Enriching the Output of a Parser Using Memory-Based Learning

Valentin Jijkoun and **Maarten de Rijke**
Informatics Institute, University of Amsterdam
{jijkoun, mdr}@science.uva.nl

Abstract

We describe a method for enriching the output of a parser with information available in a corpus. The method is based on graph rewriting using memory-based learning, applied to dependency structures. This general framework allows us to accurately recover both grammatical and semantic information as well as non-local dependencies. It also facilitates dependency-based evaluation of phrase structure parsers. Our method is largely independent of the choice of parser and corpus, and shows state of the art performance.

1 Introduction

We describe a method to automatically enrich the output of parsers with information that is present in existing treebanks but usually not produced by the parsers themselves. Our motivation is two-fold. First and most important, for applications requiring information extraction or semantic interpretation of text, it is desirable to have parsers produce grammatically and semantically rich output. Second, to facilitate dependency-based comparison and evaluation of different parsers, their outputs may need to be transformed into specific rich dependency formalisms.

The method allows us to automatically transform the output of a parser into structures as they are annotated in a dependency treebank. For a phrase structure parser, we first convert the produced phrase structures into dependency graphs in a straightforward way, and then apply a sequence of graph transformations: changing dependency labels, adding new nodes, and adding new dependencies. A memory-based learner trained on a dependency corpus is used to detect which modifications should be performed. For a dependency corpus derived from the Penn Treebank and the parsers we considered, these transformations correspond to adding Penn functional tags (e.g., -SBJ, -TMP, -LOC), empty nodes (e.g., NP PRO) and non-local dependencies (controlled traces, WH-extraction, etc.). For these specific sub-tasks our method achieves state of the art performance. The evaluation of the transformed output of the parsers of Charniak (2000) and Collins (1999) gives 90% unlabelled and 84% labelled accuracy with respect to dependencies, when measured against a dependency corpus derived from the Penn Treebank.

The paper is organized as follows. After providing some background and motivation in Section 2, we give the general overview of our method in Section 3. In Sections 4 through 8, we describe all stages of the transformation process, providing evaluation results and comparing our methods to earlier work. We discuss the results in Section 9.

2 Background and Motivation

State of the art statistical parsers, e.g., parsers trained on the Penn Treebank, produce syntactic parse trees with bare phrase labels, such as NP, PP, S, although the training corpora are usually much richer and often contain additional grammatical and semantic information (distinguishing various modifiers, complements, subjects, objects, etc.), including non-local dependencies, i.e., relations between phrases not adjacent in the parse tree. While this information may be explicitly annotated in a treebank, it is rarely used or delivered by parsers.[1] The reason is that bringing in more information of this type usually makes the underlying parsing model more complicated: more parameters need to be estimated and independence assumptions may no longer hold. Klein and Manning (2003), for example, mention that using functional tags of the Penn Treebank (temporal, location, subject, predicate, etc.) with a simple unlexicalized PCFG generally had a negative effect on the parser's performance. Currently, there are no parsers trained on the Penn Treebank that use the structure of the treebank in full and that are thus

[1] Some notable exceptions are the CCG parser described in (Hockenmaier, 2003), which incorporates non-local dependencies into the parser's statistical model, and the parser of Collins (1999), which uses WH traces and argument/modifier distinctions.

capable of producing syntactic structures containing all or nearly all of the information annotated in the corpus.

In recent years there has been a growing interest in getting more information from parsers than just bare phrase trees. Blaheta and Charniak (2000) presented the first method for assigning Penn functional tags to constituents identified by a parser. Pattern-matching approaches were used in (Johnson, 2002) and (Jijkoun, 2003) to recover non-local dependencies in phrase trees. Furthermore, experiments described in (Dienes and Dubey, 2003) show that the latter task can be successfully addressed by shallow preprocessing methods.

3 An Overview of the Method

In this section we give a high-level overview of our method for transforming a parser's output and describe the different steps of the process. In the experiments we used the parsers described in (Charniak, 2000) and (Collins, 1999). For Collins' parser the text was first POS-tagged using Ratnaparkhi's maximum enthropy tagger.

