# Finding non-local dependencies: beyond pattern matching

**Valentin Jijkoun**
Language and Inference Technology Group,
ILLC, University of Amsterdam
jijkoun@science.uva.nl

## Abstract

We describe an algorithm for recovering non-local dependencies in syntactic dependency structures. The pattern-matching approach proposed by Johnson (2002) for a similar task for phrase structure trees is extended with machine learning techniques. The algorithm is essentially a classifier that predicts a non-local dependency given a connected fragment of a dependency structure and a set of structural features for this fragment. Evaluating the algorithm on the Penn Treebank shows an improvement of both precision and recall, compared to the results presented in (Johnson, 2002).

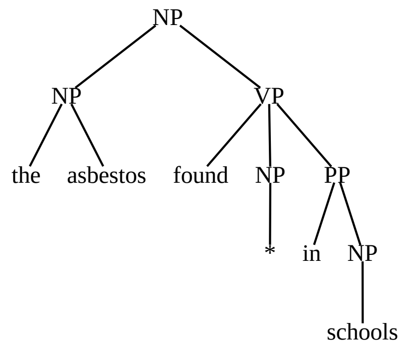
## 1 Introduction

Non-local dependencies (also called long-distance, long-range or unbounded) appear in many frequent linguistic phenomena, such as passive, WH-movement, control and raising etc. Although much current research in natural language parsing focuses on extracting local syntactic relations from text, non-local dependencies have recently started to attract more attention. In (Clark et al., 2002) long-range dependencies are included in parser's probabilistic model, while Johnson (2002) presents a method for recovering non-local dependencies after parsing has been performed.

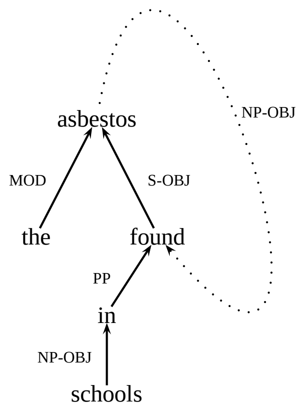
More specifically, Johnson (2002) describes a pattern-matching algorithm for inserting empty nodes and identifying their antecedents in phrase structure trees or, to put it differently, for recovering non-local dependencies. From a training corpus with annotated empty nodes Johnson's algorithm first extracts those local fragments of phrase trees which connect empty nodes with their antecedents, thus "licensing" corresponding non-local dependencies. Next, the extracted tree fragments are used as patterns to match against previously unseen phrase structure trees: when a pattern is matched, the algorithm introduces a corresponding non-local dependency, inserting an empty node and (possibly) co-indexing it with a suitable antecedent.

In (Johnson, 2002) the author notes that the biggest weakness of the algorithm seems to be that it fails to robustly distinguish co-indexed and free empty nodes and it is lexicalization that may be needed to solve this problem. Moreover, the author suggests that the algorithm may suffer from over-learning, and using more abstract "skeletal" patterns may be helpful to avoid this.

In an attempt to overcome these problems we developed a similar approach using dependency structures rather than phrase structure trees, which, moreover, extends bare pattern matching with machine learning techniques. A different definition of *pattern* allows us to significantly reduce the number of patterns extracted from the same corpus. Moreover, the patterns we obtain are quite general and in most cases directly correspond to specific linguistic phenomena. This helps us to understand what information about syntactic structure is important for the recovery of non-local dependencies and in which cases lexicalization (or even semantic analysis) is required. On the other hand, using these simplified patterns, we may loose some structural information important for recovery of non-local dependencies. To avoid this, we associate patterns with certain structural features and use statistical classifi-



(a) The Penn Treebank format



(b) Derived dependency structure

Figure 1: Past participle (reduced relative clause).

cation methods on top of pattern matching.

The evaluation of our algorithm on data automatically derived from the Penn Treebank shows an increase in both precision and recall in recovery of non-local dependencies by approximately 10% over the results reported in (Johnson, 2002). However, additional work remains to be done for our algorithm to perform well on the output of a parser.

## 2 From the Penn Treebank to a dependency treebank

This section describes the corpus of dependency structures that we used to evaluate our algorithm. The corpus was automatically derived from the Penn Treebank II corpus (Marcus et al., 1993), by means of the script `chunklink.pl` (Buchholz, 2002) that we modified to fit our purposes. The script uses a sort of head percolation table to identify heads of constituents, and then converts the result to a dependency format. We refer to (Buchholz, 2002) for a thorough description of the conversion algorithm, and will only emphasize the two most important modifications that we made.

