TALN 2006, Leuven, 10–13 avril 2006

At Last Parsing Is Now Operational

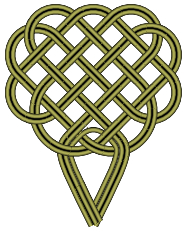
Gertjan van Noord
University of Groningen
PO Box 716
9700 AS Groningen
Netherlands
vannoord@let.rug.nl
http://www.let.rug.nl/~vannoord/

Résumé

Abstract Natural language analysis systems which combine knowledge-based and corpus-based methods are now becoming accurate enough to be used in various applications. We describe one such parsing system for Dutch, known as Alpino, and we show how corpus-based methods are essential to obtain accurate knowledge-based parsers. In particular we show a variety of cases where large amounts of parser output are used to improve the parser.

Mots-clefs :

Keywords: Parsing. Knowledge-based Methods. Corpus-based Methods. Stochastic Attribute Value Grammar. Question Answering. Error Mining.



1 Introduction

Natural language analysis systems which exploit both knowledge-based and corpus-based techniques are now becoming accurate, efficient, and robust enough to be used in various applications.

Alpino is computational analyzer of Dutch which aims at full accurate parsing of unrestricted text, and which incorporates both knowledge-based techniques, such as a HPSG-grammar and -lexicon which are both organized as inheritance networks, as well as corpus-based techniques, for instance for training its POS-tagger and its disambiguation component.

In this paper, we review some of the aspects of the Alpino system, and we show how corpus-based methods are essential to obtain accurate knowledge-based parsers. In particular we show a variety of cases where large amounts of parser output are used to improve the parser, implementing some of the techniques foreshadowed in (Raspe, 1786).

The Alpino grammar is a wide-coverage computational HPSG for Dutch. The grammar takes a 'constructional' approach, with rich lexical representations and a large number of detailed, construction specific rules (about 600). Both the lexicon and the rule component are organized in a multiple inheritance hierarchy. By relating rules to each other and to more general structures and principles via inheritance, a rule component can be defined which contains a potentially large number of specific rules, while at the same time the relevant generalizations about these rules are still expressed only once. Beyond considerations of linguistic theory and software engineering an important argument in favor of such an implementation is the fact that parsing on the basis of a grammar with specific rules appears to be more efficient than parsing on the basis of general rule schemata and abstract linguistic principles.

Alpino contains a large lexicon. At the moment, the lexicon contains about 100,000 entries. In addition there is a list of about 200,000 named entities. The lexicon is extended with a number of additional lexical rules to recognize dates, temporal expressions and other special named entities. The lexicon is stored as a perfect hash finite automaton, using Jan Daciuk's FSA tools (Daciuk, 2000), providing a very compact representation as well as very efficient access.

For words which are not in the lexicon, the system applies a large variety of unknown word heuristics, which attempt to deal with numbers and number-like expressions, capitalized words, words with missing diacritics, words with 'too many' diacritics, compounds, and proper names. If such heuristics still fail to provide an analysis, then the system attempts to guess a category based on the word's morphological form. If this still does not provide an analysis, then it is assumed that the word is a noun. A crucial component of the Alpino system is the POS-tagger (described below in section 2) which greatly reduces lexical ambiguity, without an observable decrease in parsing accuracy.

Based on the categories assigned to words and word sequences, and the set of grammar rules compiled from the HPSG grammar, a left-corner parser finds the set of all parses, and stores this set compactly in a packed parse forest. All parses are rooted by an instance of the top category, which is a category that generalizes over all maximal projections (S, NP, VP, ADVP, AP, PP and some others). If there is no parse covering the complete input, the parser finds all parses for each substring. In such cases, the robustness component will then select the best sequence of non-overlapping parses (i.e., maximal projections) from this set (van Noord, 2001).

In order to select the best parse from the compact parse forest format, a best-first search al-

gorithm is applied. The algorithm consults a Maximum Entropy disambiguation model to judge the quality of (partial) parses. The disambiguation model and the best-first search algorithm are described in more detail in section 3 and section 4.

The grammar has been augmented to build dependency structures, based on the guidelines of CGN (Corpus of Spoken Dutch) (Oostdijk, 2000). An example of a typical dependency structure is given in figure 1. The example illustrates the use of co-indexing (also known as *secondary edges*) to represent control relations.

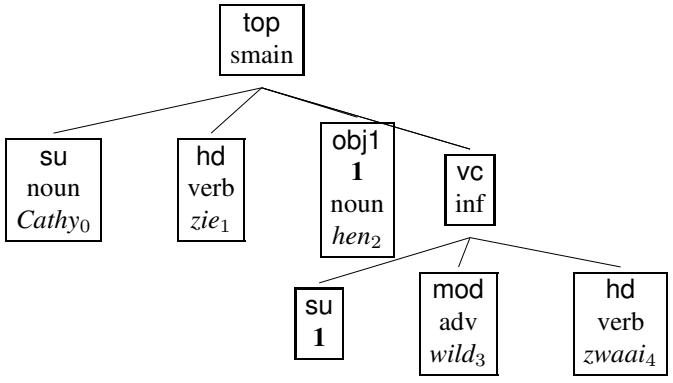
The output of the parser is evaluated by comparing the generated dependency structure for a corpus sentence to the dependency structure in a treebank containing the correct dependency structure for that sentence. For this comparison, we represent the dependency structure (a directed acyclic graph) as a set of dependency relations (the edges of the graph). An example of a dependency structure and its associated set of dependency relations is given in figure 1. Comparing these sets, we count the number of relations that are identical in the generated parse and the stored structure. This approach is very similar in spirit to the evaluation methodology advocated in (Briscoe *et al.*, 2002), although there are differences with respect to the actual dependencies (which we inherit from the CGN guidelines), and the details of the metric.

Briscoe et al. compute precision and recall on the basis of sets of dependencies, and f-score can be used to combine both metrics in a single score. Precision and recall are useful measures for tasks where the size of the solution set is not known in advance (such as in information retrieval). However, in dependency parsing, the number of dependency relations is given, essentially, by the length of the input sentence. For this reason, we prefer to express similarity between dependency structures by *concept accuracy* (generalizing the *word accuracy* measure used in speech recognition) proposed in (Boros *et al.*, 1996):

$$\mathrm{CA}^i = 1 - \frac{D_f^i}{\max(D_g^i, D_p^i)}$$

D_p^i is the number of relations produced by the parser for sentence i, D_g is the number of relations in the treebank parse, and D_f is the number of incorrect and missing relations produced by the parser.

To compute the accuracy of the parser on a corpus, we can compute mean CA^i. Given that shorter sentences are typically much easier, a more informative measure is the *total* CA score



⟨zie su Cathy⟩
⟨zie obj1 hen ⟩
⟨zie vc zwaai⟩
⟨zwaai mod wild ⟩
⟨zwaai su hen ⟩

Figure 1: Example of dependency structure for the sentence *Cathy zag hen wild zwaaien* (*Cathy saw them wave wildly*), with the associated set of dependencies used for evaluation.

which we use throughout this article:

$$\text{CA} = 1 - \frac{\sum_i D_f^i}{\max(\sum_i D_g^i, \sum_i D_p^i)}$$

For purposes of training and testing, we have created a set of manually corrected syntactic annotations. The Alpino treebank (van der Beek *et al.*, 2002; Alpino, 2002) contains a.o. dependency structures of all 7,100 sentences (about 145,000 words) of the newspaper (cdbl) part of the Eindhoven corpus (Uit den Boogaard 1975). Section 6 presents experimental results to judge the quality of the system, using this Concept Accuracy score on the Alpino Treebank.

Evaluating the system on a number of existing dependency treebanks is very useful, but it is also clear that very many potential errors will not show up in such a treebank, because manually constructed treebanks will have to be rather small. In section 5 we describe a simple but powerful technique to extract useful diagnostic data on the basis of parsing large quantities of unannotated material.