The training phase of the method consists in learning which transformations need to be applied to the output of a parser to make it as similar to the treebank data as possible.

As a preliminary step (**Step 0**), we convert the WSJ[2] to a dependency corpus without losing the annotated information (functional tags, empty nodes, non-local dependencies). The same conversion is applied to the output of the parsers we consider. The details of the conversion process are described in Section 4 below.

The training then proceeds by comparing graphs derived from a parser's output with the graphs from the dependency corpus, detecting various mismatches, such as incorrect arc labels and missing nodes or arcs. Then the following steps are taken to fix the mismatches:

Step 1: changing arc labels

Step 2: adding new nodes

Step 3: adding new arcs

Obviously, other modifications are possible, such as deleting arcs or moving arcs from one node to another. We leave these for future work, though, and focus on the three transformations mentioned above.

The dependency corpus was split into training (WSJ sections 02–21), development (sections 00–01) and test (section 23) corpora. For each of the steps 1, 2 and 3 we proceed as follows:

1. compare the training corpus to the output of the parser on the strings of the corpus, after applying the transformations of the previous steps
2. identify possible beneficial transformations (which arc labels need to be changed or where new nodes or arcs need to be added)
3. train a memory-based classifier to predict possible transformations given their context (i.e., information about the local structure of the dependency graph around possible application sites).

While the definitions of the context and application site and the graph modifications are different for the three steps, the general structure of the method remains the same at each stage. Sections 6, 7 and 8 describe the steps in detail.

In the application phase of the method, we proceed similarly. First, the output of the parser is converted to dependency graphs, and then the learners trained during the steps 1, 2 and 3 are applied in sequence to perform the graph transformations.

Apart from the conversion from phrase structures to dependency graphs and the extraction of some linguistic features for the learning, our method does not use any information about the details of the treebank annotation or the parser's output: it works with arbitrary labelled directed graphs.

4 Step 0: From Constituents to Dependencies

To convert phrase trees to dependency structures, we followed the commonly used scheme (Collins, 1999). The conversion routine,[3] described below, is applied both to the original WSJ structures and the output of the parsers, though the former provides more information (e.g., traces) which is used by the conversion routine if available.

First, for the treebank data, all traces are resolved and corresponding empty nodes are replaced with links to target constituents, so that syntactic trees become directed acyclic graphs. Second, for each constituent we detect its head daughters (more than one in the case of conjunction) and identify lexical heads. Then, for each constituent we output new dependencies between its lexical head and the lexical heads of its non-head daughters. The label of every new dependency is the constituent's phrase

[2] Thoughout the paper WSJ refers to the Penn Treebank II Wall Street Journal corpus.

[3] Our converter is available at `http://www.science.uva.nl/~jijkoun/software`.

