Reinforcing Parser Preferences through Tagging

Robbert Prins — Gertjan van Noord

Alfa-Informatica
University of Groningen
PO Box 716
NL 9700 AS Groningen Netherlands
r.p.prins@let.rug.nl

ABSTRACT. *Lexical ambiguity is an important source of inefficiency for wide-coverage HPSG parsing. In this paper, we propose a lexical analysis filter which removes unlikely lexical categories. The filter is implemented as a straightforward HMM n-gram POS-tagger, which computes the 'a posteriori' probability of each lexical category. A lexical category is removed if a competing lexical category is sufficiently more likely. The novel aspect of our approach is the fact that the tagger is trained on the output of the parser itself; therefore there is no need for hand-annotated material. Use of this filter increases the speed of the parser considerably, and in addition gives rise to an improvement in parsing accuracy.*

RÉSUMÉ. *L'ambiguïté lexicale est une source importante de l'inefficacité de l'analyse syntaxique HPSG à large couverture. Dans cette contribution, nous proposons un filtre analyseur lexical qui élimine des catégories lexicales improbables. Le filtre est implémenté comme un étiqueteur markovien (HMM) n-gramme standard, qui calcule la probabilité 'a posteriori' de chaque catégorie lexicale. Une catégorie lexicale est rejetée, quand celui-ci est en concurrence avec des catégories lexicales qui sont suffisamment plus probables. La nouveauté de l'approche exposée ici consiste à entraîner l'étiqueteur sur la sortie de l'analyseur lui-même ; par conséquence, on n'a pas besoin de corpus étiquetés manuellement. L'emploi de ce filtre augmente considérablement la rapidité de l'analyseur, et en plus, en améliore la précision.*

KEYWORDS: *parsing, lexical ambiguity, part-of-speech tagging, HMM.*

MOTS-CLÉS : *analyse syntaxique, ambiguïté lexicale, étiquetage morpho-syntaxique, HMM.*

1. Introduction

Full parsing of unrestricted text on the basis of a wide-coverage computational HPSG grammar remains a challenge. In our recent experience in the development of the Alpino system, a parsing system based on a wide-coverage HPSG for Dutch, we found that even in the presence of various sophisticated chart parsing and ambiguity packing techniques, lexical ambiguity in particular has an important 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 [KIE 99]. 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.

In this paper we extend this filtering component using a part-of-speech (POS) filter. 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 [WAU 95], [CHA 96], [VOU 98]. 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*.

For this approach to be feasible, the parser needs to be able to distinguish between competing parses. In our case, the Alpino parser is complemented by a maximum entropy disambiguation component. The disambiguation component is discussed briefly in section 2.

Secondly, the HPSG for Dutch that is implemented in Alpino is heavily lexicalist. This implies that (especially) verbs 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 [CAR 96]). 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 increases.

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 lexical dependency relations, similar to the proposal in [CAR 98b]. This evaluation measure presupposes the availability of a treebank, but is expected to reflect much better the accuracy of the system. In particular, as we will argue below, the coverage measure appears to be a very misleading evaluation metric for parse accuracy.

In the following section we describe the Alpino wide-coverage parser of Dutch, with which we performed our experiments. In section 3 we give a description of the trigram HMM tagger. In section 4 we show how using this tagger as a filter in Alpino improves the parser's performance, and in section 5 we consider ideas for future work.

2. Alpino: Wide-coverage Parsing of Dutch

Alpino is a wide-coverage computational analyzer of Dutch which aims at accurate full parsing of unrestricted text. The system is described in more detail in [BOU 01b]. The grammar produces dependency structures, thus providing a reasonably abstract and theory-neutral level of linguistic representation. The dependency relations encoded in the dependency structures have been used to develop and evaluate both hand-coded and statistical disambiguation methods.

2.1. *Grammar*

The Alpino grammar is an extension of the successful[1] OVIS grammar [NOO 99], a lexicalized grammar in the tradition of HPSG [POL 94].