One modification of the conversion algorithm concerns participles and reduced relative clauses modifying NPs. Regular participles in the Penn Treebank II are simply annotated as VPs adjoined to the modified NPs (see Figure 1(a)). These participles (also called *reduced relative clauses*, as they lack auxiliary verbs and complementizers) are both syntactically and semantically similar to full relative clauses, but the Penn annotation does not introduce empty complementizers, thus preventing co-indexing of a trace with any antecedent. We perform a simple heuristic modification while converting the Treebank to the dependency format: when we encounter an NP modified by a VP headed by a past participle, an object dependency is introduced between the head of the VP and the head of the NP. Figure 1(b) shows an example, with solid arrows denoting local and dotted arrows denoting non-local dependencies. Arrows are marked with dependency labels and go from dependents to heads.

This simple heuristics does not allow us to handle all reduced relative clauses, because some of them correspond to PPs or NPs rather than VPs, but the latter are quite rare in the Treebank.

The second important change to Buchholz' script concerns the structure of VPs. For every verb cluster, we choose the main verb as the head of the cluster, and leave modal and auxiliary verbs as dependents of the main verb. A similar modification was used by Eisner (1996) for the study of dependency parsing models. As will be described below, this allows us to "factor out" tense and modality of finite clauses from our patterns, making the patterns more general.

## 3 Pattern extraction and matching

After converting the Penn Treebank to a dependency treebank, we first extracted non-local dependency patterns. As in (Johnson, 2002), our patterns are minimal connected fragments containing both nodes involved in a non-local dependency. However, in our

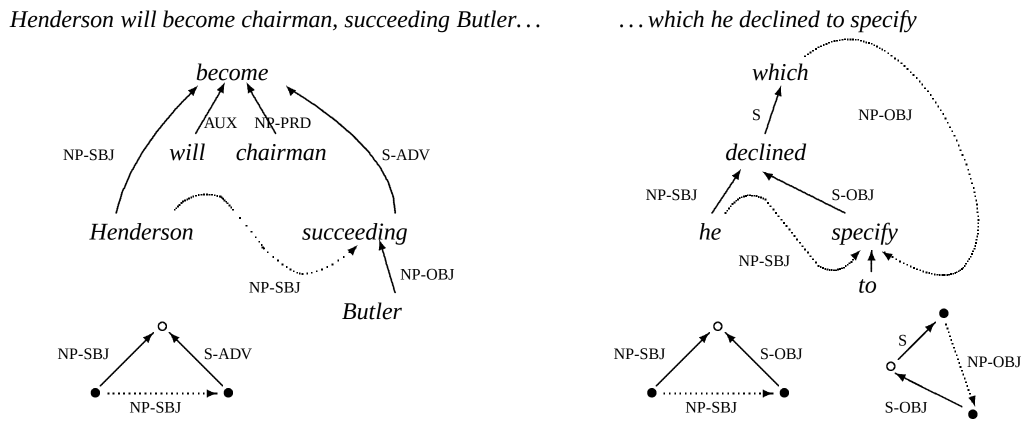


Figure 2: Dependency graphs and extracted patterns.

case these fragments are not connected sets of local trees, but shortest paths in local dependency graphs, leading from heads to non-local dependents. Patterns do not include POS tags of the involved words, but only labels of the dependencies. Thus, a *pattern* is a directed graph with labeled edges, and two distinguished nodes: the head and the dependent of a corresponding non-local dependency. When several patterns intersect, as may be the case, for example, when a word participates in more than one non-local dependency, these patterns are handled independently. Figure 2 shows examples of dependency graphs (above) and extracted patterns (below, with filled bullets corresponding to the nodes of a non-local dependency). As before, dotted lines denote non-local dependencies.

The definition of a structure *matching* a pattern, and the algorithms for pattern matching and pattern extraction from a corpus are straightforward and similar to those described in (Johnson, 2002).

