Memory-Based Dependency Parsing

Joakim Nivre, Johan Hall and **Jens Nilsson**
School of Mathematics and Systems Engineering
Växjö University
SE-35195 Växjö
Sweden
firstname.lastname@msi.vxu.se

Abstract

This paper reports the results of experiments using memory-based learning to guide a deterministic dependency parser for unrestricted natural language text. Using data from a small treebank of Swedish, memory-based classifiers for predicting the next action of the parser are constructed. The accuracy of a classifier as such is evaluated on held-out data derived from the treebank, and its performance as a parser guide is evaluated by parsing the held-out portion of the treebank. The evaluation shows that memory-based learning gives a signficant improvement over a previous probabilistic model based on maximum conditional likelihood estimation and that the inclusion of lexical features improves the accuracy even further.

1 Introduction

Deterministic dependency parsing has recently been proposed as a robust and efficient method for syntactic parsing of unrestricted natural language text (Yamada and Matsumoto, 2003; Nivre, 2003). Dependency parsing means that the goal of the parsing process is to construct a dependency graph, of the kind depicted in Figure 1. Deterministic parsing means that we always derive a single analysis for each input string. Moreover, this single analysis is derived in a monotonic fashion with no redundancy or backtracking, which makes it possible to parse natural language sentences in linear time (Nivre, 2003).

In this paper, we report experiments using memory-based learning (Daelemans, 1999) to guide the parser described in Nivre (2003), using data from a small treebank of Swedish (Einarsson, 1976). Unlike most previous work on data-driven dependency parsing (Eisner, 1996; Collins et al., 1999; Yamada and Matsumoto, 2003; Nivre, 2003), we assume that dependency graphs are labeled with dependency types, although the evaluation will give results for both labeled and unlabeled representations.

The paper is structured as follows. Section 2 gives the necessary background definitions and introduces the idea of guided parsing as well as memory-based learning. Section 3 describes the data used in the experiments, the evaluation metrics, and the models and algorithms used in the learning process. Results from the experiments are given in section 4, while conclusions and suggestions for further research are presented in section 5.

2 Background

2.1 Dependency Graphs

The linguistic tradition of dependency grammar comprises a large and fairly diverse family of theories and formalisms that share certain basic assumptions about syntactic structure, in particular the assumption that syntactic structure consists of *lexical nodes* linked by binary relations called *dependencies* (see, e.g., Tesnière (1959), Sgall (1986), Mel'čuk (1988), Hudson (1990)). Thus, the common formal property of dependency structures, as compared to the representations based on constituency (or phrase structure), is the lack of nonterminal nodes.

In a dependency structure, every word token is dependent on at most one other word token, usually called its *head* or *regent*, which means that the structure can be represented as a directed graph, with nodes representing word tokens and arcs representing dependency relations. In addition, arcs may be labeled with specific dependency types. Figure 1 shows a labeled dependency graph for a simple Swedish sentence, where each word of the sentence is labeled with its part of speech and each arc labeled with a grammatical function.

Formally, we define dependency graphs in the following way:

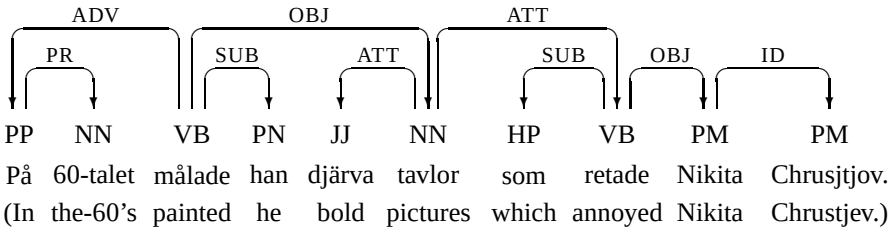

Figure 1: Dependency graph for Swedish sentence

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,

   (b) A is a set of labeled arcs (w_i, r, w_j) (where $w_i, w_j \in W$ and $r \in R$).