The grammar formalism is carefully designed to allow linguistically sophisticated analyses as well as efficient and robust processing. In contrast to earlier work on HPSG, grammar rules in Alpino are relatively detailed. However, as pointed out in [SAG 97],

1. The OVIS grammar was part of the language understanding component of a spoken dialogue system for public transport information. In a formal evaluation, it was shown to perform much better than a competing language understanding component based on data-oriented parsing [ZAN 99].

by organizing rules in an inheritance hierarchy, the relevant linguistic generalizations can still be captured. The Alpino grammar currently contains over 350 rules, defined in terms of a few general rule structures and principles (almost all rules are defined in terms of a set of 15 different structures, which make use of about 10 different principles). The grammar covers the basic constructions of Dutch (including main and subordinate clauses, (indirect) questions, imperatives, (free) relative clauses, a wide range of verbal and nominal complementation and modification patterns, and coordination) as well as a wide variety of more idiosyncratic constructions (appositions, verb-particle constructions, PP's including a particle, NP's modified by an adverb, punctuation, etc.). The lexicon contains definitions for various nominal types (nouns with various complementation patterns, proper names, pronouns, temporal nouns, deverbalized nouns), various complementizer, determiner, and adverb types, adjectives, and about 100 verbal subcategorization types. Lexical generalizations are captured by organizing these lexical definitions in an inheritance network. The lexicon contains descriptions for more than 100,000 word forms.

The formalism supports the use of recursive constraints over feature-structures (using delayed evaluation, [NOO 94]). This allowed us to incorporate an analysis of cross-serial dependencies based on argument-inheritance [BOU 98] and a trace-less account of extraction along the lines of [BOU 01a].

The Alpino grammar produces dependency structures compatible with the CGN-guidelines. Within the CGN-project [OOS 00], guidelines have been developed for syntactic annotation of spoken Dutch [MOO 00], using dependency structures similar to those used for the German Negra corpus [SKU 97]. The CGN-guidelines are available from the CGN-website: `http://lands.let.kun.nl/cgn`.

Below, some of the experiments make use of the Alpino Treebank [Alp02]. The Alpino Treebank contains hand-corrected syntactic annotations compatible with the CGN-guidelines for the newspaper (`cdbl`) part of the Eindhoven corpus [BOO 75]. The annotation consists of dependency structures, rather than phrase-structure trees.

2.2. *Robust Parsing*

The initial design and implementation of the Alpino parser is inherited from the system described in [NOO 97], [NOO 99] and [NOO 01]. However, a number of improvements have been implemented which are described below. The construction of a dependency structure proceeds in a number of steps. The first step consists of lexical analysis. In the second step a parse forest is constructed. The third step consists of the selection of the best parse from the parse forest.

2.2.1. *Lexical Analysis*

The lexicon associates a word or a sequence of words with one or more *lexical categories*. Such lexical categories contain information such as part-of-speech, inflection as well as a subcategorization frame. For verbs, the lexicon typically hypothesizes many different lexical categories, differing mainly in the subcategorization frame.

For the words in the following sentence (2), for instance, the lexicon produces initially a total of 199 lexical categories:

(2) Mercedes zou haar nieuwe model gisteren hebben aangekondigd
Mercedes should her new model yesterday have announced
Mercedes should have announced its new model yesterday

Many of those lexical categories are obviously wrong. For example, one of the lexical categories for `hebben` is `verb(hebben,pl,part_sbar_transitive(door))`. This lexical category indicates a finite plural verb which requires a separable prefix `door`, and which subcategorizes for an SBAR complement. Since `door` does not occur anywhere in sentence (2), this lexical category will not be useful for this sentence. A filter containing a number of hand-written rules has been implemented which checks that such simple co-occurrence conditions hold. For sentence (2), the filter removes 173 lexical categories. The remaining 26 lexical categories are input to the HMM tagger described below. The tagger (using the default settings) will remove 8 of these. After the filter has been applied, feature structures are associated with each of the remaining 18 tags. Often, a single tag is mapped to multiple feature structures. The remaining 18 tags give rise to 81 feature structures.

2.2.2. *Creating Parse Forests*

The Alpino parser takes the set of feature structures found during lexical analysis as its input, and constructs a *parse forest*: a compact representation of all parse trees. The Alpino parser is a left-corner parser with selective memoization and goal-weakening. It is a variant of the parsers described in [NOO 97]. We generalized some of the techniques described there to take into account relational constraints, which are delayed until sufficiently instantiated [NOO 94].

