in case the graph contains several non-projective arcs whose lifts interact, but we use the following algorithm to construct a minimal projective transformation $D' = (W, A')$ of a (non-projective) dependency graph $D = (W, A)$:

PROJECTIVIZE(W, A)
1 $A' \leftarrow A$
2 **while** (W, A') is non-projective
3 $\quad a \leftarrow$ SMALLEST-NONP-ARC(A')
4 $\quad A' \leftarrow (A' - \{a\}) \cup \{\textsc{Lift}(a)\}$
5 **return** (W, A')

The function SMALLEST-NONP-ARC returns the non-projective arc with the shortest distance from head to dependent (breaking ties from left to right). Applying the function PROJECTIVIZE to the graph in Figure 1 yields the graph in Figure 2, where the problematic arc pointing to *Z* has been lifted from the original head *jedna* to the ancestor *je*. Using the terminology of Kahane et al. (1998), we say that *jedna* is the *syntactic head* of *Z*, while *je* is its *linear head* in the projectivized representation.

Unlike Kahane et al. (1998), we do not regard a projectivized representation as the final target of the parsing process. Instead, we want to apply an in-

	Lifted arc label	Path labels	Number of labels
Baseline	d	p	n
Head	$d{\uparrow}h$	p	$n(n+1)$
Head+Path	$d{\uparrow}h$	$p{\downarrow}$	$2n(n+1)$
Path	$d{\uparrow}$	$p{\downarrow}$	$4n$

Table 1: Encoding schemes (d = dependent, h = syntactic head, p = path; n = number of dependency types)

verse transformation to recover the underlying (non-projective) dependency graph. In order to facilitate this task, we extend the set of arc labels to encode information about lifting operations. In principle, it would be possible to encode the exact position of the syntactic head in the label of the arc from the linear head, but this would give a potentially infinite set of arc labels and would make the training of the parser very hard. In practice, we can therefore expect a trade-off such that increasing the amount of information encoded in arc labels will cause an increase in the accuracy of the inverse transformation but a decrease in the accuracy with which the parser can construct the labeled representations. To explore this tradeoff, we have performed experiments with three different encoding schemes (plus a baseline), which are described schematically in Table 1.

The baseline simply retains the original labels for all arcs, regardless of whether they have been lifted or not, and the number of distinct labels is therefore simply the number n of distinct dependency types.[2] In the first encoding scheme, called **Head**, we use a new label $d{\uparrow}h$ for each lifted arc, where d is the dependency relation between the syntactic head and the dependent in the non-projective representation, and h is the dependency relation that the syntactic head has to its own head in the underlying structure. Using this encoding scheme, the arc from *je* to *Z* in Figure 2 would be assigned the label AuxP↑Sb (signifying an AuxP that has been lifted from a Sb). In the second scheme, **Head+Path**, we in addition modify the label of every arc along the lifting path from the syntactic to the linear head so that if the original label is p the new label is $p{\downarrow}$. Thus, the arc from *je* to *jedna* will be labeled $Sb{\downarrow}$ (to indicate that there is a syntactic head below it). In the third and final scheme, denoted **Path**, we keep the extra information on path labels but drop the information about the syntactic head of the lifted arc, using the label $d{\uparrow}$ instead of $d{\uparrow}h$ (AuxP↑ instead of AuxP↑Sb).

As can be seen from the last column in Table 1, both **Head** and **Head+Path** may theoretically lead to a quadratic increase in the number of distinct arc labels (**Head+Path** being worse than **Head** only by a constant factor), while the increase is only linear in the case of **Path**. On the other hand, we can expect **Head+Path** to be the most useful representation for reconstructing the underlying non-projective dependency graph. In approaching this problem, a variety of different methods are conceivable, including a more or less sophisticated use of machine learning. In the present study, we limit ourselves to an algorithmic approach, using a deterministic breadth-first search. The details of the transformation procedure are slightly different depending on the encoding schemes:

- **Head:** For every arc of the form $w_i \xrightarrow{d{\uparrow}h} w_n$, we search the graph top-down, left-to-right, breadth-first starting at the head node w_i. If we find an arc $w_l \xrightarrow{h} w_m$, called a *target arc*, we replace $w_i \xrightarrow{d{\uparrow}h} w_n$ by $w_m \xrightarrow{d} w_n$; otherwise we replace $w_i \xrightarrow{d{\uparrow}h} w_n$ by $w_i \xrightarrow{d} w_n$ (i.e. we let the linear head be the syntactic head).
- **Head+Path:** Same as **Head**, but the search only follows arcs of the form $w_j \xrightarrow{p{\downarrow}} w_k$ and a target arc must have the form $w_l \xrightarrow{h{\downarrow}} w_m$; if no target arc is found, **Head** is used as backoff.
- **Path:** Same as **Head+Path**, but a target arc must have the form $w_l \xrightarrow{p{\downarrow}} w_m$ and no outgoing arcs of the form $w_m \xrightarrow{p'{\downarrow}} w_o$; no backoff.