The total number of non-local dependencies found in the Penn WSJ is 57325. The number of different extracted patterns is 987. The 80 most frequent patterns (those that we used for the evaluation of our algorithm) cover 53700 out of all 57325 non-local dependencies (93,7%). These patterns were further cleaned up manually, e.g., most Penn functional tags (-TMP, -CLR etc., but not -OBJ, -SBJ, -PRD) were removed. Thus, we ended up with 16 structural patterns (covering the same 93,7% of the Penn Treebank).

Table 1 shows some of the patterns found in the Penn Treebank. The column *Count* gives the number of times a pattern introduces non-local dependencies in the corpus. The *Match* column is the number of times a pattern actually occurs in the corpus (whether it introduces a non-local dependency or not). The patterns are shown as dependency graphs with labeled arrows from dependents to heads. The column *Dependency* shows labels and directions of introduced non-local dependencies.

Clearly, an occurrence of a pattern alone is not enough for inserting a non-local dependency and determining its label, as for many patterns *Match* is significantly greater than *Count*. For this reason we introduce a set of other structural features, associated with patterns. For every occurrence of a pattern and for every word of this occurrence, we extract the following features:

- *pos*, the POS tag of the word;
- *class*, the simplified word class (similar to (Eisner, 1996));
- *fin*, whether the word is a verb and a head of a finite verb cluster (as opposed to infinitives, gerunds or participles);
- *subj*, whether the word has a dependent (probably not included in the pattern) with a dependency label NP-SBJ; and
- *obj*, the same for NP-OBJ label.

Thus, an occurrence of a pattern is associated with a sequence of symbolic features: five features for each node in the pattern. E.g., a pattern consisting of 3 nodes will have a feature vector with 15 elements.

| Id | Count | Match | Pattern | Dependency | Dep. count | P | R | f |
|---|---|---|---|---|---|---|---|---|
| 1 | 10527 | 12716 | $\bullet \xrightarrow{\text{S}} \bullet$ | $\bullet \xleftarrow{\text{NP-SBJ}} \bullet$ | **7481** | 1.00 | 1.00 | **1.00** |
| 2 | | | | $\bullet \xleftarrow{\text{ADVP}} \bullet$ | **1945** | 0.82 | 0.90 | **0.86** |
| 3 | | | | $\bullet \xleftarrow{\text{NP-OBJ}} \bullet$ | 727 | 0.60 | 0.71 | 0.65 |
| 4 | 8789 | 17911 | $\bullet \xrightarrow{\text{S-*}} \circ \xleftarrow{\text{NP-SBJ}} \bullet$ | $\bullet \xleftarrow{\text{NP-SBJ}} \bullet$ | **8562** | 0.84 | 0.95 | **0.89** |
| 5 | | | | $\bullet \xleftarrow{\text{NP-OBJ}} \bullet$ | 227 | 0.83 | 0.71 | 0.77 |
| 6 | 8120 | 8446 | $\bullet \xrightarrow{\text{VP-OBJ}} \circ \xleftarrow{\text{NP-SBJ}} \bullet$ | $\bullet \xleftarrow{\text{NP-OBJ}} \bullet$ | **8120** | 0.99 | 1.00 | **1.00** |
| 7 | 2518 | 34808 | $\bullet \xrightarrow{\text{SBAR}} \bullet$ | $\bullet \xleftarrow{\text{NP-OBJ}} \bullet$ | 1013 | 0.73 | 0.84 | 0.79 |
| 8 | | | | $\bullet \xleftarrow{\text{NP-SBJ}} \bullet$ | 836 | 0.60 | 0.96 | 0.74 |
| 9 | | | | $\bullet \xleftarrow{\text{ADVP}} \bullet$ | 669 | 0.56 | 0.16 | 0.25 |
| 10 | 1424 | 1442 | $\bullet \xrightarrow{\text{S-OBJ}} \circ \xrightarrow{\text{VP-OBJ}} \circ \xleftarrow{\text{NP-SBJ}} \bullet$ | $\bullet \xleftarrow{\text{NP-SBJ}} \bullet$ | **1424** | 0.99 | 1.00 | **0.99** |
| 11 | 1265 | 28170 | $\bullet \xrightarrow{\text{S*}} \circ \xrightarrow{\text{S*}} \bullet$ | $\bullet \xleftarrow{\text{NP-SBJ}} \bullet$ | **1047** | 0.86 | 0.83 | **0.85** |
| 12 | | | | $\bullet \xleftarrow{\text{NP-OBJ}} \bullet$ | 218 | 0.77 | 0.71 | 0.74 |
| 13 | 880 | 1699 | $\bullet \xrightarrow{\text{S-NOM}} \circ \xrightarrow{\text{PP}} \circ \xleftarrow{\text{NP-SBJ}} \bullet$ | $\bullet \xleftarrow{\text{NP-SBJ}} \bullet$ | **880** | 0.85 | 0.87 | **0.86** |

