Inductive Dependency Parsing

Joakim Nivre

Växjö University
School of Mathematics and Systems Engineering
SE-35195 Växjö
Sweden
Joakim.Nivre@msi.vxu.se

Abstract. This paper describes experiments on using inductive machine learning to guide a deterinistic dependency parser for unrestricted natural language text. Using data from a small treebank of Swedish, an eager probabilistic learning algorithm is used to induce context-sensitive parse tables. Evaluation shows a significant improvement over the baseline, which uses a table without contextual information.

1 Introduction

Deterministic dependency parsing has recently been proposed as a robust and efficient method for syntactic parsing of unrestricted natural language text [1,2]. Dependency parsing means that the goal of the parsing process is to construct a dependency graph, of the kind depicted in Figure 1 (rather than, say, a context-free phrase structure tree). 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 [2].

In this paper, I will continue to investigate the parsing algorithm proposed in [2], which can be described as a left-to-right incremental algorithm, which blends bottom-up and top-down processing, as opposed to the similar algorithm described in [1], which proceeds strictly bottom-up. More precisely, I will explore the possibility of using inductive machine learning to guide the parser, using data from a small treebank of Swedish [3]. Unlike in previous work [1,2], I will 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 based on inductive machine 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.

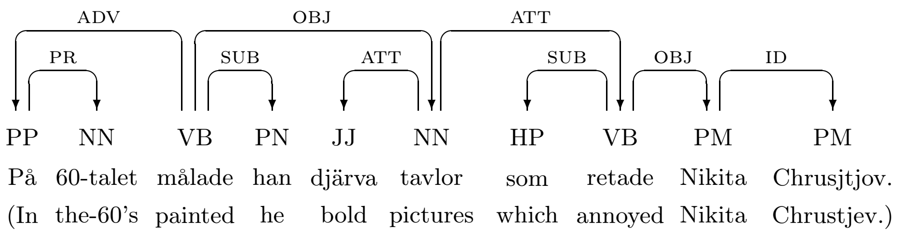
ADV
OBJ
ATT
PR
SUB
ATT
SUB
OBJ
ID
PP NN VB PN JJ NN HP VB PM PM
På 60-talet målade han djärva tavlor som retade Nikita Chrusjtjov.
(In the-60's painted he bold pictures which annoyed Nikita Chrustjev.)

Fig. 1. Dependency graph for Swedish sentence

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 [4], Sgall et al. [5], Mel'čuk [6], Hudson [7]). 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:

1. Let $R = \{r_1, \ldots, r_m\}$ be the set of permissible dependency types (arc labels).
2. A depencency 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) $(w_i, w_j \in W,\ r \in R)$.

   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 following conditions are satisfied:

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	$\neg(w_i \rightarrow w_j \wedge w_i \rightarrow^* w_j)$
Connected	$w_i \leftrightarrow^* w_j$
Projective	$(w_i \leftrightarrow w_k \wedge w_i < w_j < w_k) \Rightarrow (w_i \rightarrow^* w_j \vee w_k \rightarrow^* w_j)$

For a more detailed discussion of dependency graphs and well-formedness conditions, the reader is referred to [2].

2.2 Parsing Algorithm

The parsing algorithm presented in [2] is in many ways similar to the basic shift-reduce algorithm for context-free grammars (Aho et al. [8]), although the parse actions are different given that no nonterminal symbols are used. Moreover, unlike the algorithm presented in [1], 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 [9].

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 2 (where w_i and w_j are arbitrary word tokens):

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

Fig. 2. Parser transitions

1. The transition **Left-Arc** 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** 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** reduces (pops) the token w_i on top of the stack.
4. The transition **Shift** shifts (pushes) the next input token n 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 always satisfied. By contrast, the **Reduce** transition can only be applied if the token on top of the stack already has a head. For **Shift**, finally, 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 [2]. 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 [2].

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. [10,11]. 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 [11].