In section 4 we evaluate these transformations with respect to projectivized dependency treebanks, and in section 5 they are applied to parser output. Before

[2] Note that this is a baseline for the parsing experiment only (Experiment 2). For Experiment 1 it is meaningless as a baseline, since it would result in 0% accuracy.

Feature type		Top−1	Top	Next	Next+1	Next+2	Next+3
Word form		+	+	+	+		
Part-of-speech		+	+	+	+	+	+
Dep type of	head		+				
	leftmost dep		+	+			
	rightmost dep		+				

Table 2: Features used in predicting the next parser action

we turn to the evaluation, however, we need to introduce the data-driven dependency parser used in the latter experiments.

3 Memory-Based Dependency Parsing

In the experiments below, we employ a data-driven deterministic dependency parser producing labeled projective dependency graphs,[3] previously tested on Swedish (Nivre et al., 2004) and English (Nivre and Scholz, 2004). The parser builds dependency graphs by traversing the input from left to right, using a stack to store tokens that are not yet complete with respect to their dependents. At each point during the derivation, the parser has a choice between pushing the next input token onto the stack – with or without adding an arc from the token on top of the stack to the token pushed – and popping a token from the stack – with or without adding an arc from the next input token to the token popped. More details on the parsing algorithm can be found in Nivre (2003).

The choice between different actions is in general nondeterministic, and the parser relies on a memory-based classifier, trained on treebank data, to predict the next action based on features of the current parser configuration. Table 2 shows the features used in the current version of the parser. At each point during the derivation, the prediction is based on six word tokens, the two topmost tokens on the stack, and the next four input tokens. For each token, three types of features may be taken into account: the word form; the part-of-speech assigned by an automatic tagger; and labels on previously assigned dependency arcs involving the token – the arc from its head and the arcs to its leftmost and rightmost dependent, respectively. Except for the leftmost dependent of the next input token, dependency type features are limited to tokens on the stack.

The prediction based on these features is a k-nearest neighbor classification, using the IB1 algorithm and $k = 5$, the modified value difference metric (MVDM) and class voting with inverse distance weighting, as implemented in the TiMBL software package (Daelemans et al., 2003). More details on the memory-based prediction can be found in Nivre et al. (2004) and Nivre and Scholz (2004).

4 Experiment 1: Treebank Transformation

The first experiment uses data from two dependency treebanks. The Prague Dependency Treebank (PDT) consists of more than 1M words of newspaper text, annotated on three levels, the morphological, analytical and tectogrammatical levels (Hajič, 1998). Our experiments all concern the analytical annotation, and the first experiment is based only on the training part. The Danish Dependency Treebank (DDT) comprises about 100K words of text selected from the Danish PAROLE corpus, with annotation of primary and secondary dependencies (Kromann, 2003). The entire treebank is used in the experiment, but only primary dependencies are considered.[4] In all experiments, punctuation tokens are included in the data but omitted in evaluation scores.

In the first part of the experiment, dependency graphs from the treebanks were projectivized using the algorithm described in section 2. As shown in Table 3, the proportion of sentences containing some non-projective dependency ranges from about 15% in DDT to almost 25% in PDT. However, the overall percentage of non-projective arcs is less than 2% in PDT and less than 1% in DDT. The last four

[3] The graphs satisfy all the well-formedness conditions given in section 2 except (possibly) connectedness. For robustness reasons, the parser may output a set of dependency trees instead of a single tree.

[4] If secondary dependencies had been included, the dependency graphs would not have satisfied the well-formedness conditions formulated in section 2.

Data set	# Sentences	% NonP	# Tokens	% NonP	# Lifts in projectivization			
					1	2	3	>3
PDT training	73,088	23.15	1,255,333	1.81	93.79	5.60	0.51	0.11
DDT total	5,512	15.48	100,238	0.94	79.49	13.28	4.36	2.87

Table 3: Non-projective sentences and arcs in PDT and DDT (NonP = non-projective)

Data set	Head		H+P		Path	
PDT training (28 labels)	92.3	(230)	99.3	(314)	97.3	(84)
DDT total (54 labels)	92.3	(123)	99.8	(147)	98.3	(99)

Table 4: Percentage of non-projective arcs recovered correctly (number of labels in parentheses)

columns in Table 3 show the distribution of non-projective arcs with respect to the number of lifts required. It is worth noting that, although non-projective constructions are less frequent in DDT than in PDT, they seem to be more deeply nested, since only about 80% can be projectivized with a single lift, while almost 95% of the non-projective arcs in PDT only require a single lift.