Table 1: Several non-local dependency patterns, frequencies of patterns and pattern-dependency pairs in Penn Treebank, and evaluation results. The best scores are in boldface.

| Id | Dependency | Example |
|---|---|---|
| 1 | NP-SBJ | …sympthoms **that**$_{\text{dep}}$ **show**$_{\text{head}}$ up decades later… |
| 2 | ADVP | …buying futures **when**$_{\text{dep}}$ future prices **fall**$_{\text{head}}$… |
| 3 | NP-OBJ | …practices **that**$_{\text{dep}}$ the government has **identified**$_{\text{head}}$… |
| 4 | NP-SBJ | …the **airline**$_{\text{dep}}$ had been *planning* to **initiate**$_{\text{head}}$ service… |
| 5 | NP-OBJ | …that its **absence**$_{\text{dep}}$ *is* to **blame**$_{\text{head}}$ for the sluggish development… |
| 6 | NP-OBJ | …the **situation**$_{\text{dep}}$ will *get* **settled**$_{\text{head}}$ in the short term… |
| 7 | NP-OBJ | …the **number**$_{\text{dep}}$ of planes the company has **sold**$_{\text{head}}$… |
| 8 | NP-SBJ | …one of the first **countries**$_{\text{dep}}$ to **conclude**$_{\text{head}}$ its talks… |
| 9 | ADVP | …buying sufficient **options**$_{\text{dep}}$ to **purchase**$_{\text{head}}$ shares… |
| 10 | NP-SBJ | …both **magazines**$_{\text{dep}}$ *are expected* to **announce**$_{\text{head}}$ their ad rates… |
| 11 | NP-SBJ | …**which**$_{\text{dep}}$ is *looking* to **expand**$_{\text{head}}$ its business… |
| 12 | NP-OBJ | …the **programs**$_{\text{dep}}$ we *wanted* to **do**$_{\text{head}}$… |
| 13 | NP-SBJ | …**you**$_{\text{dep}}$ can't *make* soap *without* **turning**$_{\text{head}}$ up the flame… |

Table 2: Examples of the patterns. The "support" words, i.e. words that appear in a pattern but are neither heads nor non-local dependents, are in italic; they correspond to empty bullets in patterns in Table 1. Boldfaced words correspond to filled bullets in Table 1.

## 4 Classification of pattern instances

Given a pattern instance and its feature vector, our task now is to determine whether the pattern introduces a non-local dependency and, if so, what the label of this dependency is. In many cases this is not a binary decision, since one pattern may introduce several possible labeled dependencies (e.g., the pattern $\bullet \xrightarrow{S} \bullet$ in Table 1). Our task is a classification task: an instance of a pattern must be assigned to two or more classes, corresponding to several possible dependency labels (or absence of a dependency). We train a classifier on instances extracted from a corpus, and then apply it to previously unseen instances.

The procedure for finding non-local dependencies now consists of the two steps:

1. given a local dependency structure, find matching patterns and their feature vectors;
2. for each pattern instance found, use the classifier to identify a possible non-local dependency.

## 5 Experiments and evaluation

In our experiments we used sections 02-22 of the Penn Treebank as the training corpus and section 23 as the test corpus. First, we extracted all non-local patterns from the Penn Treebank, which resulted in 987 different (pattern, non-local dependency) pairs. As described in Section 3, after cleaning up we took 16 of the most common patterns.

For each of these 16 patterns, instances of the pattern, pattern features, and a non-local dependency label (or the special label "*no*" if no dependency was introduced by the instance) were extracted from the training and test corpora.

We performed experiments with two statistical classifiers: the decision tree induction system C4.5 (Quinlan, 1993) and the Tilburg Memory-Based Learner (TiMBL) (Daelemans et al., 2002). In most cases TiBML performed slightly better. The results described in this section were obtained using TiMBL.