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 [12] and the use of support vector machines to guide a deterministic dependency parser [1]. In the experiments reported in this paper, the guide will take the form of a *parse table*, where the parser can simply look up the next transition given the current configuration. However, since the number of possible parser configurations is in principle infinite (given that there is no upper bound on sentence length), we also need to define suitable abstractions over configurations, which will be called *parser states*. Given a particular definition of a parser state, the learning problem can then be defined as that of inducing the appropriate parse table from a treebank.

3 Method

3.1 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 [13] has been used to train and evaluate the best available parsers of unrestricted English text [14,15]. 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 [3]. Although the original annotation scheme is an eclectic combination of constituent structure, dependency structure, and topological fields [16], it has proven possible to convert the annotated sentences to dependency graphs with very high accuracy. In the conversion process, we have also reduced the original fine-grained classification of grammatical functions to a more restricted set of 16 dependency types.

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

3.2 Evaluation

Parsing accuracy was measured by the *attachment score* used by Eisner [17] and Collins et al. [18], which is computed as the proportion of tokens in a sentence (excluding punctuation) that are assigned the correct head (or no head if the token is a root). The overall attachment score is then calculated as the mean attachment score over all sentences in the sample. 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. Finally, we will report labeled precision and recall and unlabeled attachment score for selected dependency types.

3.3 Models and Algorithms

In order to define a parse table, where the parser can look up the next transition given the current configuration, we need to define a space of parser states, which are abstractions over configurations. For this purpose we define a number of variables that can be used to define different models of parser state. The variables used in this study are listed in Table 1.

Variable	**Explanation**
TOP	The token on top of the stack
NEXT	The next input token
LOOK	The next plus one input token
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.LEFT	The dependency type of NEXT's leftmost dependent (if any)

Table 1. Parser state variables

The first three variables (TOP–LOOK) refer to word tokens in the input string. However, in the experiments reported here, the value of these variables is not the actual word form but the part-of-speech tag assigned to the token in a preprocessing phase. (The tagger used in the experiments is a standard HMM tagger trained on the Stockholm-Umeå Corpus [19] and found to have an accuracy of 95–96% when tested on held-out data from the same corpus.)

The four remaining variables (TOP.DEP–NEXT.LEFT) concern dependency types and can be treated either as binary boolean variables – being true if the relevant token has been assigned a head and dependency type and false otherwise – or as multi-valued variables – where the value is the specific dependency type if one exists and a **nil** value otherwise. I will use the subscript b when the former interpretation is intended (thus, TOP.DEP_b, TOP.LEFT_b, etc.).

By combining these variables in different ways, we can obtain different parser state models. The three models used in the experiments are defined in Table 2. In the baseline model, the parser state is basically determined by the token on top of the stack and the next input token. Besides being sensitive to whether the top already has a head or not (which is necessary to ensure that the transitions **Left-Arc** and **Reduce** are only performed when permissible), the model does not incorporate any contextual information at all. Model 1 introduces context-dependence by making the state sensitive to the occurrence of right dependents of TOP and left dependents of NEXT, as well as the token after NEXT (the lookahead token LOOK). Model 2 extends the context-sensitivity in two different ways. First, it also considers left dependents of TOP. Secondly, it considers not only whether there exists a dependent in a certain structural position but also looks at its specific dependency type.

The learning algorithm used in the experiments is based on a probabilistic model and selects the most probable transition for each state and the most probable dependency type given the transition and the state. Formally:

Model	**Variables**
Baseline:	$\langle \textsc{top}, \textsc{next}, \textsc{top.dep}_b \rangle$
Model 1:	$\langle \textsc{top}, \textsc{next}, \textsc{top.dep}_b, \textsc{top.right}_b, \textsc{next.left}_b, \textsc{look} \rangle$
Model 2:	$\langle \textsc{top}, \textsc{next}, \textsc{top.dep}, \textsc{top.right}, \textsc{next.left}, \textsc{top.left}, \textsc{look} \rangle$