In the second part of the experiment, we applied the inverse transformation based on breadth-first search under the three different encoding schemes. The results are given in Table 4. As expected, the most informative encoding, **Head+Path**, gives the highest accuracy with over 99% of all non-projective arcs being recovered correctly in both data sets. However, it can be noted that the results for the least informative encoding, **Path**, are almost comparable, while the third encoding, **Head**, gives substantially worse results for both data sets. We also see that the increase in the size of the label sets for **Head** and **Head+Path** is far below the theoretical upper bounds given in Table 1. The increase is generally higher for PDT than for DDT, which indicates a greater diversity in non-projective constructions.

5 Experiment 2: Memory-Based Parsing

The second experiment is limited to data from PDT.[5] The training part of the treebank was projectivized under different encoding schemes and used to train memory-based dependency parsers, which were run on the test part of the treebank, consisting of 7,507 sentences and 125,713 tokens.[6] The inverse transformation was applied to the output of the parsers and the result compared to the gold standard test set.

Table 5 shows the overall parsing accuracy attained with the three different encoding schemes, compared to the baseline (no special arc labels) and to training directly on non-projective dependency graphs. Evaluation metrics used are Attachment Score (AS), i.e. the proportion of tokens that are attached to the correct head, and Exact Match (EM), i.e. the proportion of sentences for which the dependency graph exactly matches the gold standard. In the labeled version of these metrics (L) both heads and arc labels must be correct, while the unlabeled version (U) only considers heads.

The first thing to note is that projectivizing helps in itself, even if no encoding is used, as seen from the fact that the projective baseline outperforms the non-projective training condition by more than half a percentage point on attachment score, although the gain is much smaller with respect to exact match. The second main result is that the pseudo-projective approach to parsing (using special arc labels to guide an inverse transformation) gives a further improvement of about one percentage point on attachment score. With respect to exact match, the improvement is even more noticeable, which shows quite clearly that even if non-projective dependencies are rare on the token level, they are nevertheless important for getting the global syntactic structure correct.

All improvements over the baseline are statistically significant beyond the 0.01 level (McNemar's

[5] Preliminary experiments using data from DDT indicated that the limited size of the treebank creates a severe sparse data problem with respect to non-projective constructions.

[6] The part-of-speech tagging used in both training and testing was the uncorrected output of an HMM tagger distributed with the treebank; cf. Hajič et al. (2001a).

Encoding	UAS	LAS	UEM	LEM
Non-projective	78.5	71.3	28.9	20.6
Baseline	79.1	72.0	29.2	20.7
Head	80.1	72.8	31.6	22.2
Head+Path	80.0	72.8	31.8	22.4
Path	80.0	72.7	31.6	22.0

Table 5: Parsing accuracy (AS = attachment score, EM = exact match; U = unlabeled, L = labeled)

	Unlabeled			Labeled		
Encoding	**P**	**R**	**F**	**P**	**R**	**F**
Head	61.3	54.1	57.5	55.2	49.8	52.4
Head+Path	63.9	54.9	59.0	57.9	50.6	54.0
Path	58.2	49.5	53.4	51.0	45.7	48.2

Table 6: Precision, recall and F-measure for non-projective arcs

test). By contrast, when we turn to a comparison of the three encoding schemes it is hard to find any significant differences, and the overall impression is that it makes little or no difference which encoding scheme is used, as long as there is some indication of which words are assigned their linear head instead of their syntactic head by the projective parser. This may seem surprising, given the experiments reported in section 4, but the explanation is probably that the non-projective dependencies that can be recovered at all are of the simple kind that only requires a single lift, where the encoding of path information is often redundant. It is likely that the more complex cases, where path information could make a difference, are beyond the reach of the parser in most cases.

However, if we consider precision, recall and F-measure on non-projective dependencies only, as shown in Table 6, some differences begin to emerge. The most informative scheme, **Head+Path**, gives the highest scores, although with respect to **Head** the difference is not statistically significant, while the least informative scheme, **Path** – with almost the same performance on treebank transformation – is significantly lower ($p < 0.01$). On the other hand, given that all schemes have similar parsing accuracy overall, this means that the **Path** scheme is the least likely to introduce errors on projective arcs.

The overall parsing accuracy obtained with the pseudo-projective approach is still lower than for the best projective parsers. Although the best published results for the Collins parser is 80% UAS (Collins, 1999), this parser reaches 82% when trained on the entire training data set, and an adapted version of Charniak's parser (Charniak, 2000) performs at 84% (Jan Hajič, pers. comm.). However, the accuracy is considerably higher than previously reported results for robust non-projective parsing of Czech, with a best performance of 73% UAS (Holan, 2004).