2.2.3. *Robustness*

As described in [NOO 99] and [NOO 01], the parser can be instructed to find all occurrences of the start category *anywhere in the input*. In case the parser cannot find an instance of the start category from the beginning of the sentence to the end, then the parser produces parse trees for chunks of the input. A best-first search procedure then picks out the best sequence of such chunks, generally preferring larger chunks over shorter ones, preferring connected paths over paths with uncovered words, and taking into account the scores assigned to the chunks by the disambiguation component (described in the following paragraph).

2.2.4. *Unpacking and Parse Selection*

The motivation to construct a parse forest is efficiency: the number of parse trees for a given sentence can be enormous. In addition to this, in most applications the objective will not be to obtain *all* parse trees, but rather the *best* parse tree. Thus, the final component of the parser consists of a procedure to select the best parse tree from the parse forest.

In order to select the best parse tree from a parse forest, we assume a parse evaluation function which assigns a score to each parse. In [BOU 01b] some initial experiments with a variety of parse evaluation functions are described. In the experiments discussed here, the parse evaluation function consisted of a log-linear model.

Log-linear models were introduced to natural language processing by [BER 96] and [Del 97], and applied to stochastic constraint-based grammars by [ABN 97] and [JOH 99]. Given a conditional log-linear model, the probability of a sentence x having the parse y is:

$$p(y|x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

Here, each $f_i(x, y)$ is a property function which will return the number of times a specific property i occurs in parse y of sentence x. Each property function has an associated weight λ_i (the weights are determined in training). The partition function $Z(x)$ will be the same for every parse of a given sentence and can be ignored, so the score for a parse is simply the weighted sum of the property functions $f_i(x, y)$.

In the log-linear model employed in our parser, we employed several types of features corresponding to grammar rules as well as some more idiosyncratic features indicating complementation / modification, long / short distance dependencies etc. The model was trained on the sentences of up to 40 words of the Alpino treebank (about 6000 sentences from the newspaper part of the Eindhoven corpus) [2].

A naive algorithm constructs all possible parse trees, assigns each one a score, and then selects the best one. Since it is too inefficient to construct all parse trees, we have implemented the algorithm which computes parse trees from the parse forest as a best-first search. This requires that the parse evaluation function is extended to partial parse trees. We implemented a variant of a best-first search algorithm in such a way that for each state in the search space, we maintain 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 too. [3]

2. The training data did not contain the first 220 sentences which are used for the evaluation described in section 4.1.
3. Note that this procedure differs from best-first parsing (e.g. [CAR 98a]) since in our case only the parse selection phase is best-first; the construction of the parse-forest finds all parses.

3. Using a POS-tagger as a Filter

3.1. *Mapping Lexical Categories to POS-tags*

As indicated earlier, the wide coverage lexicon for Dutch that we worked with makes many detailed distinctions. Therefore, there are many different lexical categories: more than 18,000. In order to use this tagset in a POS-tagger, the lexical categories are mapped to a (smaller) set of lexical category classes, by ignoring some of the information present in lexical categories (in particular subcategorization information). In the experiments described here, there were 1365 lexical category classes.

The HMM tagger described below finds the best tag for each position in the string, and then removes competing tags for the same position under certain conditions. The Alpino lexicon sometimes assigns lexical categories to a sequence of words in the input. For example, the three words

(3) met betrekking tot
with respect to

are analyzed as a single preposition. In order to keep the architecture of the tagger simple, the words that make up such multi-word-units are mapped to separate tags. If the sequence of words $w_1 \dots w_n$ is assigned category c, then this category is removed, and instead for each j, $1 \leq j \leq n$, we assign a new category (j, c) to word w_j. In example (3) the category $(1, \text{preposition})$ is assigned to `met`, $(2, \text{preposition})$ is assigned to `betrekking` and $(3, \text{preposition})$ is assigned to `tot`. Due to this expansion, the number of lexical category classes increases (in the experiments below) to 2392. The Alpino lexicon also assigns lexical categories to sequences of words in the case of named entities (proper names, temporal expressions) and multi-word-units (often expressions from other languages) such as *à priori*, *up to date*, *credit card*, etc.