   We write $w_i < w_j$ to express that w_i precedes w_j in the string W (i.e., $i < j$); we write $w_i \xrightarrow{r} w_j$ to say that there is an arc from w_i to w_j labeled r, and $w_i \rightarrow w_j$ to say that there is an arc from w_i to w_j (regardless of the label); we use $\rightarrow^*$ to denote the reflexive and transitive closure of the unlabeled arc relation; and we use $\leftrightarrow$ and $\leftrightarrow^*$ for the corresponding undirected relations, i.e. $w_i \leftrightarrow w_j$ iff $w_i \rightarrow w_j$ or $w_j \rightarrow w_i$.

3. A dependency graph $D = (W, A)$ is well-formed iff the five conditions given in Figure 2 are satisfied.

For a more detailed discussion of dependency graphs and well-formedness conditions, the reader is referred to Nivre (2003).

2.2 Parsing Algorithm

The parsing algorithm presented in Nivre (2003) is in many ways similar to the basic shift-reduce algorithm for context-free grammars (Aho et al., 1986), although the parse actions are different given that no nonterminal symbols are used. Moreover, unlike the algorithm of Yamada and Matsumoto (2003), the algorithm considered here actually uses a blend of bottom-up and top-down processing, constructing left-dependencies bottom-up and right-dependencies top-down, in order to achieve incrementality. For a similar but nondeterministic approach to dependency parsing, see Obrebski (2003).

Parser configurations are represented by triples $\langle S, I, A \rangle$, where S is the stack (represented as a list), I is the list of (remaining) input tokens, and A is the (current) arc relation for the dependency graph. Given an input string W, the parser is initialized to $\langle \mathbf{nil}, W, \emptyset \rangle$ and terminates when it reaches a configuration $\langle S, \mathbf{nil}, A \rangle$ (for any list S and set of arcs A). The input string W is *accepted* if the dependency graph $D = (W, A)$ given at termination is well-formed; otherwise W is *rejected*. The behavior of the parser is defined by the transitions defined in Figure 3 (where w_i, w_j and w_k are arbitrary word tokens, and r and r' are arbitrary dependency relations):

1. The transition **Left-Arc** (**LA**) adds an arc $w_j \xrightarrow{r} w_i$ from the next input token w_j to the token w_i on top of the stack and reduces (pops) w_i from the stack.

2. The transition **Right-Arc** (**RA**) adds an arc $w_i \xrightarrow{r} w_j$ from the token w_i on top of the stack to the next input token w_j, and shifts (pushes) w_j onto the stack.

3. The transition **Reduce** (**RE**) reduces (pops) the token w_i on top of the stack.

4. The transition **Shift** (**SH**) shifts (pushes) the next input token w_i onto the stack.

The transitions **Left-Arc** and **Right-Arc** are subject to conditions that ensure that the graph conditions **Unique label** and **Single head** are satisfied. By contrast, the **Reduce** transition can only be applied if the token on top of the stack already has a head. For **Shift**, the only condition is that the input list is non-empty.

As it stands, this transition system is nondeterministic, since several transitions can often be applied to the same configuration. Thus, in order to get a deterministic parser, we need to introduce a mechanism for resolving transition conflicts. Regardless of which mechanism is used, the parser is guaranteed to terminate after at most $2n$ transitions, given an input string of length n (Nivre, 2003). This means that as long as transitions can be performed in constant time, the running time of the parser will be linear in the length of the input. Moreover, the parser is guaranteed to produce a dependency graph that is acyclic and projective (and satisfies the unique-label and single-head constraints). This means that the dependency graph given at termination is well-formed if and only if it is connected (Nivre, 2003).