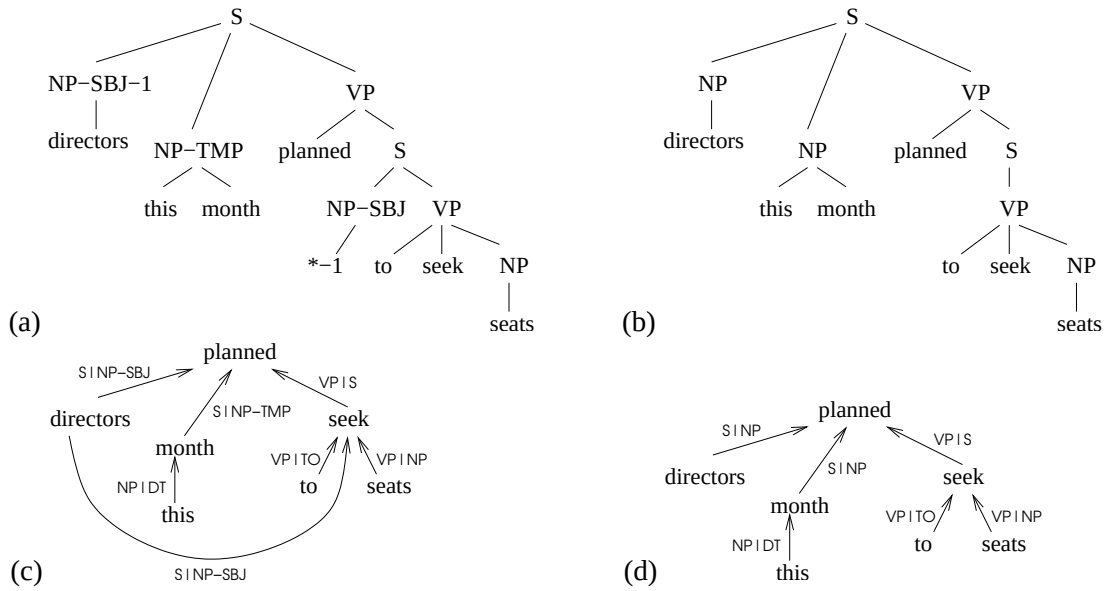

Figure 1: Example of (a) the Penn Treebank WSJ annotation, (b) the output of Charniak's parser, and the results of the conversion to dependency structures of (c) the Penn tree and of (d) the parser's output

label, stripped of all functional tags and coindexing marks, conjoined with the label of the non-head daughter, with its functional tags but without coindexing marks. Figure 1 shows an example of the original Penn annotation (a), the output of Charniak's parser (b) and the results of our conversion of these trees to dependency structures (c and d). The interpretation of the dependency labels is straightforward: e.g., the label S|NP-TMP corresponds to a sentence (S) being modified by a temporal noun phrase (NP-TMP).

The core of the conversion routine is the selection of head daughters of the constituents. Following (Collins, 1999), we used a head table, but extended it with a set of additional rules, based on constituent labels, POS tags or, sometimes actual words, to account for situations where the head table alone gave unsatisfactory results. The most notable extension is our handling of conjunctions, which are often left relatively flat in WSJ and, as a result, in a parser's output: we used simple pattern-based heuristics to detect conjuncts and mark all conjuncts as heads of a conjunction.

After the conversion, every resulting dependency structure is modified deterministically:

- auxiliary verbs (*be, do, have*) become dependents of corresponding main verbs (similar to modal verbs, which are handled by the head table);
- to fix a WSJ inconsistency, we move the -LGS tag (indicating logical subject of passive in a *by*-phrase) from the PP to its child NP.

5 Dependency-based Evaluation of Parsers

After the original WSJ structures and the parsers' outputs have been converted to dependency structures, we evaluate the performance of the parsers against the dependency corpus. We use the standard precision/recall measures over sets of dependencies (excluding punctuation marks, as usual) and evaluate Collins' and Charniak's parsers on WSJ section 23 in three settings:

- on unlabelled dependencies;
- on labelled dependencies with only bare labels (all functional tags discarded);
- on labelled dependencies with functional tags.

Notice that since neither Collins' nor Charniak's parser outputs WSJ functional labels, all dependencies with functional labels in the gold parse will be judged incorrect in the third setting. The evaluation results are shown in Table 1, in the row "step 0".[4]

As explained above, the low numbers for the dependency evaluation with functional tags are expected, because the two parsers were not intended to produce functional labels.

Interestingly, the ranking of the two parsers is different for the dependency-based evaluation than for PARSEVAL: Charniak's parser obtains a higher PARSEVAL score than Collins' (89.0% vs. 88.2%),

[4]For meaningful comparison, the Collins' tags -A and -g are removed in this evaluation.