For each of the 16 structural patterns, a separate classifier was trained on the set of (feature-vector, label) pairs extracted from the training corpus, and then evaluated on the pairs from the test corpus. Table 1 shows the results for some of the most frequent patterns, using conventional metrics: precision (the fraction of the correctly labeled dependencies among all the dependencies found), recall (the fraction of the correctly found dependencies among all the dependencies with a given label) and f-score (harmonic mean of precision and recall). The table also shows the number of times a pattern (together with a specific non-local dependency label) actually occurs in the whole Penn Treebank corpus (the column *Dependency count*).

In order to compare our results to the results presented in (Johnson, 2002), we measured the overall performance of the algorithm across patterns and non-local dependency labels. This corresponds to the row "*Overall*" of Table 4 in (Johnson, 2002), repeated here in Table 4. We also evaluated the procedure on NP traces across all patterns, i.e., on non-local dependencies with NP-SBJ, NP-OBJ or NP-PRD labels. This corresponds to rows 2, 3 and 4 of Table 4 in (Johnson, 2002). Our results are presented in Table 3. The first three columns show the results for those non-local dependencies that are actually covered by our 16 patterns (i.e., for 93.7% of all non-local dependencies). The last three columns present the evaluation with respect to *all* non-local dependencies, thus the precision is the same, but recall drops accordingly. These last columns give the results that can be compared to Johnson's results for section 23 (Table 4).

| | On covered deps | | | On all deps | | |
|---|---|---|---|---|---|---|
| | P | R | f | P | R | f |
| All | 0.89 | 0.93 | 0.91 | 0.89 | 0.84 | 0.86 |
| NPs | 0.90 | 0.96 | 0.93 | 0.90 | 0.87 | 0.89 |

Table 3: Overall performance of our algorithm.

| | On section 23 | | | On parser output | | |
|---|---|---|---|---|---|---|
| | P | R | f | P | R | f |
| Overall | 0.80 | 0.70 | 0.75 | 0.73 | 0.63 | 0.68 |

Table 4: Results from (Johnson, 2002).

It is difficult to make a strict comparison of our results and those in (Johnson, 2002). The two algorithms are designed for slightly different purposes: while Johnson's approach allows one to recover free empty nodes (without antecedents), we look for non-local dependencies, which corresponds to identification of co-indexed empty nodes (note, however, the modifications we describe in Section 2, when we actually transform free empty nodes into co-indexed empty nodes).

## 6 Discussion

The results presented in the previous section show that it is possible to improve over the simple pattern matching algorithm of (Johnson, 2002), using dependency rather than phrase structure information, more skeletal patterns, as was suggested by Johnson, and a set of features associated with instances of patterns.

One of the reasons for this improvement is that our approach allows us to discriminate between different syntactic phenomena involving non-local dependencies. In most cases our patterns correspond to linguistic phenomena. That helps to understand why a particular construction is easy or difficult for our approach, and in many cases to make the necessary modifications to the algorithm (e.g., adding other features to instances of patterns). For example, for patterns 11 and 12 (see Tables 1 and 2) our classifier distinguishes subject and object reasonably well, apparently, because the feature *has a local object* is explicitly present for all instances (for the examples 11 and 12 in Table 2, *expand* has a local object, but *do* doesn't).

Another reason is that the patterns are general enough to factor out minor syntactic differences in linguistic phenomena (e.g., see example 4 in Table 2). Indeed, the most frequent 16 patterns cover 93.7% of all non-local dependencies in the corpus. This is mainly due to our choices in the dependency representation, such as making the main verb a head of a verb phrase. During the conversion to a dependency treebank and extraction of patterns some important information may have been lost (e.g., the finiteness of a verb cluster, or presence of subject and object); for that reason we had to associate patterns with additional features, encoding this information and providing it to the classifier. In other words, we first take an "oversimplified" representation of the data, and then try to find what other data features can be useful. This strategy appears to be successful, because it allows us to identify which information is important for the recovery of non-local dependencies.