Table 2. Parser state models

1. Let S be the set of all possible parser states according to a given model.
2. Let $T : S \to \{\textbf{Left-Arc}, \textbf{Right-Arc}, \textbf{Reduce}, \textbf{Shift}\} \times (R \cup \{\textbf{nil}\})$ be the parse table for this state space.
3. Then, for every parser state $s \in S$, it should hold that $T(s) = (t_{max}, r_{max})$, where

$$t_{max} = \arg\max_t P(t|s)$$
$$r_{max} = \begin{cases} \arg\max_r P(r|t_{max}, s) & \text{if } t_{max} \in \{\textbf{Left-Arc}, \textbf{Right-Arc}\} \\ \textbf{nil} & \text{otherwise} \end{cases}$$

The parse table T defines a mapping from each state s to the most probable transition from that state and, if the most probable transition is **Left-Arc** or **Right-Arc**, to the most probable dependency type given the state and the transition.

The learning problem then becomes the problem of estimating the probabilities $P(t|s)$ and $P(r|t,s)$ for every state s, transition t and dependency type r. The learning method used to solve this problem is the following:

1. The parser is simulated on the treebank in order to determine the correct transition sequence for each sentence.
2. The simulation gives rise to frequency counts for different combinations of states, transitions and dependency types.
3. Frequency counts are used to derive maximum-likelihood estimates of the relevant probabilities as follows:

$$\hat{P}(t|s) = \frac{C(t,s)}{C(s)}$$
$$\hat{P}(r|t,s) = \frac{C(r,t,s)}{C(t,s)}$$

4. For cases where frequency counts are below a certain threshold, smoothing is performed by backing of to a more coarse-grained parser state model. The following backoff sequences were used:

$$\text{Model 1: } \langle \textsc{top}, \textsc{next}, \textsc{top.dep}_b, \textsc{top.right}_b, \textsc{next.left}_b, \textsc{look} \rangle$$
$$\Downarrow$$
$$\langle \textsc{top}, \textsc{next}, \textsc{top.dep}_b, \textsc{top.right}_b, \textsc{next.left}_b, \rangle$$
$$\Downarrow$$
$$\langle \textsc{top}, \textsc{next}, \textsc{top.dep}_b \rangle$$

$$\text{Model 2: } \langle \textsc{top}, \textsc{next}, \textsc{top.dep}, \textsc{top.right}, \textsc{next.left}, \textsc{top.left}, \textsc{look} \rangle$$
$$\Downarrow$$
$$\langle \textsc{top}, \textsc{next}, \textsc{top.dep}, \textsc{top.right}, \textsc{next.left} \rangle$$
$$\Downarrow$$
$$\langle \textsc{top}, \textsc{next}, \textsc{top.dep} \rangle$$

If the frequency is too low even in the last backoff model (or in the baseline model), all the probability mass is assigned to **Reduce**, if this is permissible, and to **Shift** otherwise.

4 Results

Table 3 gives the results obtained with the three models defined in the previous section and with a frequency threshold of 1 (i.e. as soon as there is at least one occurrence of the relevant combination of state, transition and dependency type, no backoff is performed). Both Model 1 and Model 2 give a substantial and statistically significant improvement over the baseline, but there is no significant difference between the two models. This is somewhat surprising, given that parser states in Model 2 contain more information. However, it is possible that Model 2 suffers (more) from sparse data, given that it has a much larger number of free parameters and given that the training set is fairly small. Therefore, it may be possible to improve the performance of Model 2 using a more sophisticated smoothing method.

Model	**LAS**	**UAS**
Baseline	72.4	79.8
Model 1	77.0	84.1
Model 2	76.9	83.8

Table 3. Parsing accuracy for different models (frequency threshold = 1)

If we compare the results to those obtained for other languages (given that there are no comparable results available for Swedish), we note that the unlabeled attachment score is considerably lower than for English, where the best results are above 90% [18,1], but slightly better than for Czech [18]. However, it should be remembered that the size of the training set in our experiments is fairly small, only about 10% of the standard training set from the Penn Treebank.