Evaluation	Parser	unlabelled			labelled			with func. tags		
		P	R	f	P	R	f	P	R	f
after conversion	Charniak	89.9	83.9	86.8	85.9	80.1	82.9	68.0	63.5	65.7
(step 0, Section 4)	Collins	90.4	83.7	87.0	86.7	80.3	83.4	68.4	63.4	65.8
after relabelling	Charniak	89.9	83.9	86.8	86.3	80.5	83.3	83.8	78.2	80.9
(step 1, Section 6)	Collins	90.4	83.7	87.0	87.0	80.6	83.7	84.6	78.4	81.4
after adding nodes	Charniak	90.1	85.4	87.7	86.5	82.0	84.2	84.1	79.8	81.9
(step 2, Section 7)	Collins	90.6	85.3	87.9	87.2	82.1	84.6	84.9	79.9	82.3
after adding arcs	Charniak	90.0	89.7	89.8	86.5	86.2	86.4	84.2	83.9	84.0
(step 3, Section 8)	Collins	90.4	89.4	89.9	87.1	86.2	86.6	84.9	83.9	84.4

Table 1: Dependency-based evaluation of the parsers after different transformation steps

but slightly lower f-score on dependencies without functional tags (82.9% vs. 83.4%).

To summarize the evaluation scores at this stage, both parsers perform with f-score around 87% on unlabelled dependencies. When evaluating on bare dependency labels (i.e., disregarding functional tags) the performance drops to 83%. The new errors that appear when taking labels into account come from different sources: incorrect POS tags (NN vs. VBG), different degrees of flatness of analyses in gold and test parses (JJ vs. ADJP, or CD vs. QP) and inconsistencies in the Penn annotation (VP vs. RRC). Finally, the performance goes down to around 66% when taking into account functional tags, which are not produced by the parsers at all.

6 Step 1: Changing Dependency Labels

Intuitively, it seems that the 66% performance on labels with functional tags is an underestimation, because much of the missing information is easily recoverable. E.g., one can think of simple heuristics to distinguish subject NPs, temporal PPs, etc., thus introducing functional labels and improving the scores. Developing such heuristics would be a very time consuming and ad hoc process: e.g., Collins' `-A` and `-g` tags may give useful clues for this labelling, but they are not available in the output of other parsers. As an alternative to hard-coded heuristics, Blaheta and Charniak (2000) proposed to recover the Penn functional tags automatically. On the Penn Treebank, they trained a statistical model that, given a constituent in a parsed sentence and its context (parent, grandparent, head words thereof etc.), predicted the functional label, possibly empty. The method gave impressive performance, with 98.64% accuracy on all constituents and 87.28% f-score for non-empty functional labels, when applied to constituents correctly identified by Charniak's parser. If we extrapolate these results to labelled PARSEVAL *with* functional labels, the method would give around 87.8% performance (98.64% of the "usual" 89%) for Charniak's parser.

Adding functional labels can be viewed as a relabelling task: we need to change the labels produced by a parser. We considered this more general task, and used a different approach, taking dependency graphs as input. We first parsed the training part of our dependency treebank (sections 02–21) and identified possible relabellings by comparing dependencies output by a parser to dependencies from the treebank. E.g., for Collins' parser the most frequent relabellings were S|NP→S|NP-SBJ, PP|NP-A→PP|NP, VP|NP-A→VP|NP, S|NP-A→S|NP-SBJ and VP|PP→VP|PP-CLR. In total, around 30% of all the parser's dependencies had different labels in the treebank. We then learned a mapping from the parser's labels to those in the dependency corpus, using TiMBL, a memory-based classifier (Daelemans et al., 2003). The features used for the relabelling were similar to those used by Blaheta and Charniak, but redefined for dependency structures. For each dependency we included:

- the head (h) and dependent (d), their POS tags;
- the leftmost dependent of d and its POS;
- the head of h (h'), its POS and the label of the dependency $h \rightarrow h'$;
- the closest left and right siblings of d (dependents of h) and their POS tags;
- the label of the dependency ($d \rightarrow h$) as derived from the parser's output.

When included in feature vectors, all dependency labels were split at '|', e.g., the label S|NP-A resulted in two features: S and NP-A.

Testing was done as follows. The test corpus (section 23) was also parsed, and for each dependency a feature vector was formed and given to

TiMBL to correct the dependency label. After this transformation the outputs of the parsers were evaluated, as before, on dependencies in the three settings. The results of the evaluation are shown in Table 1 (the row marked "step 1").