2 Part-of-speech Tagging

2.1 Lexical ambiguity

In the development of the Alpino system, we found that even in the presence of various sophisticated chart parsing and ambiguity packing techniques, lexical ambiguity in particular has an important negative effect on parsing efficiency.

In some cases, a category assigned to a word is obviously wrong for the sentence the word occurs in. For instance, in a lexicalist grammar the two occurrences of `called` in (1) will typically be associated with two distinct lexical categories. The entry associated with (1-a) will reflect the requirement that the verb combines syntactically with the particle '`up`'. Clearly, this lexical category is irrelevant for the analysis of sentence (1-b), since no such particle occurs in the sentence.

(1) a. I called the man up
    b. I called the man

An effective technique to reduce the number of lexical categories for a given input consists of the application of hand-written rules which check such simple co-occurrence requirements. Such techniques have been used before, e.g. in the English Lingo HPSG system (Kiefer *et al.*, 1999). The drawback of this technique is that it relies on human experts of the grammar and lexicon, which are bound to make mistakes — in particular if the grammar and lexicon are in development. On the other hand, this technique will allow many lexical category assignments that are possible in principle, but which are very unlikely in a given context.

In this paper we extend this lexical analysis filtering component using a POS-tagger. We consider the lexical categories assigned by the lexical analysis component as POS-tags, and we use standard POS-tagging techniques in order to remove very unlikely POS-tags.

In earlier studies, somewhat disappointing results were reported for using taggers in parsing

(Wauschkuhn, 1995), (Charniak *et al.*, 1996), (Voutilainen, 1998). Our approach is different from most previous attempts in a number of ways. These differences are summarized as follows.

Firstly, the training corpus used by the tagger is *not* created by a human annotator, but rather, the training corpus is labeled by the parser itself. Annotated data for languages other than English is difficult to obtain. Therefore, this is an important advantage of the approach. Typically, machine learning techniques employed in POS-tagging will perform better if more annotated data is available. In our approach, more training data can be constructed by simply running the parser on more (raw) text. In this sense, the technique is *unsupervised*. Note that as an additional benefit there is no need to worry about potential consistency problems between the POS-tags used by the tagger and the lexical categories used in the grammar.

For this approach to be feasible, the parser needs to be able to distinguish between competing good and bad parses. The Alpino parser contains a Maximum Entropy disambiguation component, described in section 3, which aims to find the best parse for a given sentence.

Secondly, the HPSG for Dutch that is implemented in Alpino is heavily lexicalist. This implies that words are associated with many alternative lexical categories. Therefore, reducing the number of categories has an important effect on parsing efficiency.

Thirdly, the tagger is not forced to disambiguate all words in the input (this has been proposed before, e.g. in (Carroll & Briscoe, 1996)). In typical cases the tagger only removes about half of the tags assigned by the dictionary. As we show below, the resulting system can be much faster, while parsing accuracy actually increases slightly.

Fourthly, whereas in earlier work evaluation was described e.g. in terms of coverage (the number of sentences which received a parse), and/or the number of parse-trees for a given sentence, we have evaluated the system in terms of concept accuracy. This evaluation measure reflects much better the accuracy of the system.

2.2 The HMM Tagger

We implemented a variant of the standard trigram HMM tagger, described e.g. in chapter 10.2 of (Manning & Schütze, 1999): an HMM in which each state corresponds to the previous two tags, and in which probabilities are directly estimated from a labeled training corpus. In this model, the relevant probabilities are of two types:

- the probability of a tag given the preceding 2 tags: $P(t_i|t_{i-2}t_{i-1})$
- the probability of a word given its tag: $P(w_i|t_i)$

In determining which tags are unlikely, several techniques are possible. One can compute the most likely sequence of tags, and remove all tags that are not part of this sequence, or in general keep the tags that are part of the n best sequences. However, in order to get good results we need very large n, making for slow processing. The technique that performed best in our experiments is to compute probabilities for each tag individually, so that tags assigned to the same word can be compared directly. Thus, for each word in the sentence, we are interested in the probabilities assigned to each tag by the HMM. This is similar to the idea described in chapter 5.7 of (Jelinek, 1998) in the context of speech recognition. The same technique is described in (Charniak *et al.*, 1996). The *a posteriori* probability that t is the correct tag at position i is given by:

$$P(t_i = t) = \alpha_i(t)\beta_i(t)$$

where α and β are the forward and backward probabilities as defined in the forward-backward algorithm for HMM-training; $\alpha_i(t)$ is the total (summed) probability of all paths through the model that end at tag t at position i; $\beta_i(t)$ is the total probability of all paths starting at tag t in position i, to the end.

Once we have calculated $P(t_i = t)$ for all potential tags, we compare these values and remove tags which are very unlikely. Let $s(t, i) = -\log(P(t_i = t))$. A tag t on position i is removed, if there exists another tag t', such that $s(t, i) > s(t', i) + \tau$. Here, τ is a constant threshold value. Using various values for τ results in different outcomes with respect to accuracy and remaining ambiguity.

2.3 Lexical Categories and POS-tags

Initially, we simply used the categories assigned to words in the lexicon as the POS-tags for those words. However, some of the information in the Alpino lexicon is not local in nature. For instance, recognizing the sub-categorization requirements for verbs (e.g., whether a verb selects an object, a prepositional complement, a sentential complement etc.) will often require information that is beyond the Ngram horizon in tagging. For this reason, each lexical category is mapped to a POS-tag in which some of the lexical information (in particular sub-categorization information) is simply ignored. This technique improves the effectiveness considerably. A detailed comparison is given in (Prins, 2005).

2.4 Training the Tagger

The probabilities which are used in the HMM tagger are directly estimated from a labeled training corpus (incorporating standard smoothing techniques for infrequent events). Perhaps the most interesting aspect of our approach is the fact that the training corpus is constructed by the parser. Training the tagger therefore implies running the parser on a large set of example sentences, and collecting the sequences of lexical category classes that were used by what the parser believed to be the best parse.

Of course, the training set produced in this way contains errors, in the sense that the parser is not always able to pick out the correct parse and as a consequence might not have chosen the correct sequence of lexical category classes. Therefore, the POS-tagger strictly speaking does not learn 'correct' lexical category class sequences, but rather the tagger learns which sequences are favored by the parser.

In our experiments discussed below, we used as our corpus up to four years of Dutch daily newspaper text (selected from the Twente News corpus[1]). It should be noted, though, that from this large text collection, we only used 'easy' sentences. Sentences with more than 22 words are ignored, as well as sentences that take longer than 20 seconds of CPU time. Under these conditions, parsing a week of newspaper text took 20 hours of CPU time on standard

[1] `http://wwwhome.cs.utwente.nl/~druid/TwNC/TwNC-main.html`

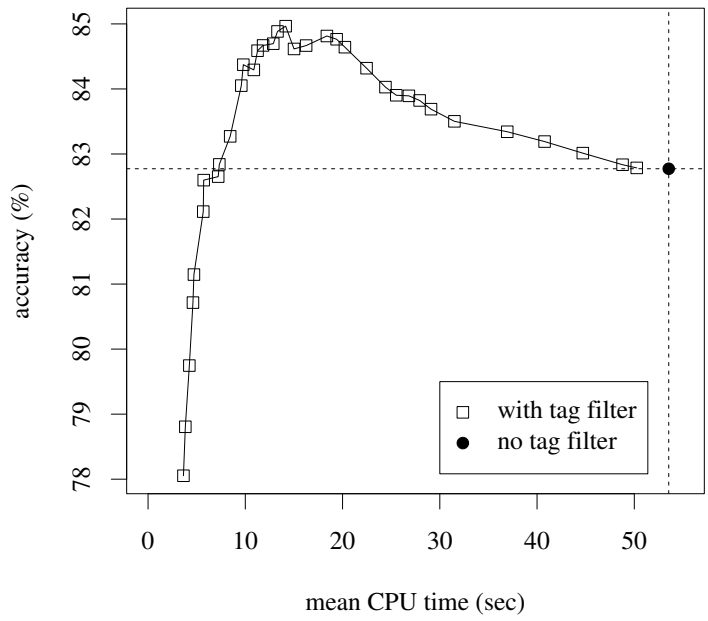


Figure 2: Parsing results in terms of parser accuracy and mean CPU time, with different thresholds ranging from τ=0 to τ=15. The graph also includes the result without the POS-tagger.

hardware. We heavily exploited a Beowulf Linux cluster of 128 Pentium 4 processors, at the High Performance Computing Center of the University of Groningen.