Unique label	$(w_i \xrightarrow{r} w_j \ \wedge \ w_i \xrightarrow{r'} w_j) \ \Rightarrow \ r = r'$
Single head	$(w_i \rightarrow w_j \ \wedge \ w_k \rightarrow w_j) \ \Rightarrow \ w_i = w_k$
Acyclic	$\neg(w_i \rightarrow w_j \ \wedge \ w_j \rightarrow^* w_i)$
Connected	$w_i \leftrightarrow^* w_j$
Projective	$(w_i \leftrightarrow w_k \ \wedge \ w_i < w_j < w_k) \ \Rightarrow \ (w_i \rightarrow^* w_j \ \vee \ w_k \rightarrow^* w_j)$

Figure 2: Well-formedness conditions on dependency graphs

Initialization	$\langle \mathbf{nil}, W, \emptyset \rangle$	
Termination	$\langle S, \mathbf{nil}, A \rangle$	
Left-Arc	$\langle w_i \vert S, w_j \vert I, A \rangle \rightarrow \langle S, w_j \vert I, A \cup \{(w_j, r, w_i)\} \rangle$	$\neg \exists w_k \exists r'(w_k, r', w_i) \in A$
Right-Arc	$\langle w_i \vert S, w_j \vert I, A \rangle \rightarrow \langle w_j \vert w_i \vert S, I, A \cup \{(w_i, r, w_j)\} \rangle$	$\neg \exists w_k \exists r'(w_k, r', w_j) \in A$
Reduce	$\langle w_i \vert S, I, A \rangle \rightarrow \langle S, I, A \rangle$	$\exists w_j \exists r(w_j, r, w_i) \in A$
Shift	$\langle S, w_i \vert I, A \rangle \rightarrow \langle w_i \vert S, I, A \rangle$	

Figure 3: Parser transitions

2.3 Guided Parsing

One way of turning a nondeterministic parser into a deterministic one is to use a *guide* (or *oracle*) that can inform the parser at each nondeterministic choice point; cf. Kay (2000), Boullier (2003). Guided parsing is normally used to improve the efficiency of a nondeterministic parser, e.g. by letting a simpler (but more efficient) parser construct a first analysis that can be used to guide the choice of the more complex (but less efficient) parser. This is the approach taken, for example, in Boullier (2003).

In our case, we rather want to use the guide to improve the accuracy of a deterministic parser, starting from a baseline of randomized choice. One way of doing this is to use a treebank, i.e. a corpus of analyzed sentences, to train a classifier that can predict the next transition (and dependency type) given the current configuration of the parser. However, in order to maintain the efficiency of the parser, the classifier must also be implemented in such a way that each transition can still be performed in constant time.

Previous work in this area includes the use of memory-based learning to guide a standard shift-reduce parser (Veenstra and Daelemans, 2000) and the use of support vector machines to guide a deterministic dependency parser (Yamada and Matsumoto, 2003). In the experiments reported in this paper, we apply memory-based learning within a deterministic dependency parsing framework.

2.4 Memory-Based Learning

Memory-based learning and problem solving is based on two fundamental principles: learning is the simple storage of experiences in memory, and solving a new problem is achieved by reusing solutions from similar previously solved problems (Daelemans, 1999). It is inspired by the nearest neighbor approach in statistical pattern recognition and artificial intelligence (Fix and Hodges, 1952), as well as the analogical modeling approach in linguistics (Skousen, 1989; Skousen, 1992). In machine learning terms, it can be characterized as a lazy learning method, since it defers processing of input until needed and processes input by combining stored data (Aha, 1997).

Memory-based learning has been successfully applied to a number of problems in natural language processing, such as grapheme-to-phoneme conversion, part-of-speech tagging, prepositional-phrase attachment, and base noun phrase chunking (Daelemans et al., 2002). Most relevant in the present context is the use of memory-based learning to predict the actions of a shift-reduce parser, with promising results reported in Veenstra and

Daelemans (2000).

The main reason for using memory-based learning in the present context is the flexibility offered by similarity-based extrapolation when classifying previously unseen configurations, since previous experiments with a probabilistic model has shown that a fixed back-off sequence does not work well in this case (Nivre, 2004). Moreover, the memory-based approach can easily handle multi-class classification, unlike the support vector machines used by Yamada and Matsumoto (2003).

For the experiments reported in this paper, we have used the software package TiMBL (Tilburg Memory Based Learner), which provides a variety of metrics, algorithms, and extra functions on top of the classical k nearest neighbor classification kernel, such as value distance metrics and distance weighted class voting (Daelemans et al., 2003).