Let us take a closer look at the evaluation results. Obviously, relabelling does not change the unlabelled scores. The 1% improvement for evaluation on bare labels suggests that our approach is capable not only of adding functional tags, but can also correct the parser's phrase labels and part-of-speech tags: for Collins' parser the most frequent correct changes not involving functional labels were NP|NN→NP|JJ and NP|JJ→NP|VBN, fixing POS tagging errors. A very substantial increase of the labelled score (from 66% to 81%), which is only 6% lower than unlabelled score, clearly indicates that, although the parsers do not produce functional labels, this information is to a large extent implicitly present in trees and can be recovered.

6.1 Comparison to Earlier Work

One effect of the relabelling procedure described above is the recovery of Penn functional tags. Thus, it is informative to compare our results with those reported in (Blaheta and Charniak, 2000) for this same task. Blaheta and Charniak measured tagging accuracy and precision/recall for functional tag identification only for constituents *correctly identified* by the parser (i.e., having the correct span and nonterminal label). Since our method uses the dependency formalism, to make a meaningful comparison we need to model the notion of a constituent being correctly found by a parser. For a word w we say that the constituent corresponding to its maximal projection is *correctly identified* if there exists h, the head of w, and for the dependency $w \rightarrow h$ the right part of its label (e.g., NP-SBJ for S|NP-SBJ) is a nonterminal (i.e., not a POS tag) and matches the right part of the label in the gold dependency structure, after stripping functional tags. Thus, the constituent's label and headword should be correct, but not necessarily the span. Moreover, 2.5% of all constituents with functional labels (246 out of 9928 in section 23) are not maximal projections. Since our method ignores functional tags of such constituents (these tags disappear after the conversion of phrase structures to dependency graphs), we consider them as errors, i.e., reducing our recall value.

Below, the tagging accuracy, precision and recall are evaluated on constituents correctly identified by Charniak's parser for section 23.

Method	Accuracy	P	R	f
Blaheta	**98.6**	87.2	**87.4**	87.3
This paper	94.7	**90.2**	86.9	**88.5**

The difference in the accuracy is due to two reasons. First, because of the different definition of a *correctly identified constituent* in the parser's output, we apply our method to a greater portion of all labels produced by the parser (95% vs. 89% reported in (Blaheta and Charniak, 2000)). This might make the task for out system more difficult. And second, whereas 22% of *all* constituents in section 23 have a functional tag, 36% of the *maximal projections* have one. Since we apply our method only to labels of maximal projections, this means that our accuracy baseline (i.e., never assign any tag) is lower.

7 Step 2: Adding Missing Nodes

As the row labelled "step 1" in Table 1 indicates, for both parsers the recall is relatively low (6% lower than the precision): while the WSJ trees, and hence the derived dependency structures, contain non-local dependencies and empty nodes, the parsers simply do not provide this information. To make up for this, we considered two further tranformations of the output of the parsers: adding new nodes (corresponding to empty nodes in WSJ), and adding new labelled arcs. This section describes the former modification and Section 8 the latter.

As described in Section 4, when converting WSJ trees to dependency structures, traces are resolved, their empty nodes removed and new dependencies introduced. Of the remaining empty nodes (i.e., non-traces), the most frequent in WSJ are: NP PRO, empty units, empty complementizers, empty relative pronouns. To add missing empty nodes to dependency graphs, we compared the output of the parsers on the strings of the training corpus after steps 0 and 1 (conversion to dependencies and relabelling) to the structures in the corpus itself. We trained a classifier which, for every word in the parser's output, had to decide whether an empty node should be added as a new dependent of the word, and what its symbol ('*', '*U*' or '0' in WSJ), POS tag (always -NONE- in WSJ) and the label of the new dependency (e.g., 'S|NP-SBJ' for NP PRO and 'VP|SBAR' for empty complementizers) should be. This decision is conditioned on the word itself and its context. The features used were:

- the word and its POS tag, whether the word has any subject and object dependents, and whether it is the head of a finite verb group;
- the same information for the word's head (if

any) and also the label of the corresponding dependency;

- the same information for the rightmost and leftmost dependents of the word (if exist) along with their dependency labels.