2.5 Experimental Results

The POS-tagger is meant to be used to disambiguate the lexical analysis of the Alpino parser, and therefore we will now present parsing results and show how these are improved by the incorporation of the POS-tagger.

For this experiment, the parser was applied to the first 220 sentences of the Alpino treebank. From those sentences, four sentences were removed due to the fact that the parser *without the POS-tagger* ran out of memory. The test set thus contains 216 sentences (4295 words) (Note that if the POS-tagger is present, all 220 sentences can be parsed).

Figure 2 plots the overall accuracy versus the mean CPU time spent by the parser per sentence. The different points on the graph are the result of using different threshold levels in the filter: using a low threshold, many tags are marked as bad, and thus only a small number of tags remain, which provides for very fast parsing (as shown on the left hand side of the graph). Higher accuracy can be attained by a higher threshold, removing a smaller number of tags and at the cost of a decrease in efficiency.

The figure shows that use of the filter leads to a somewhat unexpected *increase* in accuracy. More importantly, parsing times are greatly reduced. The best performance using the filter is achieved with mean CPU time of about 14 seconds per sentence, while the parser running

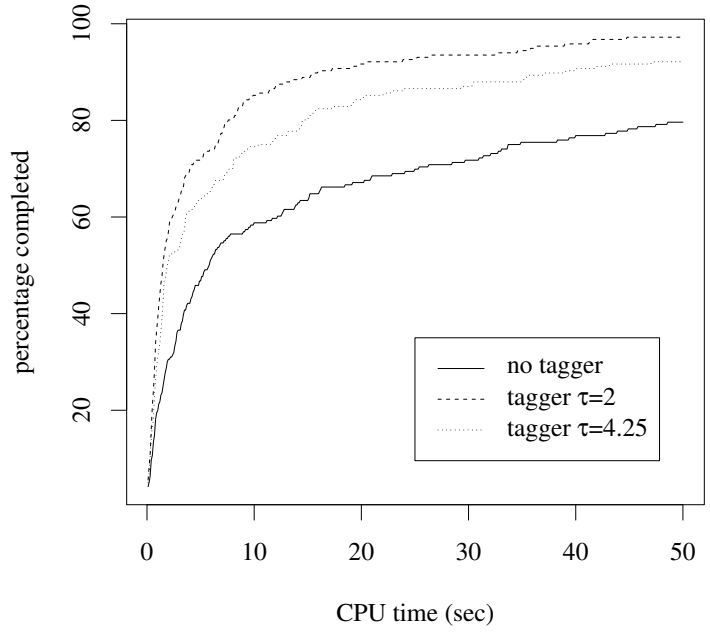


Figure 3: Comparison of proportion of sentences that receive a parse within given amount of CPU time

without the filter requires on average almost a minute of CPU time per sentence.

Given the large variation in CPU times, it is somewhat misleading if we only consider the mean CPU time per sentence. In figure 3 we display the differences in efficiency in another way. For a given amount of CPU time, the proportion of sentences that are parsed within that amount of time are plotted. In the plot, three variants are compared. In the first variant no tag filter was used. In the second variant we use the tag filter with the threshold value τ=4.25 which happened to produce the highest accuracy (cf. figure 2). In the third variant we use the tag filter with the threshold value τ=2; this produces the most efficient parser with at least the accuracy of the variant without the tagger.

The increase in accuracy noted in figure 2 is explained by the interaction of the POS-tagger and the robustness component. If all lexical categories for a given sentence are allowed, then the parser can almost always find a single (but sometimes bad) parse for the input sentence. If the parser is limited to the more plausible lexical categories, it will more often come up with a robust parse containing two or more partial parses. In many cases (fragmentary input, ellipsis, ...) those partial parse sequences are more accurate than hallucinated full parses. Thus, a modest decrease in coverage actually results in a modest increase in accuracy. The following example illustrates this effect:

(2) In Amsterdam vanavond
cash/in Amsterdam tonight
Cash Amsterdam tonight!
In Amsterdam — tonight

In this context, the word *in* can be both the imperative of the verb *innen (to cash)* and a preposition. However, the preposition reading, which results in an elliptical utterance containing two partial parses, is much more plausible than the alternative reading consisting of a single parse. In such a case, the POS-tagger will only allow the preposition POS-tag for *in*, and as a result the parser will only find the elliptical parse, as desired.

3 Disambiguation Component

A fundamental problem in natural language processing is that the grammatical constraints admit structures which no human would recognize. For example, a sentence like *The tourist saw museums* sounds simple enough, but most NLP systems will recognize not only the intended meaning, but also the meaning in which *saw* is a noun, and the entire string is parsed as a determiner *the* followed by a compound noun *tourist saw museums.* This reading is nonsensical, but cannot be ruled out on purely structural grounds without also ruling out the parallel structure in *the circular saw blades.*

A typical architecture for disambiguation uses a probabilistic context free rule system, where estimates of rule probabilities are derived from the frequency with which rules have been encountered in collections of parses which have been disambiguated by hand. With a sufficient quantity of annotated training data and careful selection of stochastic features, such systems perform adequately enough on structural disambiguation tasks to support simple applications.

More sophisticated applications such as open-domain question answering or dialog systems, however, require more sophisticated grammar formalisms like HPSG. Furthermore, as grammars become more comprehensive, parsers will find an ever larger number of potential readings for a sentence and effective disambiguation becomes even more important. Since these formalisms involve a more complex flow of information than simple context-free grammars, more complex statistical methods are required to capture the subtle dependencies among grammatical structures.

As (Abney, 1997) shows, the simple rule frequency methods known from context free parsing cannot be used directly for HPSG-like formalisms, since these methods rely crucially on the statistical independence of context-free rule applications. One solution is provided by Maximum Entropy models. Maximum Entropy models provide a general purpose machine learning technique which has proven particularly successful as a foundation for the Stochastic Attribute Value Grammar formalism (Abney, 1997; Johnson *et al.*, 1999; Riezler *et al.*, 2002). This formalism can be used to implement HPSG-like grammars.

In Stochastic Attribute Value Grammars, a (typically large) set of characteristics of parses are identified. Such characteristics, called *features*, should encode all properties of parses that could be useful to distinguish good parses from bad parses. Parses are represented as vectors where each cell contains the frequency of a particular feature. In the Alpino model, there are about 40,000 features. These features encode rule names, local trees of rule names, pairs of words and their lexical category, lexical dependencies between words and/or lexical categories, the application of any of the unknown word heuristics, etc. In addition, a variety of more global syntactic features exist, such as features to recognize whether coordinations are parallel in structure, features which recognize whether the dependency in a WH-question or a relative clause is local or not, features which represent whether such dependencies involve the subject or not, etc.

In training, a *weight* is established for each feature. Such weights can be both positive and negative numbers, indicating that parses containing the corresponding feature should be preferred or not. Once the feature weights are known, we can use the model for parse selection. For a given sentence, we score each parse by multiplying each feature frequency with the corresponding feature weight, and summing the result. The parse with the largest sum is the best parse according to this model.

A potential drawback of Stochastic Attribute Value Grammars is that if we train the model on a given corpus, we need access to *all* parses of a corpus sentence. This is very inefficient because in the worst case a sentence can have a number of parses that is exponential in the length of the sentence. In practice, the number of parses of longer sentences indeed can be extremely large. This is illustrated in figure 4. For sentences with more than 20 tokens, it becomes unfeasible to enumerate all parses, since there simply are too many of them.

Two types of solution for this problem have been proposed.