3 Method

3.1 Target Function and Approximation

The function we want to approximate is a mapping f from parser configurations to parser actions, where each action consists of a transition and (unless the transition is **Shift** or **Reduce**) a dependency type:

$$f : \mathit{Config} \rightarrow \{\mathbf{LA}, \mathbf{RA}, \mathbf{RE}, \mathbf{SH}\} \times (R \cup \{\mathbf{nil}\})$$

Here Config is the set of all possible parser configurations and R is the set of dependency types as before. However, in order to make the problem tractable, we try to learn a function $\hat{f}$ whose domain is a finite space of parser *states*, which are abstractions over configurations. For this purpose we define a number of features that can be used to define different models of parser state. The features used in this study are listed in Table 1.

The first five features (TOP–TOP.RIGHT) deal with properties of the token on top of the stack. In addition to the word form itself (TOP), we consider its part-of-speech (as assigned by an automatic part-of-speech tagger in a preprocessing phase), the dependency type by which it is related to its head (which may or may not be available in a given configuration depending on whether the head is to the left or to the right of the token in question), and the dependency types by which it is related to its leftmost and rightmost dependent, respectively (where the current rightmost dependent may or may not be the rightmost dependent in the complete dependency tree).

The following three features (NEXT–NEXT.LEFT) refer to properties of the next input token. In this case, there are no features corresponding to TOP.DEP and TOP.RIGHT, since the relevant dependencies can never be present at decision time. The final feature (LOOK) is a simple lookahead, using the part-of-speech of the next plus one input token.

In the experiments reported below, we have used two different parser state models, one called the *lexical* model, which includes all nine features, and one called the *non-lexical* model, where the two lexical features TOP and NEXT are omitted. For both these models, we have used memory-based learning with different parameter settings, as implemented TiMBL.

For comparison, we have included an earlier classifier that uses the same features as the non-lexical model, but where prediction is based on maximum conditional likelihood estimation. This classifier always predicts the most probable transition given the state and the most probable dependency type given the transition and the state, with conditional probabilities being estimated by the empirical distribution in the training data. Smoothing is performed only for zero frequency events, in which case the classifier backs off to more general models by omitting first the features TOP.LEFT and LOOK and then the features TOP.RIGHT and NEXT.LEFT; if even this does not help, the classifier predicts **Reduce** if permissible and **Shift** otherwise. This model, which we will refer to as the MCLE model, is described in more detail in Nivre (2004).

3.2 Data

It is standard practice in data-driven approaches to natural language parsing to use treebanks both for training and evaluation. Thus, the Penn Treebank of American English (Marcus et al., 1993) has been used to train and evaluate the best available parsers of unrestricted English text (Collins, 1999; Charniak, 2000). One problem when developing a parser for Swedish is that there is no comparable large-scale treebank available for Swedish.

For the experiments reported in this paper we have used a manually annotated corpus of written Swedish, created at Lund University in the 1970's and consisting mainly of informative texts from official sources (Einarsson, 1976). Although the original annotation scheme is an eclectic combination of constituent structure, dependency structure, and topological fields (Teleman, 1974), it has proven possible to convert the annotated sentences to dependency graphs with fairly high accuracy.

In the conversion process, we have reduced the original fine-grained classification of grammatical functions to a more restricted set of 16 dependency types, which are listed in Table 2. We have also replaced the original (manual) part-of-speech annotation by using the same automatic tagger that is used for preprocessing in the parser. This is a standard probabilistic tagger trained on the Stockholm-Umeå Corpus of written Swedish (SUC, 1997) and found to have an accuracy of 95–96% when tested on held-out data.

Since the function we want to learn is a mapping from parser states to transitions (and dependency types), the treebank data cannot be used directly as training and test

Feature	Description
TOP	The token on top of the stack
TOP.POS	The part-of-speech of TOP
TOP.DEP	The dependency type of TOP (if any)
TOP.LEFT	The dependency type of TOP's leftmost dependent (if any)
TOP.RIGHT	The dependency type of TOP's rightmost dependent (if any)
NEXT	The next input token
NEXT.POS	The part-of-speech of NEXT
NEXT.LEFT	The dependency type of NEXT's leftmost dependent (if any)
LOOK.POS	The part-of-speech of the next plus one input token

Table 1: Parser state features

data. Instead, we have to simulate the parser on the treebank in order to derive, for each sentence, the transition sequence corresponding to the correct dependency tree. Given the result of this simulation, we can construct a data set consisting of pairs $\langle s, t \rangle$, where s is a parser state and t is the correct transition from that state (including a dependency type if applicable). Unlike standard shift-reduce parsing, the simulation of the current algorithm is almost deterministic and is guaranteed to be correct if the input dependency tree is well-formed.