Table 4 shows the effect of using different frequency thresholds for Model 1. We see that raising the threshold consistently degrades performance. Although the differences are too small to be statistically significant given the size of the test set, it seems likely that this is a reliable result. This can be taken to corroborate the observations made in research on memory-based language processing to the

Threshold	LAS	UAS
1	77.0	84.1
3	76.3	83.5
5	75.7	82.7
10	74.4	81.3

Table 4. Parsing accuracy with different frequency thresholds (Model 1)

effect that information derived from low-frequency events is important, despite the fact that it can be unreliable in statistical estimation; cf. [20].

Table 5 shows the precision and recall for a selection of dependency types for Model 1 with a frequency threshold of 1. For the major grammatical functions on the clause level, i.e. SUB(ject), OBJ(ect) and PR(e)D(icative), the unlabeled attachment score seems fairly reasonable, indicating that these head words often find their correct head, while labeling is much less reliable, especially for OBJ and PRD. It seems that even if attachment accuracy can be improved, it will nevertheless be difficult to assign labels deterministically in the same process. However, it should be possible to improve the labeled precision and recall using a separate post-processing phase. Finally, we observe that the attachment score is considerably lower for ADV(erbial) and ATT(ribute) (especially for the former), which to a large extent is due to the infamous PP-attachment problem.

Dependency	LP	LR	UAS
SUB	81.2	76.9	86.7
OBJ	67.0	67.5	81.7
PRD	71.6	62.4	81.0
ADV	66.6	65.3	69.7
ATT	61.7	74.4	76.0
DET	88.0	74.4	88.0

Table 5. Precision and recall for selected dependency types (Model 1)

5 Conclusion

In this paper I have shown that a combination of inductive machine 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, at least with respect to attachment accuracy.

Suggestions for further research includes the exploration of alternative guiding and learning algorithms, 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.

References

1. Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In van Noord, G., ed.: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03). (2003) 195–206
2. Nivre, J.: An efficient algorithm for projective dependency parsing. In van Noord, G., ed.: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03). (2003) 149–160
3. Einarsson, J.: Talbankens skriftspråkskonkordans. Lund University (1976)
4. Tesnière, L.: Éléments de syntaxe structurale. Editions Klincksieck (1959)
5. Sgall, P., Hajicova, E., Panevova, J.: The Meaning of the Sentence in Its Pragmatic Aspects. Reidel (1986)
6. Mel'cuk, I.: Dependency Syntax: Theory and Practice. State University of New York Press (1988)
7. Hudson, R.A.: English Word Grammar. Blackwell (1990)
8. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles Techniques, and Tools. Addison Wesley (1986)
9. Obrebski, T.: Dependency parsing using dependency graph. In van Noord, G., ed.: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03). (2003) 217–218
10. Kay, M.: Guides and oracles for linear-time parsing. In: Proceedings of the 6th International Workshop on Parsing Technologies (IWPT 03). (2000) 6–9
11. Boullier, P.: Guided earley parsing. In van Noord, G., ed.: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03). (2003) 43–54
12. Venstra, J., Daelemans, W.: A memory-based alternative for connectionist shift-reduce parsing. Technical Report ILK-0012, University of Tilburg (2000)
13. Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a large annotated corpus of English: The Penn treebank. Computational Linguistics **19** (1993) 313–330
14. Collins, M.: Head-Driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania (1999)
15. Charniak, E.: A maximum-entropy-inspired parser. In: Proceedings NAACL-2000. (2000)
16. Teleman, U.: Manual för grammatisk beskrivning av talad och skriven svenska. Studentlitteratur (1974)
17. Eisner, J.M.: An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, Institute for Research in Cognitive Science, University of Pennsylvania (1996)
18. Collins, M., Hajič, J., Brill, E., Ramshaw, L., Tillmann, C.: A Statistical Parser of Czech. In: Proceedings of 37th ACL Conference, University of Maryland, College Park, USA (1999) 505–512
19. Ejerhed, E., Källgren, G.: Stockholm Umeå Corpus. Version 1.0. Produced by Department of Linguistics, Umeå University and Department of Linguistics, Stockholm University. ISBN 91-7191-348-3. (1997)
20. Daelemans, W., van den Bosch, A., Zavrel, J.: Forgetting exceptions is harmful in language learning. Machine Learning **34** (2002) 11–43