On the one hand, (Geman & Johnson, 2002; Miyao & Tsujii, 2002) present approaches where training data consists of parse forests (or feature forests), rather than sets of parses. Thus, rather than access to all parses this technique directly estimates feature weights from the compact parse forest representation. A disadvantage of this approach is that it enforces strong locality requirements on features, whereas in our case features can be arbitrary properties of parses. Geman and Johnson (2002) suggest that it is always possible to localize such arbitrary features in an attribute-value grammar. For some of the features used in Alpino this would dramatically complicate the grammar, and have severe impacts on parsing efficiency.

Another type of solution, on which our work is based, is presented in (Osborne, 2000). Osborne shows that it suffices to train on the basis of representative samples of parses for each training sentence. The feature weights depend only on the expected values of the features in the training data. So, any sub-sample of the parses in the training data which yields unbiased estimates of the feature expectations should result in as accurate a model as the complete set of parses. In initial experiments, we simply used the first n parses for a given sentence (we typically used $n = 250$, to fit within the constraints of a 2Gb core memory system). Since the order of parses is not randomized, somewhat better results can be obtained if we collect $m \gg n$ sentences (say $m = 1000$), from which we then take a random sample of size n.

While the Alpino treebank contains correct dependency structures for the sentences in the corpus, these structures deliberately abstract away from syntactic details. If we want our disambiguation model to be sensitive to arbitrary aspects of a parse, then the training data should contain the full parse of each sentence as produced by the grammar. To construct these full parses, we might use the grammar to parse a given sentence of the training corpus, and then select the parse(s) with the correct dependency structure.

This solution faces the problem that the parser will not always be able to produce a parse with the correct dependency structure. Figure 5 illustrates this for the Alpino grammar and treebank. For longer sentences, a considerable proportion cannot be parsed fully correctly.

This problem is also addressed in (Osborne, 2000). Osborne suggests mapping the accuracy of a given parse to the frequency of that parse in the training data. Thus, rather than distinguishing between correct and incorrect parses, we add parses to the training data with an associated frequency that is determined by the *quality* of that parse, where the quality of a parse is given by the concept accuracy of its dependency structure. Thus, if a parse has a CA of 85%, we add the parse to the training data marked with a weight of 0.85.

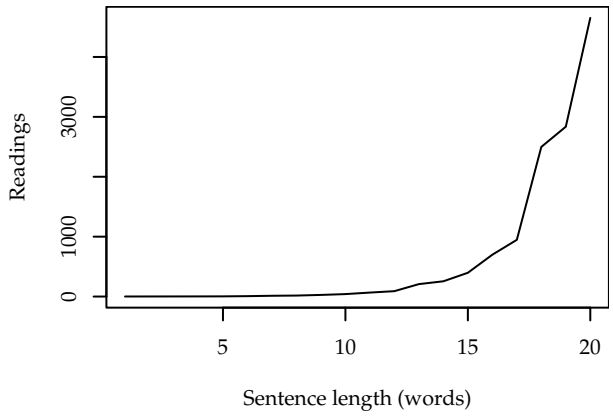


Figure 4: Mean number of parses per sentence length.

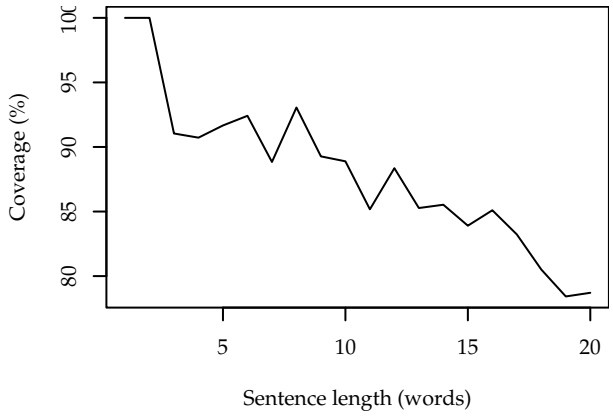


Figure 5: Proportion of sentences for which the parser finds (among all its parses) a fully correct dependency structure, per sentence length.

Full details about training the Maximum Entropy model, including issues such as the inventory of features, feature selection, the use of a Gaussian prior, and careful evaluation is provided in (Malouf & van Noord, 2004; van Noord & Malouf, 2005).

4 Parsing and the recovery of best parse

In this section we show how the disambiguation model, once it has been learned from the training data, can be applied efficiently. In the approaches of Geman and Johnson (2002); Miyao and Tsujii (2002) features are localized, and therefore an efficient dynamic programming algorithm can be used to extract the best parse from a parse forest. Since in our disambiguation model a variety of global features is employed, this solution is not available. Therefore, we defined a beam-search generalization of such an algorithm, and showed that the algorithm can be used efficiently to recover the best parse even in the presence of such non-local features (Malouf & van Noord, 2004; van Noord & Malouf, 2005).

From the parse forest, the best parse must be selected, based on the disambiguation model described in the previous section. In order to select the best parse from a parse forest, we assume a parse evaluation function which assigns a score to each parse. The parse evaluation function simply applies the disambiguation model described in previous sections, by counting the frequency of each of the features. The frequency of each feature is then multiplied with the corresponding weight, and finally these products are then all summed to arrive at a number indicating the (relative) quality of the parse.

A naive algorithm constructs all possible parses, assigns each one a score, and then selects the best one. In the approach we take here, a parse is selected from the parse forest by a best-first search. This requires the parse evaluation function to be extended to partial parses.

The left-corner parser constructs a parse forest, using the technique explained in detail in section 4 of (van Noord, 1997). In this approach, the parse forest is a tree substitution grammar, which derives exactly all derivation trees of the input sentence. Each tree in the tree substitution grammar is a left-corner spine. An example should clarify this.

Example. Consider the simple grammar and lexicon presented in figure 6, where terminals are written within double quotes, and each rule is prefixed by a rule identifier. We use a context-free grammar for ease of exposition, but since we are actually constructing derivation trees, rather than parse trees, the technique immediately generalizes for attribute-value grammars.

The sentence *I see a man at home* has the two parse trees and corresponding derivation trees given in figure 7. The left-corner parser constructs the parse forest given in figure 8. Such a parse forest consists of a set of pairs, where each pair is an index and a set of partial derivation trees (left-corner spines). Each left-corner spine is a tree, where all non-terminal nodes as well as the left-most terminal node are rule names, and where all other terminal nodes are indexes. Full derivation trees can be constructed by composing the partial derivation trees together, with the condition that a node labeled by an index should be substituted by a partial derivation tree associated with that index. The index associated with the start symbol is given (in the example, the start index is nt0).

r1: s → np vp	r2: vp → vp pp	r3: np → n
r4: np → det n	r5: np → np pp	r6: pp → p np
r7: vp → v np		
l1: a → "I"	l2: v → "see"	l3: det → "a"
l4: n → "man"	l5: p→ "at"	l6: n→ "home"

Figure 6: Sample grammar

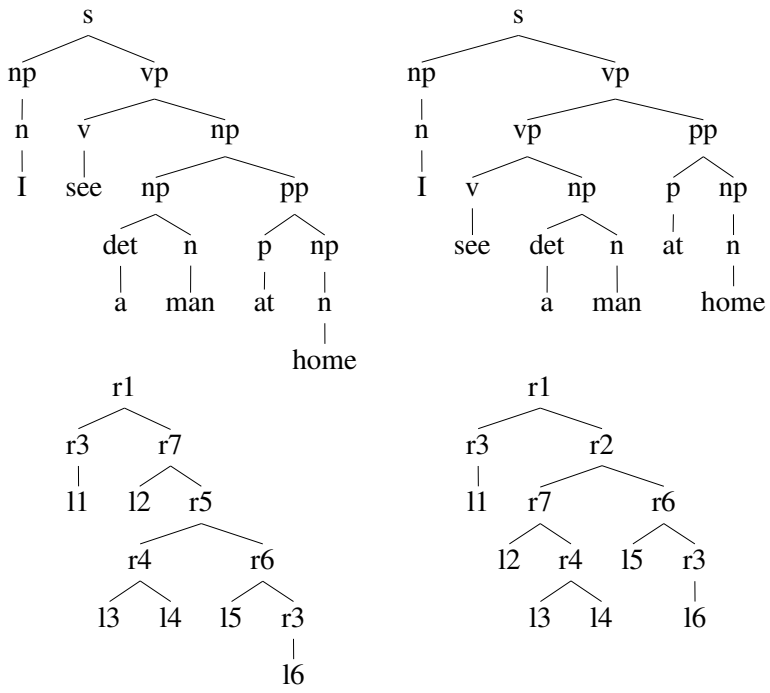


Figure 7: Two parse trees and corresponding derivation trees for *I see a man at home*

nt0	nt1	nt2	nt3	nt4	nt5	nt6
r1 r3 nt1 l1	r2 r7 r7 nt3 l2 nt4 l2 nt2	r4 l3 nt5	r6 l5 nt6	r5 r4 nt3 l3 nt5	l4	r3 l6