The complete converted treebank contains 6316 sentences and 97623 word tokens, which gives a mean sentence length of 15.5 words. The treebank has been divided into three non-overlapping data sets: 80% for training 10% for development/validation, and 10% for final testing (random samples). The results presented below are all from the validation set. (The final test set has not been used at all in the experiments reported in this paper.)

When talking about test and validation data, we make a distinction between the *sentence data*, which refers to the original annotated sentences in the treebank, and the *transition data*, which refers to the transitions derived by simulating the parser on these sentences. While the sentence data for validation consists of 631 sentences, the corresponding transition data contains 15913 instances. For training, only transition data is relevant and the training data set contains 371977 instances.

3.3 Evaluation

The output of the memory-based learner is a classifier that predicts the next transition (including dependency type), given the current state of the parser. The quality of this classifier has been evaluated with respect to both *prediction accuracy* and *parsing accuracy*.

Prediction accuracy refers to the quality of the classifier as such, i.e. how well it predicts the next transition given the correct parser state, and is measured by the classification accuracy on unseen transition data (using a 0-1 loss function). We use McNemar's test for statistical significance.

Parsing accuracy refers to the quality of the classifier as a guide for the deterministic parser and is measured by the accuracy obtained when parsing unseen sentence data. More precisely, parsing accuracy is measured by the *attachment score*, which is a standard measure used in studies of dependency parsing (Eisner, 1996; Collins et al., 1999). The attachment score is computed as the proportion of tokens (excluding punctuation) that are assigned the correct head (or no head if the token is a root). Since parsing is a sentence-level task, we believe that the overall attachment score should be computed as the mean attachment score *per sentence*, which gives an estimate of the expected attachment score for an arbitrary sentence. However, since most previous studies instead use the mean attachment score *per word* (Eisner, 1996; Collins et al., 1999), we will give this measure as well. In order to measure label accuracy, we also define a *labeled attachment score*, where both the head and the label must be correct, but which is otherwise computed in the same way as the ordinary (unlabeled) attachment score. For parsing accuracy, we use a paired t-test for statistical significance.

4 Results

Table 3 shows the prediction accuracy achieved with memory-based learning for the lexical and non-lexical model, with two different parameter settings for the learner. The results in the first column were obtained with the default settings of the TiMBL package, in particular:

- The IB1 classification algorithm (Aha et al., 1991).
- The overlap distance metric.
- Features weighted by Gain Ratio (Quinlan, 1993).
- $k = 1$, i.e. classification based on a single nearest neighbor.[1]

[1] In TiMBL, the value of k in fact refers to k nearest distances rather than k nearest neighbors, which means that, even with $k = 1$, the nearest neighbor set can contain several in-

Label	Dependency Type
ADV	Adverbial modifier
APP	Apposition
ATT	Attribute
CC	Coordination (conjunction or second conjunct)
DET	Determiner
ID	Non-first element of multi-word expression
IM	Infinitive dependent on infinitive marker
IP	Punctuation mark dependent on lexical head
INF	Infinitival complement
OBJ	Object
PR	Complement of preposition
PRD	Predicative complement
SUB	Subject
UK	Main verb of subordinate clause dependent on complementizer
VC	Verb chain (nonfinite verb dependent on other verb)
XX	Unclassifiable dependent

Table 2: Dependency types in Swedish treebank

Model	Default	Maximum
Non-lexical	86.8	87.4
Lexical	88.4	89.7

Table 3: Prediction accuracy for MBL models

The second column shows the accuracy for the best parameter settings found in the experiments (averaged over both models), which differ from the default in the following respects:

- Overlap metric replaced by the modified value distance metric (MVDM) (Stanfill and Waltz, 1986; Cost and Salzberg, 1993).
- No weighting of features.
- $k = 5$, i.e. classification based on 5 nearest neighbors.
- Distance weighted class voting with inverse distance weighting (Dudani, 1976).