In total, we extracted 23 symbolic features for every word in the corpus. TiMBL was trained on sections 02–21 and applied to the output of the parsers (after steps 0 and 1) on the test corpus (section 23), producing a list of empty nodes to be inserted in the dependency graphs. After insertion of the empty nodes, the resulting structures were evaluated against section 23 of the gold dependency treebank. The results are shown in Table 1 (the row "step 2"). For both parsers the insertion of empty nodes improves the recall by 1.5%, resulting in a 1% increase of the f-score.

7.1 Comparison to Earlier Work

A procedure for empty node recovery was first described in (Johnson, 2002), along with an evaluation criterion: an empty node is correct if its category and position in the sentence are correct. Since our method works with dependency structures, not phrase trees, we adopt a different but comparable criterion: an empty node should be attached as a dependent to the correct word, and with the correct dependency label. Unlike the first metric, our correctness criterion also requires that possible attachment ambiguities are resolved correctly (e.g., as in *the number of reports 0 they sent*, where the empty relative pronoun may be attached either to *number* or to *reports*).

For this task, the best published results (using Johnson's metric) were reported by Dienes and Dubey (2003), who used shallow tagging to insert empty elements. Below we give the comparison to our method. Notice that this evaluation does not include traces (i.e., empty elements with antecedents): recovery of traces is described in Section 8.

Type	This paper			Dienes&Dubey		
	P	R	f	P	R	f
PRO-NP	**73.1**	63.89	68.1	68.7	**70.4**	**69.5**
COMP-SBAR	82.6	**83.1**	82.8	**93.8**	78.6	**85.5**
COMP-WHNP	65.3	**40.0**	**49.6**	**67.2**	38.3	48.8
UNIT	95.4	91.8	93.6	**99.1**	**92.5**	**95.7**

For comparison we use the notation of Dienes and Dubey: PRO-NP for uncontrolled PROs (nodes '*' in the WSJ), COMP-SBAR for empty complementizers (nodes '0' with dependency label VP|SBAR), COMP-WHNP for empty relative pronouns (nodes '0' with dependency label X|SBAR, where X≠VP) and UNIT for empty units (nodes '*U*').

It is interesting to see that for empty nodes except for UNIT both methods have their advantages, showing better precision or better recall. Yet shallow tagging clearly performs better for UNIT.

8 Step 3: Adding Missing Dependencies

We now get to the third and final step of our transformation method: adding missing arcs to dependency graphs. The parsers we considered do not explicitly provide information about non-local dependencies (control, WH-extraction) present in the treebank. Moreover, newly inserted empty nodes (step 2, Section 7) might also need more links to the rest of a sentence (e.g., the inserted empty complementizers). In this section we describe the insertion of missing dependencies.

Johnson (2002) was the first to address recovery of non-local dependencies in a parser's output. He proposed a pattern-matching algorithm: first, from the training corpus the patterns that license non-local dependencies are extracted, and then these patterns are detected in unseen trees, dependencies being added when matches are found. Building on these ideas, Jijkoun (2003) used a machine learning classifier to detect matches. We extended Jijkoun's approach by providing the classifier with lexical information and using richer patterns with labels containing the Penn functional tags and empty nodes, detected at steps 1 and 2.

First, we compared the output of the parsers on the strings of the training corpus after steps 0, 1 and 2 to the dependency structures in the training corpus. For every dependency that is missing in the parser's output, we find the shortest undirected path in the dependency graph connecting the head and the dependent. These paths, connected sequences of labelled dependencies, define the set of possible patterns. For our experiments we only considered patterns occuring more than 100 times in the training corpus. E.g., for Collins' parser, 67 different patterns were found.

Next, from the parsers' output on the strings of the training corpus, we extracted all occurrences of the patterns, along with information about the nodes involved. For every node in an occurrence of a pattern we extracted the following features:

- the word and its POS tag;
- whether the word has subject and object dependents;
- whether the word is the head of a finite verb cluster.