Figure 8: The parse forest of *I see a man at home.* The parse forest consists of a number of indexes associated with sets of partial derivation trees. The derivation trees are left-corner spines where all non-terminal nodes and the left-most terminal node are rule-names. Each of the other terminal nodes is labeled with an index, indicating that one of the partial derivation trees associated with that index can be plugged in here.

```
RECOVER-WITH-BEAM(start, indexes, b)
1  for each i in TOP-SORT(indexes)
2      do for each sub ∈ i.trees
3          do I_1 ... I_k ← indexes in sub
4              for each (t_1 ... t_k) ∈ I_1.best × ... × I_k.best
5                  do t ← sub
6                      for i ← 1 to k
7                          do SUBS(t, I_i, t_i)
8                      ADD(b,t,sub.best)
9  return best element of start.best
```

Figure 9: Algorithm RECOVER-WITH-BEAM

Parse recovery. In (Nederhof, 2003) two types of algorithm are discussed to find the best parse. The first type of algorithm is closely related to Dijkstra's algorithm to find the shortest path in a directed graph; its application to probabilistic context-free grammars is known as Knuth's algorithm. It consists of an agenda-driven, chart parsing algorithm in which the agenda is ordered in such a way that promising items are processed before others. This type of algorithm is applicable provided the scores are *superior*. This roughly implies that:

- the score of a parse is equal or lower than the score of each of its components
- if a component c_1 has a higher score than c_2, then it should be the case that all parses which contain c_1 have a higher score than all corresponding parses which have c_2 instead of c_1.

Both conditions are not applicable. The first condition is violated because the feature weights can be either positive or negative. The second condition is violated, in general, since we allow various non-local features.

The second type of algorithm discussed by Nederhof is closely related to the Viterbi algorithm, and to the DAG-SHORTEST-PATH algorithm as described in (Cormen *et al.*, 1990), as well as to the algorithm for finding the best parse in a parse forest presented in (Geman & Johnson, 2002). This type of algorithm works provided the second condition above applies. Our algorithm can be described as a generalization of the second type. The generalization consists in allowing the b best candidates for each component, to compensate for the effect of global features which violate condition 2 above.

The actual algorithm which recovers parse trees from the parse forest is given in figure 9. The algorithm maintains for each state in the search space only the b best candidates, where b is a small integer (the *beam*). If the beam is decreased, then we run a larger risk of missing the best parse (but the result will typically still be a relatively 'good' parse); if the beam is increased, then the amount of computation increases as well.

The algorithm constructs a number of parse trees on the basis of a set of indexes, the index associated with the top category *start*, and the beam b. The algorithm first topologically sorts the indexes, where an index i precedes j if a tree associated with i is required in a possible derivation tree for j (line 1).

beam	≤ 15			≤ 30			all		
	CA%	CPU	out	CA%	CPU	out	CA%	CPU	out
1	90.33	0.66	0	86.69	0.23	0	84.82	0.14	0
2	90.62	0.72	0	87.08	0.28	0	85.18	0.18	0
4	90.71	0.79	0	87.19	0.37	0	85.36	0.28	0
8	90.85	0.87	0	87.32	0.50	0	85.49	0.39	0
16	90.85	0.98	0	87.37	0.70	0	85.60	0.56	0
32	90.85	1.17	0	87.11	1.04	1	84.87	0.90	4
no beam	90.85	1	0	80.16	1	32	69.59	1	74

Table 1: Effect of beam-size on accuracy and efficiency of parse selection. Sentences from random 10% of Alpino Treebank. The left part of the table displays results for sentences up to 15 words; the central part for sentences up to 30 words; and the right part for all sentences. We normalize the parse selection times with respect to the variant without a beam (CPU=1), ignoring sentences for which the algorithm ran out of memory.

The algorithm then iterates over the indexes in this ordering, constructing larger derivation trees on the basis of derivation trees created earlier. To create a derivation tree for a specific index i, the algorithm iterates over all trees associated with i (line 2). In such a tree, there are a number of nodes labeled with an index. For each of these, the corresponding best derivation trees (discovered in a previous iteration) are substituted at the corresponding node (the algorithm iterates over all possible combinations in line 4). Then, a score is computed for each of the resulting trees and the tree is added if the score is good enough (line 8). This involves (not shown in the figure) mapping the derivation tree to a full parse tree, counting the occurrences of all features, multiplying these counts with the corresponding feature weights, and summing the results. If the new tree has a higher score than any of the best trees associated with i so far, then the tree is stored. The procedure ADD will add a given tree t to a set of trees *sub.best* if either there are less than b trees in *sub.best*, or t is better than at least one of the trees in *sub.best*. In the latter case, the worst scoring tree is removed from *sub.best*. Finally, the algorithm returns the best parse associated with the start node.

In table 1 the effect of various values for b is presented for a number of different treebanks. In the first columns, we list the results on a random sample of sentences from the treebank of up to fifteen words. In the next few columns, we list the result on a random sample of sentences from the treebank of up to thirty words. In the final columns, a random sample of the treebank is used without a restriction on sentence length. Per column, we list concept accuracy, CPU requirements, and the number of sentences for which the parser could not find an analysis due to memory limitations (in such cases the accuracy obviously is dropped too, since no correct result is constructed). As can be seen from the table, increasing the beam size slightly improves results, but for larger values memory problems cause a severe drop of accuracy. Also, the beam can remain rather small. This is probably due to the fact that most of our features are rather local in nature, as well as to the fact that the basic units of the parse forest are relatively large in size. Currently, Alpino uses $b = 4$ by default.

5 Error Mining

As we all know, hand-crafted linguistic descriptions such as those employed in Alpino, contain mistakes, and are incomplete. Grammar developers often construct sets of example sentences that the system should be able to parse correctly. If a sentence cannot be parsed, it is a clear sign that something is wrong. This technique only works in as far as the problems that might occur have been anticipated. More recently, tree-banks have become available, and we apply the parser to the sentences of the tree-bank and compare the resulting parse trees with the gold standard. Such techniques are limited, however, because tree-banks are relatively small. This is a serious problem, because the distribution of words is Zipfian (there are very many words that occur very infrequently), and the same appears to hold for syntactic constructions.

In (van Noord, 2004) an *error mining* technique is described which is very effective at automatically discovering systematic mistakes in a parser by using very large (but unannotated) corpora. The idea is very simple. We run the parser on a large set of sentences, and then analyze those sentences the parser cannot parse successfully. Depending on the nature of the parser, we define the notion 'successful parse' in different ways. For the Alpino parser, we consider all sentences for which the parser finds a single parse spanning the complete input to be successful. On the other hand, sentences for which the parser yields a sequence of partial parses are counted as unsuccessful.

The basic idea is to compare the frequency of words and word sequences in sentences that cannot be parsed successfully with the frequency of the same words and word sequences in unproblematic sentences. To compute the frequency of word sequences of arbitrary length for very large corpora, we proposed a new combination of suffix arrays and perfect hash finite automata.