For more information about the different parameters and settings, the reader is referred to Daelemans et al. (2003).

The results show that the lexical model performs consistently better than the non-lexical model, and that the difference increases with the optimization of the learning algorithm (all differences being significant at the .0001 level according to McNemar's test). This confirms previous results from statistical parsing indicating that lexical information is crucial for disambiguation (Collins, 1999; Charniak, 2000). As regards optimization, we may note that although there is a significant improvement for both models, the magnitude of the difference is relatively small.

Table 4 shows the parsing accuracy obtained with the optimized versions of the MBL models (lexical and non-lexical), compared to the MCLE model described in section 3. We see that MBL outperforms the MCLE model even when limited to the same features (all differences again being significant at the .0001 level according to a paired t-test). This can probably be explained by the fact that the similarity-based smoothing built into the memory-based approach gives a better extrapolation than the fixed back-off sequence in the MCLE model. We also see that the lexical MBL model outperforms both the other models. If we compare the labeled attachment score to the prediction accuracy (which also takes dependency types into account), we observe a substantial drop (from 89.7 to 81.7 for the lexical model, from 87.4 to 76.5 for the non-lexical model), which is of course only to be expected. The unlabeled attachment score is naturally higher, and it is worth noting that the relative difference between the MBL lexical model and the other two models is much smaller. This indicates that the advantage of the lexical model mainly concerns the accuracy in predicting dependency type in addition to transition.

stances that are equally distant to the test instance. This is different from the original IB1 algorithm, as described in Aha et al. (1991).

Model	Labeled	Unlabeled
MCLE	74.7 (72.3)	81.5 (79.7)
MBL non-lexical	76.5 (74.7)	82.9 (81.7)
MBL lexical	81.7 (80.6)	85.7 (84.7)

Table 4: Parsing accuracy for MCLE and MBL models, attachment score per sentence (per word in parentheses)

If we compare the results concerning parsing accuracy to those obtained for other languages (given that there are no comparable results available for Swedish), we note that the best unlabeled attachment score is lower than for English, where the best results are above 90% (attachment score per word) (Collins et al., 1999; Yamada and Matsumoto, 2003), but higher than for Czech (Collins et al., 1999). This is encouraging, given that the size of the training set in our experiments is fairly small, only about 10% of the standard training set for the Penn Treebank. One reason why our results nevertheless compare reasonably well with those obtained with the much larger training set is probably that the conversion to dependency trees is more accurate for the Swedish treebank, given the explicit annotatation of grammatical functions. Moreover, the fact that our parser uses labeled dependencies is probably also significant, since the possibility of using information from previously assigned (labeled) dependencies during parsing seems to have a positive effect on accuracy (Nivre, 2004).

Finally, it may be interesting to consider the accuracy for individual dependency types. Table 5 gives labeled precision, labeled recall and unlabeled attachment score for four of the most important types with the MBL lexical model. The results indicate that subjects have the highest accuracy, especially when labels are taken into account. Objects and predicative complements have comparable attachment accuracy, but are more often misclassified with respect to dependency type. For adverbial modifiers, finally, attachment accuracy is lower than for the other dependency types, which is largely due to the notorious PP-attachment problem.

5 Conclusion

In this paper we have shown that a combination of memory-based learning and deterministic dependency parsing can be used to construct a robust and efficient parser for unrestricted natural language text, achieving a parsing accuracy which is close to the state of the art even with relatively limited amounts of training data. Classifiers based on memory-based learning achieve higher parsing accuracy than previous probabilistic models, and the improvement increases if lexical information is added to the model.

Suggestions for further research includes the further exploration of alternative models and parameter settings, but also the combination of inductive and analytical learning to impose high-level linguistic constraints, and the development of new parsing methods (e.g. involving multiple passes over the data). In addition, it is important to evaluate the approach with respect to other languages and corpora in order to increase the comparability with other approaches.