We then trained TiMBL to predict the label of the missing dependency (or 'none'), given an occurrence of a pattern and the features of all the nodes involved. We trained a separate classifier for each pattern.

For evaluation purposes we extracted all occurrences of the patterns and the features of their nodes from the parsers' outputs for section 23 after steps 0, 1 and 2 and used TiMBL to predict and insert new dependencies. Then we compared the resulting dependency structures to the gold corpus. The results are shown in Table 1 (the row "step 3"). As expected, adding missing dependencies substantially improves the recall (by 4% for both parsers) and allows both parsers to achieve an 84% f-score on dependencies with functional tags (90% on unlabelled dependencies). The unlabelled f-score 89.9% for Collins' parser is close to the 90.9% reported in (Collins, 1999) for the evaluation on unlabelled *local* dependencies only (without empty nodes and traces). Since as many as 5% of all dependencies in WSJ involve traces or empty nodes, the results in Table 1 are encouraging.

8.1 Comparison to Earlier Work

Recently, several methods for the recovery of non-local dependencies have been described in the literature. Johnson (2002) and Jijkoun (2003) used pattern-matching on local phrase or dependency structures. Dienes and Dubey (2003) used shallow preprocessing to insert empty elements in raw sentences, making the parser itself capable of finding non-local dependencies. Their method achieves a considerable improvement over the results reported in (Johnson, 2002) and gives the best evaluation results published to date. To compare our results to Dienes and Dubey's, we carried out the transformation steps 0–3 described above, with a single modification: when adding missing dependencies (step 3), we only considered patterns that introduce non-local dependencies (i.e., traces: we kept the information whether a dependency is a trace when converting WSJ to a dependency corpus).

As before, a dependency is correctly found if its head, dependent, and label are correct. For traces, this corresponds to the evaluation using the head-based antecedent representation described in (Johnson, 2002), and for empty nodes without antecedents (e.g., NP PRO) this is the measure used in Section 7.1. To make the results comparable to other methods, we strip functional tags from the dependency labels before label comparison. Below are the overall precision, recall, and f-score for our method and the scores reported in (Dienes and Dubey, 2003) for antecedent recovery using Collins' parser.

Method	P	R	f
Dienes and Dubey	81.5	**68.7**	**74.6**
This paper	**82.8**	67.8	**74.6**

Interestingly, the overall performance of our post-processing method is very similar to that of the pre- and in-processing methods of Dienes and Dubey (2003). Hence, for most cases, traces and empty nodes can be reliably identified using only local information provided by a parser, using the parser itself as a black box. This is important, since making parsers aware of non-local relations need not improve the overall performance: Dienes and Dubey (2003) report a decrease in PARSEVAL f-score from 88.2% to 86.4% after modifying Collins' parser to resolve traces internally, although this allowed them to achieve high accuracy for traces.

9 Discussion

The experiments described in the previous sections indicate that although statistical parsers do not explicitly output some information available in the corpus they were trained on (grammatical and semantic tags, empty nodes, non-local dependencies), this information can be recovered with reasonably high accuracy, using pattern matching and machine learning methods.

For our task, using dependency structures rather than phrase trees has several advantages. First, after converting both the treebank trees and parsers' outputs to graphs with head–modifier relations, our method needs very little information about the linguistic nature of the data, and thus is largely corpus- and parser-independent. Indeed, after the conversion, the only linguistically informed operation is the straightforward extraction of features indicating the presence of subject and object dependents, and finiteness of verb groups.

Second, using a dependency formalism facilitates a very straightforward evaluation of the systems that produce structures more complex than trees. It is not clear whether the PARSEVAL evaluation can be easily extended to take non-local relations into account (see (Johnson, 2002) for examples of such extension).