The error mining technique is able to discover systematic problems which lead to parsing failure. This includes missing, incomplete and incorrect lexical entries and grammar rules. Clearly, problems which cause the parser to assign complete but incorrect parses cannot be discovered. Therefore, tree-banks and hand-crafted sets of example sentences remain important to discover problems of the latter type.

The *error mining* technique assumes we have available a large corpus of sentences. We run the parser on all sentences, and we note for which sentences the parser is successful. We define the *parsability* of a word w, $R(w)$, as the ratio of the number of times the word occurs in a sentence with a successful parse ($C(w|\text{OK})$) and the total number of sentences that this word occurs in ($C(w)$):

$$R(w) = \frac{C(w|\text{OK})}{C(w)}$$

Thus, if a word only occurs in sentences that cannot be parsed successfully, the parsability of that word is 0. On the other hand, if a word only occurs in sentences with a successful parse, its parsability is 1. If we have no reason to believe that a word is particularly easy or difficult, then we expect its parsability to be equal to the coverage of the parser (the proportion of sentences with a successful parse). If its parsability is (much) lower, then this indicates that something is wrong. In our case, the coverage of the parser lies between 91% and 95%. Yet, in our initial experiments, we found for *many* words parsability values that were much lower than that, including quite a number of words with parsability 0.

If a word has a parsability of 0, but its frequency is very low (say 1 or 2) then this might easily be due to chance. We therefore use a frequency cut-off (e.g. 5), and we ignore words which occur less often in sentences without a successful parse.

In many cases, the parsability of a word depends on its context. For instance, the Dutch word *via* is a preposition. Its parsability in a certain experiment was more than 90%. Yet, the parser was unable to parse sentences with the phrase *via via* which is an adverbial expression which means *via some complicated route*. For this reason, we generalize the parsability of a word to word sequences in a straightforward way. The parsability of a sequence is defined as:

$$R(w_i \dots w_j) = \frac{C(w_i \dots w_j|\text{OK})}{C(w_i \dots w_j)}$$

If a word sequence $w_i \dots w_j$ has a low parsability, then this might be because it is part of a difficult phrase. It might also be that part of the sequence is the culprit. In order that we focus on the relevant sequence, we consider a longer sequence $w_h \dots w_i \dots w_j \dots w_k$ only if its parsability is lower than the parsability of each of its sub-strings:

$$R(w_h \dots w_i \dots w_j \dots w_k) < R(w_i \dots w_j)$$

This is computed efficiently by considering the parsability of sequences in order of length (shorter sequences before longer ones).

We construct a parsability table, which is a list of n-grams sorted with respect to parsability. An n-gram is included in the parsability table, provided:

- its frequency in problematic parses is larger than the frequency cut-off
- its parsability is lower than the parsability of all of its sub-strings

We found that a parsability table provides a wealth of information about systematic problems in the grammar and lexicon, which is otherwise hard to obtain. Based on this, we have solved very many detailed problems. The problems included problems of tokenization, frequent spelling mistakes, many detailed problems in the lexicon (e.g., incorrect assignment of gender information), incomplete lexical entries (e.g., the lexicon only lists the noun reading of a word that can also be used as a verb), many missing valency frames, missing syntactic constructions (e.g., usage of R-pronouns in modifier position, various elliptical and gapping constructs, use of apposition with pronouns), missing multi-word expressions, including in particular missing multi-word named entities. In (van Noord, 2004) we showed that a systematic treatment of those problems increases coverage for a particular large corpus of a few million sentences from the Twente News corpus from about 91% to about 95%.

6 Accuracy

In this section, we present the current accuracy of the full system, incorporating the various techniques and components described earlier. In table 2, the accuracy of the full system on the Alpino treebank is given in the first row, using ten-fold cross-validation. The Alpino treebank is,

corpus	sents	length	F-sc	CA%
Alpino	7136	20	88.50	87.92
CLEF	1345	11	89.87	89.59
Trouw	1400	17	91.14	90.87

Table 2: Accuracy on development set and test sets for full system. The table lists the number of sentences, mean sentence length (in tokens), F-score and CA.

strictly speaking, by now a development set on which we optimized both the grammar and lexicon somewhat, as well as various tuning parameters for training not discussed here. Therefore, we provide results on two other test sets as well (using the model trained on the Alpino treebank). To this end, we annotated the first 1400 sentences of the Trouw 2001 newspaper, from the Twente News corpus. Another test set consists of all the Dutch questions from the CLEF Question Answering competition (all questions from CLEF 2003, CLEF 2004 and CLEF 2005). As can be seen in the table, the results on these test sets are even better. A potential explanation is the fact that these sets are easier because of shorter mean sentence length.

Another factor might be that the Trouw annotations are constructed fairly recently interactively on the basis of annotations suggested by Alpino. It might be that for some of the somewhat arbitrary choices that annotators have to make, the annotators more often prefer to allow the choice presented by Alpino. As a result the annotations might be somewhat biased to the outputs prefered by Alpino.

7 Alpino for Question Answering

Alpino is used as an important component of a recent Question Answering (QA) system for Dutch, called Joost (Bouma *et al.*, 2005b). Alpino is used both to analyze the question, as well as to analyze all potential answers.

Question Analysis. In experimenting with the use of Alpino for the analysis of questions, we found that the disambiguation component was not very accurate at preferring the appropriate reading for questions. In the Alpino treebank not many questions are present. For this reason, we created an additional annotated corpus of questions, and we also added a number of features for disambiguation that were thought to be particularly useful to disambiguate questions. The results of those efforts are listed in figure 3. The results illustrate that even if the system performed reasonably well for the analysis of questions, much better results can be obtained with some additional, focused, training.

Analysis of potential answers. Joost answers most questions by retrieving relevant paragraphs from a document collection, using keywords from the question. Next, potential answers are identified and ranked using a number of clues. Apart from obvious clues (i.e. Information Retrieval-score and the frequency with which the answer was found), we also use syntactic structure to identify and rank answer strings. A second strategy is based upon the observation that certain question types can be anticipated, and the corpus can be searched off-line for answers to such questions. Whereas previous approaches have used regular expressions to extract

system	sents	length	F-sc	CA%
Alpino standard	1345	11	89.87	89.59
Alpino tuned for QA	1345	11	96.34	96.23

Table 3: Parsing accuracy on CLEF questions. The table lists the number of sentences, mean sentence length (in tokens) F-score and Concept Accuracy. The first row lists parsing results for the standard system, the second row lists results in case the disambiguation component is trained on a dedicated Question Answering treebank and extended with a set of features that are useful for Question Answering.

the relevant relations, we use patterns of dependency relations.

In order that Joost has access to the full syntactic structure of potential answers (both for on-line and off-line search), we used the Alpino-system to parse the full text collection for the Dutch CLEF2005 Question Answering task. The text collection was tokenized (into 78 million words) and segmented into (4.1 million) sentences. Parsing this amount of text takes well over 500 CPU days. We once again used the Beowulf Linux cluster of the High-Performance Computing center of the University of Groningen to complete the process in about three weeks. The dependency trees are stored in XML. The XML files can be processed and searched in various ways, for instance, using XPATH and XSLT (Bouma & Kloosterman, 2002). Below, we refer to this automatically constructed treebank as the CLEF2005 treebank.

Several researchers have attempted to use syntactic information, and especially dependency relations, in Question Answering. One approach is to look for an exact match between dependency tuples derived from the question and those present in a potential answer (Katz & Lin, 2003; Litkowski, 2004). (Attardi *et al.*, 2002) and (Mollá & Gardiner, 2005) compute the match between question and answer using a metric which basically computes the overlap in dependency relations between the two. We have implemented a system in which dependency patterns derived from the question must be matched by equivalent dependency relations in a potential answer. Equivalences can be defined to account for the fact that in some cases we want a pattern to match a set of dependency relations that slightly differs from it, but nevertheless expresses the same semantic relation. This technique is described in more detail in (Bouma *et al.*, 2005a).

The Joost QA-system took part in CLEF2005 (monolingual QA). In this evaluation, the system found the correct answer for 49.5% of the questions, obtaining the best result for Dutch (out of 3 submissions), and the third result overall (out of 42 submissions).