Acknowledgements

The work presented in this paper was supported by a grant from the Swedish Research Council (621-2002-4207). The memory-based classifiers used in the experiments were constructed using the Tilburg Memory-Based Learner (TiMBL) (Daelemans et al., 2003). We are grateful to three anonymous reviewers for constructive comments on the preliminary version of the paper.

References

D. W. Aha, D. Kibler and M. Albert. 1991. Instance-based Learning Algorithms. *Machine Learning* 6, 37–66.

D. Aha. 1997. *Lazy Learning*. Dordrecht: Kluwer.

A. V. Aho, R. Sethi and J. D. Ullman. 1986. *Compilers: Principles Techniques, and Tools*. Addison Wesley.

P. Boullier. 2003. Guided Earley Parsing. In G. van Noord (ed.) *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, Nancy, France, pp. 43–54.

E. Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings NAACL-2000*.

M. Collins. 1999. Head-Driven Statistical Models for Natural Language Parsing. PhD Thesis, University of Pennsylvania.

M. Collins, J. Hajic, E. Brill, L. Ramshaw and C. Tillmann. 1999. A Statistical Parser of Czech. In *Proceedings of 37th ACL Conference*, University of Maryland, College Park, USA, pp. 505–512.

S. Cost and S. Salzberg. 1993. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning* 10, 57–78.

Dependency type	Precision	Recall	Attachment
SUB	84.3	82.7	89.2
OBJ	74.7	78.8	87.0
PRD	75.1	71.4	84.2
ADV	76.2	74.6	78.3

Table 5: Dependency type accuracy, MBL lexical model; labeled precision, labeled recall, unlabeled attachment score

W. Daelemans. 1999. Memory-Based Language Processing. Introduction to the Special Issue. *Journal of Experimental and Theoretical Artificial Intelligence* 11(3), 287–292.

W. Daelemans, A. van den Bosch, J. Zavrel. 2002. Forgetting Exceptions is Harmful in Language Learning. *Machine Learning* 34, 11–43.

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

S. A. Dudani. 1976. The Distance-Weighted K-nearest Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6, 325–327.

J. Einarsson. 1976. Talbankens skriftsprkskonkordans. Lund University.

J. M. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of COLING-96*, Copenhagen.

E. Fix and J. Hodges. 1952. Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties. Technical Report 21-49-004-11, USAF School of Aviation Medicine, Randolph Field, Texas.

R. A. Hudson. 1990. *English Word Grammar*. Blackwell.

M. Kay. 2000. Guides and Oracles for Linear-Time Parsing. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT 00)*, Trento, Italy, pp. 6–9.

M. P. Marcus, B. Santorini and M. A. Marcinkiewics. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19, 313–330.

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

J. Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In G. van Noord (ed.) *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, Nancy, France, pp. 149–160.

J. Nivre. 2004. Inductive Dependency Parsing. Technical Report, Växjö University.

T. Obrebski. 2003. Dependency Parsing Using Dependency Graph. In G. van Noord (ed.) *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, Nancy, France, pp. 217–218.

J. R. Quinlan. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

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

R. Skousen. 1989. *Analogical Modeling of Language*. Dordrecht: Kluwer.

R. Skousen. 1992. *Analogy and Structure*. Dordrecht: Kluwer.

C. Stanfill and D. Waltz. 1986. Toward Memory-Based Reasoning. *Communications of the ACM* 29(12), 1213–1228.

SUC 1997. Stockholm Umeå Corpus. Version 1.0. Produced by Department of Linguistics, Umeå University and Department of Linguistics, Stockholm University.

U. Teleman. 1974. *Manual för grammatisk beskrivning av talad och skriven svenska*. Studentlitteratur.

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

J. Veenstra and W. Daelemans. 2000. A Memory-Based Alternative for Connectionist Shift-Reduce Parsing. Technical Report ILK-0012, University of Tilburg.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In G. van Noord (ed.) *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, Nancy, France, pp. 195–206.