Finally, the independence from the details of the parser and the corpus suggests that our method can be applied to systems based on other formalisms, e.g., (Hockenmaier, 2003), to allow a meaningful dependency-based comparison of very different parsers. Furthermore, with the fine-grained set of dependency labels that our system provides, it is

possible to map the resulting structures to other dependency formalisms, either automatically in case annotated corpora exist, or with a manually developed set of rules. Our preliminary experiments with Collins' parser and the corpus annotated with grammatical relations (Carroll et al., 2003) are promising: the system achieves 76% precision/recall f-score, after the parser's output is enriched with our method and transformed to grammatical relations using a set of 40 simple rules. This is very close to the performance reported by Carroll et al. (2003) for the parser specifically designed for the extraction of grammatical relations.

Despite the high-dimensional feature spaces, the large number of lexical features, and the lack of independence between features, we achieved high accuracy using a memory-based learner. TiMBL performed well on tasks where structured, more complicated and task-specific statistical models have been used previously (Blaheta and Charniak, 2000). For all subtasks we used the same settings for TiMBL: simple feature overlap measure, 5 nearest neighbours with majority voting. During further experiments with our method on different corpora, we found that quite different settings led to a better performance. It is clear that more careful and systematic parameter tuning and the analysis of the contribution of different features have to be addressed.

Finally, our method is not restricted to syntactic structures. It has been successfully applied to the identification of semantic relations (Ahn et al., 2004), using FrameNet as the training corpus. For this task, we viewed semantic relations (e.g., *Speaker*, *Topic*, *Addressee*) as dependencies between a predicate and its arguments. Adding such semantic relations to syntactic dependency graphs was simply an additional graph transformation step.

10 Conclusions

We presented a method to automatically enrich the output of a parser with information that is not provided by the parser itself, but is available in a treebank. Using the method with two state of the art statistical parsers and the Penn Treebank allowed us to recover functional tags (grammatical and semantic), empty nodes and traces. Thus, we are able to provide virtually all information available in the corpus, without modifying the parser, viewing it, indeed, as a black box.

Our method allows us to perform a meaningful dependency-based comparison of phrase structure parsers. The evaluation on a dependency corpus derived from the Penn Treebank showed that, after our post-processing, two state of the art statistical parsers achieve 84% accuracy on a fine-grained set of dependency labels.

Finally, our method for enriching the output of a parser is, to a large extent, independent of a specific parser and corpus, and can be used with other syntactic and semantic resources.

11 Acknowledgements

We are grateful to David Ahn and Stefan Schlobach and to the anonymous referees for their useful suggestions. This research was supported by grants from the Netherlands Organization for Scientific Research (NWO) under project numbers 220-80-001, 365-20-005, 612.069.006, 612.000.106, 612.000.207 and 612.066.302.

References

David Ahn, Sisay Fissaha, Valentin Jijkoun, and Maarten de Rijke. 2004. The University of Amsterdam at Senseval-3: semantic roles and logic forms. In *Proceedings of the ACL-2004 Workshop on Evaluation of Systems for the Semantic Analysis of Text*.

Don Blaheta and Eugene Charniak. 2000. Assigning function tags to parsed text. In *Proceedings of the 1st Meeting of NAACL*, pages 234–240.

John Carroll, Guido Minnen, and Ted Briscoe. 2003. Parser evaluation using a grammatical relation annotation scheme. In Anne Abeillé, editor, *Building and Using Parsed Corpora*, pages 299–316. Kluwer.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of NAACL*, pages 132–139.

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

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch, 2003. *TiMBL: Tilburg Memory Based Learner, version 5.0, Reference Guide*. ILK Technical Report 03-10. Available from http://ilk.kub.nl/downloads/pub/papers/ilk0310.ps.gz.

Péter Dienes and Amit Dubey. 2003. Antecedent recovery: Experiments with a trace tagger. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 33–40.

Julia Hockenmaier. 2003. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Meeting of ACL*, pages 359–366.

Valentin Jijkoun. 2003. Finding non-local dependencies: Beyond pattern matching. In *Proceedings of the ACL-2003 Student Research Workshop*, pages 37–43.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th meeting of ACL*, pages 136–143.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of ACL*, pages 423–430.