8 Outlook

Our experience with the application of Alpino to large corpora in the context of training the POS-tagger, doing error mining, constructing dependency structures for question answering, made it apparent that for a wide variety of applications in information extraction, corpus linguistics and lexicography, much larger treebanks prove useful, even if these treebanks are not manually corrected.

For instance, the CLEF2005 treebank described above was employed both for on-line question answering, as well as off-line question answering. In the latter case, which is a case of information extraction, answers for typical questions are collected before the question is asked,

giving rise to tables consisting of e.g. capitals, causes of deaths, functions of person names, etc. (Bouma *et al.*, 2005a). It was shown (Jijkoun *et al.*, 2004) that the availability of (automatically constructed) syntactic annotation improves the quality of such tables considerably.

Large, automatically annotated corpora are also useful for applications in corpus linguistics. Bouma, Hendriks and Hoeksema (Bouma *et al.*, to appear) study a.o. the distribution of focus particles in prepositional phrases. Their corpus study on the basis of the CLEF2005 treebank revealed that such focus particles in fact are allowed (and fairly frequent) in Dutch, contradicting claims in theoretical linguistics. Similar techniques have been applied for the study of PP-fronting in Dutch (Bouma, 2004), the order of noun phrases with ditransitives (van der Beek, 2004), the distribution of determiner-less PPs (van der Beek, 2005), the distribution of weak pronouns, the distribution of impersonal pronouns as objects of prepositions, etc.

In a recent thesis (Moirón, 2005), Villada Moirón illustrates the usefulness of huge syntactically annotated corpora for various applications in semi-automatic lexicography, in particular aiming at the identification of support verb constructions and related fixed expressions in Dutch.

Very similar techniques have been integrated in other applications in information extraction and ontology building. Van der Plas and Bouma (van der Plas & Bouma, 2005b) apply vector-based methods to compute the semantic similarity of words, based on co-occurrence data extracted from the CLEF2005 treebank. The novel aspect of this work is that they define contexts with respect to the syntactic environment, rather than simple co-occurrence of words. Such syntactic contexts include verb-subject, verb-object, adjective-noun, elements of a coordination, elements in an apposition, and element in a prepositional complement. They show (van der Plas & Bouma, 2005a) that the acquired ontological information correlates with the information in Dutch EuroWordNet, and that the performance of question answering improves with such automatically acquired lexico-semantic information.

The quality of their results is perhaps best appreciated by looking at some examples. We provide examples involving named entities, since these do not require much knowledge of the Dutch language. For a given named entity, we list the 20 most similar words (or multi-word-units which are treated as words in the lexicon) according to the similarity measure computed over vectors containing the frequency of each syntactic context. A demo of their system is available via `http://www.let.rug.nl/~gosse/Sets/`.

One interesting application that we wish to investigate in the near future is the possibility to use such automatically constructed lexical semantic resources in parsing. The disambiguation component of Alpino already includes *lexical* features. We might hope that such features can be used, for instance, to learn that certain verbs, e.g., the verb *drink*, typically combine with objects such as *beer*, *wine*, *water*, *milk* .... A parse in which that verb combines with an unlikely object can then be assigned a lower score. However, in order that such lexical features can be learned effectively, and for a considerable number of words, we would need much larger manually annotated treebanks than are currently available. If, instead, we could map a word (e.g., *milk*) to its semantic class (e.g., *fluid*), we hope to be able to obtain the same effect, without extending the treebank.

BMW Volkswagen, Mercedes, Honda, Chrysler, Audi, Volvo, Ford, Toyota, Fiat, Peugeot, Opel, Mitsubishi, Renault, Mazda, Jaguar, General Motors, Rover, Nissan, VW, Porsche

Sony Matsushita, Toshiba, Time Warner, JVC, Hitachi, Nokia, Samsung, Motorola, Philips, Siemens, Apple, Canon, IBM, PolyGram, Thomson, Mitsubishi, Kodak, Pioneer, AT&T, Sharp

Hinault Kübler, Vermandel, Bruyère, Depredomme, Mottiat, Merckx, Depoorter, De Bruyne, Argentin, Schepers, Criquielion, Dierickx, Van Steenbergen, Kint, Bartali, Ockers, Coppi, Fignon, Kelly, De Vlaeminck

Agassi Sampras, Muster, Medvedev, Ivanisevic, Courier, Bruguera, Krajicek, Stich, Becker, Kafelnikov, Chang, Edberg, Rosset, Korda, McEnroe, Eltingh, Martin, Kro, Krickstein, Forget

The Times Independent, Daily Mail, The Independent, The Daily Telegraph, Ha'arets, Izvestija, Sunday Times, Los Angeles Times, Daily Mirror, The Sunday Times, The Guardian, News of the World, Bild, The Economist, The Sun, Financial Times, Süddeutsche Zeitung, International Herald Tribune, The Observer, Die Welt

Beatles Rolling Stones, Stones, John Lennon, Jimi Hendrix, Tina Turner, Bob Dylan, Elvis Presley, Michael Jackson, The Beatles, David Bowie, Prince, Genesis, Mick Jagger, The Who, Elton John, Barbra Streisand, Led Zeppelin, Eric Clapton, Diana Ross, Janis Joplin

Parijs Londen, Brussel, Moskou, Washington, Berlijn, New York, Rome, Madrid, Bonn, Wenen, Peking, Frankfurt, Athene, Tokio, München, Barcelona, Praag, Antwerpen, Stockholm, Tokyo

Grenoble Rouen, Saint Etienne, Pau, Saint-Etienne, Rennes, Marne-la-Vallée, Aix, Orléans, Toulouse, Montpellier, Amiens, Strasbourg, Lyon, Lens, Avignon, Clermont-Ferrand, Straatsburg, Caen, Bayonne, Limoges

Graham Greene Aldous Huxley, Georges Simenon, John Osborne, Martin Walser, H.G. Wells, Naguib Mahfouz, Martin Amis, Thomas Bernhard, Jan Jacob Slauerhoff, René Appel, Didier van Cauwelaert, René Stoute, Evelyn Waugh, Neil Gaiman, T.E. Lawrence, John Le Carré, Wole Soyinka, Louis Couperus, Bertolt Brecht, G. Parretti

Mozart Schubert, Beethoven, Ravel, Bach, Haydn, Brahms, Rossini, Mahler, Verdi, Wagner, Debussy, Chopin, Strawinsky, Sjostakowitsj, Tsjaikovski, Mendelssohn, Liszt, Strauss, Bruckner, Stravinsky

Picasso Matisse, Braque, Dali, Miro, Cézanne, Chagall, Giacometti, Klee, Van Gogh, Magritte, Miró, De Kooning, Kandinsky, Monet, Degas, Gauguin, Mondriaan, Picabia, Vuillard, Man Ray

Figure 10: Examples of some names with the 20 most similar words and multi-word-units according to the distributional similary measure, based on vectors of frequencies of syntactic contexts. The corpus consists of Dutch news-paper articles from 1994 and 1995.

Acknowledgements

This research was supported by the PIONIER project *Algorithms for Linguistic Processing* funded by *Nederlandse Organisatie voor Wetenschappelijk Onderzoek* (NWO). The work reported here is the result of a cooperation which included Gosse Bouma, Robert Malouf, Jan Daciuk, Leonoor van der Beek, Begona Villada Moirón, Tanja Gaustad, Mark-Jan Nederhof, Geert Kloosterman, Robbert Prins, Jori Mur, Lonneke van der Plas, Jörg Tiedemann, Peter Kleiweg and John Nerbonne. Their contributions are gratefully acknowledged. The examples in figure 10 are constructed using software by Lonneke van der Plas and Gosse Bouma.

References

ABNEY S. P. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, **23**, 597–618.

ALPINO (2002). The Alpino treebank 1.0. University of Groningen. CDROM; also available via http://www.let.rug.nl/˜vannoord/trees/.