Compared to related work on the recovery of long-distance dependencies in constituency-based parsing, our approach is similar to that of Dienes and Dubey (2003) in that the processing of non-local dependencies is partly integrated in the parsing process, via an extension of the set of syntactic categories, whereas most other approaches rely on post-processing only. However, while Dienes and Dubey recognize empty categories in a pre-processing step and only let the parser find their antecedents, we use the parser both to detect dislocated dependents and to predict either the type or the location of their syntactic head (or both) and use post-processing only to transform the graph in accordance with the parser's analysis.

6 Conclusion

We have presented a new method for non-projective dependency parsing, based on a combination of data-driven projective dependency parsing and graph transformation techniques. The main result is that the combined system can recover non-projective dependencies with a precision sufficient to give a significant improvement in overall parsing accuracy,

especially with respect to the exact match criterion, leading to the best reported performance for robust non-projective parsing of Czech.

Acknowledgements

This work was supported in part by the Swedish Research Council (621-2002-4207). Memory-based classifiers for the experiments were created using TiMBL (Daelemans et al., 2003). Special thanks to Jan Hajič and Matthias Trautner Kromann for assistance with the Czech and Danish data, respectively, and to Jan Hajič, Tomáš Holan, Dan Zeman and three anonymous reviewers for valuable comments on a preliminary version of the paper.

References

Cahill, A., Burke, M., O'Donovan, R., Van Genabith, J. and Way, A. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of ACL*.

Campbell, R. 2004. Using linguistic principles to recover empty categories. In *Proceedings of ACL*.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*.

Collins, M., Hajič, J., Brill, E., Ramshaw, L. and Tillmann, C. 1999. A statistical parser for Czech. In *Proceedings of ACL*.

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

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

Daelemans, W., Zavrel, J., van der Sloot, K. and van den Bosch, A. 2003. TiMBL: Tilburg Memory Based Learner, version 5.0, Reference Guide. Technical Report ILK 03-10, Tilburg University, ILK.

Dienes, P. and Dubey, A. 2003. Deep syntactic processing by combining shallow methods. In *Proceedings of ACL*.

Duchier, D. and Debusmann, R. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of ACL*.

Eisner, J. M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING*.

Foth, K., Daum, M. and Menzel, W. 2004. A broad-coverage parser for German based on defeasible constraints. In *Proceedings of KONVENS*.

Hajič, J., Krbec, P., Oliva, K., Kveton, P. and Petkevic, V. 2001. Serial combination of rules and statistics: A case study in Czech tagging. In *Proceedings of ACL*.

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

Hajič, J. 1998. Building a syntactically annotated corpus: The Prague Dependency Treebank. In *Issues of Valency and Meaning*, pages 106–132. Karolinum.

Hellwig, P. 2003. Dependency unification grammar. In *Dependency and Valency*, pages 593–635. Walter de Gruyter.

Holan, T., Kuboň, V. and Plátek, M. 2001. Word-order relaxations and restrictions within a dependency grammar. In *Proceedings of IWPT*.

Holan, T. 2004. Tvorba zavislostniho syntaktickeho analyzatoru. In *Proceedings of MIS'2004*.

Jijkoun, V. and de Rijke, M. 2004. Enriching the output of a parser using memory-based learning. In *Proceedings of ACL*.

Johnson, M. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of ACL*.

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

Kromann, M. T. 2003. The Danish Dependency Treebank and the DTAG treebank tool. In *Proceedings of TLT 2003*.

Levy, R. and Manning, C. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of ACL*.

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

Nivre, J. and Scholz, M. 2004. Deterministic dependency parsing of English text. In *Proceedings of COLING*.

Nivre, J., Hall, J. and Nilsson, J. 2004. Memory-based dependency parsing. In *Proceedings of CoNLL*.

Nivre, J. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*.

Oflazer, K., Say, B., Hakkani-Tür, D. Z. and Tür, G. 2003. Building a Turkish treebank. In *Treebanks: Building and Using Parsed Corpora*, pages 261–277. Kluwer Academic Publishers.

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

Sleator, D. and Temperley, D. 1993. Parsing English with a link grammar. In *Proceedings of IWPT*.

Tapanainen, P. and Järvinen, T. 1997. A non-projective dependency parser. In *Proceedings of ANLP*.

Wang, W. and Harper, M. P. 2004. A statistical constraint dependency grammar (CDG) parser. In *Proceedings of the Workshop in Incremental Parsing (ACL)*.

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