3.2. *The HMM Tagger*

We implemented a variant of the standard trigram HMM tagger, described e.g. in chapter 10.2 of [MAN 99]: 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 [JEL 98] in the context of speech recognition. The same technique is described in [CHA 96]. 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 filtering accuracy and remaining ambiguity.

3.3. *Smoothing*

In order to estimate the probability of a (potentially unseen) trigram, we take into account the probability of the bigram and unigram as well. Thus, the trigram model takes into account lower order models, assigning weights to each of the models to express their relative importance. This idea, known as linear interpolation, is a well-known method for combining models (see for instance [MAN 99]). By adjusting the weights, which together sum up to one, we can put the emphasis on the model that uses a greater context, while at the same time not ignoring the information provided by models that are simpler, but for which more data is available. We interpolate a uni-, bi- and trigram model by means of the following formula (where λ_1, λ_2 and λ_3 are the three respective weights, and $P^*(X)$ is the probability of some n-gram X computed directly from the training data frequency counts):

$$P(t_3|t_1, t_2) = \lambda_3 P^*(t_3|t_1, t_2) + \lambda_2 P^*(t_3|t_2) + \lambda_1 P^*(t_3)$$

The weights are computed using the notion of n-gram *diversity*, an idea described by Collins ([COL 99], borrowing from [BIK 97]). In order to compute the weight for a trigram $t_1 t_2 t_3$, we take the diversity and frequency of the bigram $t_1 t_2$ into account.

The diversity of an n-gram is the number of different tags that appear in the position following this n-gram in the training data. If the trigram starts with a low-diversity bigram, the weight associated with the trigram can be large: the probability of the bigram being followed by an unexpected tag is small, and if this does happen, it is likely to be a significant event. If on the other hand the bigram is of a high diversity, the trigram should receive a small weight.

When we have the count and diversity figures available for bigrams, the weight λ_3 for a trigram $t_1t_2t_3$ can be derived as follows, using a constant c to regulate the importance of diversity in this computation:

$$\lambda_3 = \begin{cases} 0 & \text{if count}(t_1t_2) = 0 \\ \frac{\text{count}(t_1t_2)}{\text{count}(t_1t_2)+c\times\text{diversity}(t_1t_2)} & \text{if count}(t_1t_2) > 0 \end{cases}$$

We found that differences in the value used for c resulted in only small variations in performance. In the experiments described in following sections a value of c=7 was used.

3.4. ***Training the Tagger***

The probabilities which are used in the HMM tagger are directly estimated from a labeled training corpus. 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. [4]

Of course, the training set thus produced 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. 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, and sentences for which the parser is not able to find a full parse. Under these conditions,

4. A reviewer suggests to use the Alpino Treebank corpus to train a baseline version of the tagger — however this is not possible because the treebank does not contain part-of-speech tags. Although Dutch corpora have been manually tagged in the past, we cannot use manually annotated material because the tag sets that were used in the past are quite different from the set of lexical category classes used in the Alpino lexicon.

model	accuracy (%)
unigram	87.23
bigram	94.03
trigram	94.69

Table 1. *Stand-alone tag filter accuracy using different models*

parsing a week of newspaper text takes somewhere between 15 and 20 hours of CPU time on standard hardware.[5]

3.5. *Stand-alone Results*

We experimented with the use of uni-, bi- and trigram models. The models were trained on about 45 months ($\sim$2 million sentences, $\sim$24 million words) of annotated text from Dutch daily newspapers, and the filter was applied in a stand-alone setup to a random selection of 10% of the sentences of the Alpino Treebank which contain at most 40 words. Thus, these 604 sentences were parsed by Alpino, and for each sentence the sequence of lexical category classes used by what the parser believed to be the best parse is deemed to be the gold standard.

In table 1 the accuracy is listed for the bigram and trigram model; as a baseline we also provide the unigram model. These percentages are obtained if for each position in the input only a single tag is allowed to survive.

In figure 1 we display accuracy levels for different amounts of ambiguity after filtering. The different levels of ambiguity are the result of using different threshold settings in the filter; a lower threshold means more tags will be considered bad and be removed. If the threshold value increases, then accuracy and ambiguity approach 100% (at which point all tags survive, including the correct ones).