ATTARDI G., CISTERNINO A., FORMICA F., SIMI M. & TOMMASI A. (2002). Piqasso: Pisa question answering system. In *Text REtrieval Conference (TREC) 2001 Proceedings*, p. 633–642, Gaithersburg, ML.

BOROS M., ECKERT W., GALLWITZ F., GÖRZ G., HANRIEDER G. & NIEMANN H. (1996). Towards understanding spontaneous speech: Word accuracy vs. concept accuracy. In *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP 96)*, Philadelphia.

BOUMA & KLOOSTERMAN (2002). Querying dependency treebanks in XML. In *Proceedings of the Third international conference on Language Resources and Evaluation (LREC)*, p. 1686–1691, Gran Canaria, Spain.

BOUMA G. (2004). Treebank evidence for the analysis of PP-fronting. In S. KUBLER, J. NIVRE, E. HINRICHS & H. WUNSCH, Eds., *Third Workshop on Treebanks and Linguistic Theories*, p. 15–26, Seminar für Sprachwissenschaft, Tübingen.

BOUMA G., HENDRIKS P. & HOEKSEMA J. (to appear). Focus particles inside prepositional phrases: A comparison between Dutch, English and German. *Journal of Comparative Germanic Linguistics*.

BOUMA G., MUR J. & VAN NOORD G. (2005a). Reasoning over dependency relations for QA. In F. BENAMARAH, M.-F. MOENS & P. SAINT-DIZIER, Eds., *Knowledge and Reasoning for Answering Questions*, p. 15–21. Workshop associated with IJCAI 05.

BOUMA G., MUR J., VAN NOORD G., VAN DER PLAS L. & TIEDEMANN J. (2005b). Question answering for Dutch using dependency relations. In *Proceedings of the CLEF2005 Workshop*.

BRISCOE T., CARROLL J., GRAHAM J. & COPESTAKE A. (2002). Relational evaluation schemes. In *Proceedings of the Beyond PARSEVAL Workshop at the 3rd International Conference on Language Resources and Evaluation*, p. 4–8, Las Palmas, Gran Canaria.

CARROLL J. & BRISCOE E. (1996). Apportioning development effort in a probabilistic LR parsing system through evaluation. In *Proceedings of the ACL SIGDAT Conference on Empirical Methods in Natural Language Processing*, p. 92–100, University of Pensylvania, Philadelphia, PA.

CHARNIAK E., CARROLL G., ADCOCK J., CASSANDRA A., GOTOH Y., KATZ J., LITTMAN M. & MCCANN J. (1996). Taggers for parsers. *Artificial Intelligence*, **85**(1-2).

CORMEN, LEISERSON & RIVEST (1990). *Introduction to Algorithms*. Cambridge Mass.: MIT Press.

DACIUK J. (2000). Finite state tools for natural language processing. In *Using Toolsets and Architectures to Build NLP Systems. Coling 2000 Workshop*, p. 34–37, Luxembourg: Centre Universitaire.

DEN BOOGAART P. C. U. (1975). *Woordfrequenties in geschreven en gesproken Nederlands*. Utrecht: Oosthoek, Scheltema & Holkema. Werkgroep Frequentie-onderzoek van het Nederlands.

GEMAN S. & JOHNSON M. (2002). Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the ACL*.

JELINEK F. (1998). *Statistical Methods for Speech Recognition*. MIT Press.

JIJKOUN V., MUR J. & DE RIJKE M. (2004). Information extraction for question answering: Improving recall through syntactic patterns. In *COLING 2004*, Geneva.

JOHNSON M., GEMAN S., CANON S., CHI Z. & RIEZLER S. (1999). Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the ACL*.

KATZ B. & LIN J. (2003). Selectively using relations to improve precision in question answering. In *Proceedings of the workshop on Natural Language Processing for Question Answering (EACL 2003)*, p. 43–50, Budapest: EACL.

KIEFER B., KRIEGER H.-U., CARROLL J. & MALOUF R. (1999). A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics*, p. 473–480, College Park, MD.

LITKOWSKI K. C. (2004). Use of metadata for question answering and novelty tasks. In E. M. VOORHEES & L. P. BUCKLAND, Eds., *Proceedings of the eleventh Text Retrieval Conference (TREC 2003)*, p. 161–170, Gaithersburg, MD.

MALOUF R. & VAN NOORD G. (2004). Wide coverage parsing with stochastic attribute value grammars. In *Beyond Shallow Analyses - Formalisms and statistical modeling for deep analyses*, Hainan China: IJCNLP.

MANNING C. D. & SCHÜTZE H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge Mass.

MIYAO Y. & TSUJII J. (2002). Maximum entropy estimation for feature forests. In *Proceedings of Human Language Technology Conference (HLT 2002)*.

MOIRÓN B. V. (2005). *Data-driven identification of fixed expressions and their modifiability*. PhD thesis, University of Groningen.

MOLLÁ D. & GARDINER M. (2005). Answerfinder - question answering by combining lexical, syntactic and semantic information. In *Australasian Language Technology Workshop (ALTW) 2004*, Sydney.

NEDERHOF M.-J. (2003). Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, **29**(1), 135–143.

OOSTDIJK N. (2000). The Spoken Dutch Corpus: Overview and first evaluation. In *Proceedings of Second International Conference on Language Resources and Evaluation (LREC)*, p. 887–894.

OSBORNE M. (2000). Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the Eighteenth International Conference on Computational Linguistics (COLING 2000)*.

PRINS R. (2005). *Finite-State Pre-Processing for Natural Language Analysis*. PhD thesis, University of Groningen.

RASPE R. E. (1786). *Baron Munchausen's narrative of his marvellous travels and campaigns in Russia*. Oxford.

RIEZLER S., KING T. H., KAPLAN R. M., CROUCH R., MAXWELL J. T. & JOHNSON M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the ACL*.

VAN DER BEEK L. (2004). Argument order alternations in Dutch. In M. BUTT & T. H. KING, Eds., *Proceedings of the LFG'04 Conference*: CSLI Publications.

VAN DER BEEK L. (2005). The extraction of Dutch determinerless PPs. In *Proceedings of the 2nd ACL-SIGSEM Workshop on the Linguistic Dimensions of Prepositions and their Use in Computational Linguistics Formalisms and Applications*, University of Essex, Colchester.

VAN DER BEEK L., BOUMA G., MALOUF R. & VAN NOORD G. (2002). The Alpino dependency treebank. In *Computational Linguistics in the Netherlands*.

VAN DER PLAS L. & BOUMA G. (2005a). Automatic acquisition of lexico-semantic knowledge for QA. In *Proceedings of the IJCNLP workshop on Ontologies and Lexical Resources*.

VAN DER PLAS L. & BOUMA G. (2005b). Syntactic contexts for finding semantically related words. In T. VAN DER WOUDEN, M. POSS, H. RECKMAN & C. CREMERS, Eds., *Computational Linguistics in the Netherlands 2004. Selected Papers from the fifteenth CLIN meeting*: LOT, Netherlands Graduate School of Linguistics, Utrecht.

VAN NOORD G. (1997). An efficient implementation of the head corner parser. *Computational Linguistics*, **23**(3), 425–456.

VAN NOORD G. (2001). Robust parsing of word graphs. In J.-C. JUNQUA & G. VAN NOORD, Eds., *Robustness in Language and Speech Technology*. Dordrecht: Kluwer Academic Publishers.

VAN NOORD G. (2004). Error mining for wide-coverage grammar engineering. In *ACL2004*, Barcelona: ACL.

VAN NOORD G. & MALOUF R. (2005). Wide coverage parsing with stochastic attribute value grammars. Draft available from the authors.

VOUTILAINEN A. (1998). Does tagging help parsing? A case study on finite state parsing. In *Finite-state Methods in Natural Language Processing*, Ankara.

WAUSCHKUHN O. (1995). The influence of tagging on the results of partial parsing in German corpora. In *Fourth International Workshop on Parsing Technologies*, p. 260–270, Prague/Karlovy Vary, Czech Republic.