More generally, the reasonable overall performance of the algorithm is due to the fact that for the most common non-local dependencies (extraction in relative clauses and reduced relative clauses, passivization, control and raising) the structural information we extract is enough to robustly identify non-local dependencies in a local dependency graph: the most frequent patterns in Table 1 are also those with best scores. However, many less frequent phenomena appear to be much harder. For example, performance for relative clauses with extracted objects or adverbs is much worse than for subject relative clauses (e.g., patterns 2 and 3 vs. 1 in Table 1). Apparently, in most cases this is not due to the lack of training data, but because structural information alone is not enough and lexical preferences, subcategorization information, or even semantic properties should be considered. We think that the aproach allows us to identify those "hard" cases.

The natural next step in evaluating our algorithm is to work with the output of a parser instead of the original local structures from the Penn Treebank. Obviously, because of parsing errors the performance drops significantly: e.g., in the experiments reported in (Johnson, 2002) the overall f-score decreases from 0.75 to 0.68 when evaluating on parser output (see Table 4). While experimenting with Collins' parser (Collins, 1999), we found that for our algorithm the accuracy drops even more dramatically, when we train the classifier on Penn Treebank data and test it on parser output. One of the reasons is that, since we run our algorithm not on the parser's output itself but on the output automatically converted to dependency structures, conversion errors also contribute to the performance drop. Moreover, the conversion script is highly tailored to the Penn Treebank annotation (with functional tags and empty nodes) and, when run on the parser's output, produces structures with somewhat different dependency labels. Since our algorithm is sensitive to the exact labels of the dependencies, it suffers from these systematic errors.

One possible solution to that problem could be to extract patterns and train the classification algorithm not on the training part of the Penn Treebank, but on the parser output for it. This would allow us to train and test our algorithm on data of the same nature.

## 7 Conclusions and future work

We have presented an algorithm for recovering long-distance dependencies in local dependency structures. We extend the pattern matching approach of Johnson (2002) with machine learning techniques, and use dependency structures instead of constituency trees. Evaluation on the Penn Treebank shows an increase in accuracy.

However, we do not have yet satisfactory results when working on a parser output. The conversion algorithm and the dependency labels we use are largely based on the Penn Treebank annotation, and it seems difficult to use them with the output of a parser.

A parsing accuracy evaluation scheme based on grammatical relations (GR), presented in (Briscoe et al., 2002), provides a set of dependency labels (grammatical relations) and a manually annotated dependency corpus. Non-local dependencies are also annotated there, although no explicit difference is made between local and non-local dependencies. Since our classification algorithm does not depend on a particular set of dependency labels, we can also use the set of labels described by Briscoe et al, if we convert Penn Treebank to a GR-based dependency treebank and use it as the training corpus. This will allow us to make the patterns independent of the Penn Treebank annotation details and simplify testing the algorithm with a parser'u output. We will also be able to use the flexible and parameterizable scoring schemes discussed in (Briscoe et al., 2002).

We also plan to develop the approach by using iteration of our non-local relations extraction algorithm, i.e., by running the algorithm, inserting the found non-local dependencies, running it again etc., until no new dependencies are found. While raising an important and interesting issue of *the order* in which we examine our patterns, we believe that this will allow us to handle very long extraction chains, like the one in sentence *"Aichi revised its tax calculations after being challenged for allegedly failing to report…"*, where *Aichi* is a (non-local) dependent of five verbs. Iteration of the algorithm will also help to increase the coverage (which is 93,7% with our 16 non-iterated patterns).

## Acknowledgements

This research was supported by the Netherlands Organization for Scientific Research (NWO), under project number 220-80-001. We would like to thank Maarten de Rijke, Detlef Prescher and Khalil Sima'an for many fruitful discussions and useful suggestions and comments.

## References

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Meeting of the ACL*.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

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

Sabine Buchholz. 2002. *Memory-based grammatical relation finding*. Ph.D. thesis, Tilburg University.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2002. TiMBL: Tilburg Memory Based Learner, version 4.3, Reference Guide. ILK Technical Report 02-10, Available from `http://ilk.kub.nl/downloads/pub/papers/ilk0210.ps.gz`

J. Ross Quinlan. 1993. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers.

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

Ted Briscoe, John Carroll, Jonathan Graham and Ann Copestake. 2002. Relational evaluation schemes. In *Proceedings of the Beyond PARSEVAL Workshop at LREC 2002*, pages 4–8.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of the 40th Meeting of the ACL*, pages 327-334.