The accuracy of the trigram model is lower than one might expect. In the literature, HMM trigram taggers yield accuracy levels of more than 96%. Of course, these results are hard to compare because of differences in the tag set and corpora that were used. For instance, the LOB corpus is tagged with a tagset consisting of 170 different tags [GAR 87]. The Wall Street Journal corpus is tagged with a tagset of only 48 tags [MAR 93]. The most important difference, however, is the nature of the test data in the current setup. It turns out that in many cases where the tagger and the test set disagree, the tagger actually improves upon the test set. An example illustrates this; consider the utterance:

(4)	... van	die	jongen	bij	de	doodskist...
	... of	the	young-animals[pl]/boy[sg]	near	the	coffin...

5. We heavily exploited a cluster of about 120 Linux PCs, at the High Performance Computing Center of the University of Groningen.

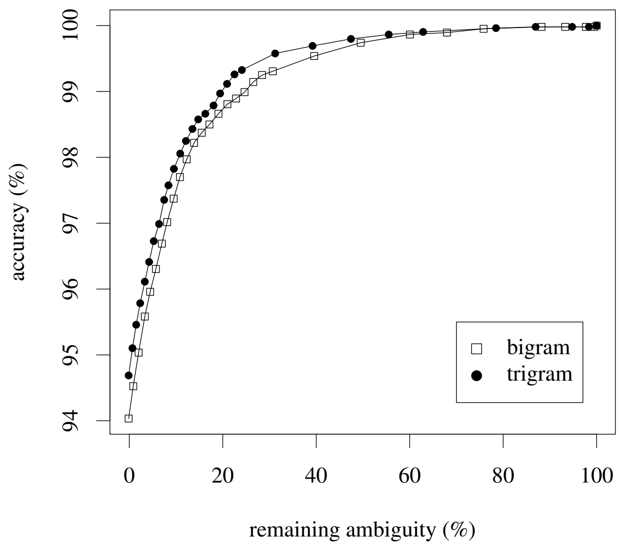

Figure 1. *Stand-alone tag filter accuracy versus ambiguity for bigram and trigram model, using different threshold settings ranging from τ=0 to τ=25. The model is trained on 45 months of newspaper data*

The Dutch word *jongen* is ambiguous between a singular noun reading *boy* and a plural noun reading *young-animals*. The first reading is much more frequent than the second reading. For some reason, the tag corresponding to the (infrequent) second reading was used by the parser in its best parse of the sentence and as a consequence this tag ended up in the test set. The HMM model quite rightly learned a preference for the frequent reading and assigned the correct tag.

3.6. *Larger Amounts of Training Data*

Given that the training data is produced automatically, the most straightforward manner of improving the tagger might be to use larger amounts of training data. In a typical language modeling situation, this is a severely limited solution since the data has to be manually annotated first. We experimented with using increasing amounts of training data. The results are shown in figure 2. To be able to contrast filtering accuracy with the number of months of training data, the filter was set to remove all but the best scoring tag for each word, thereby removing all ambiguity.

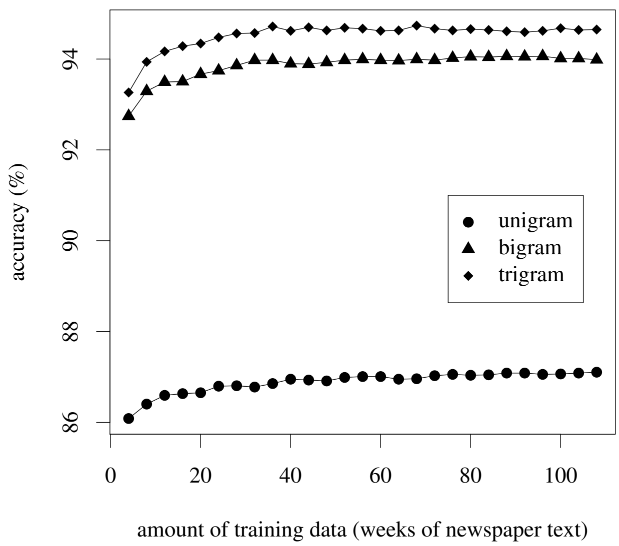

Figure 2. *Stand-alone tag filter accuracy using increasing amounts of training data and removing all but the single best tag for each word*

This graph shows that the n-gram model performs better when trained on more data, the trigram model gaining relatively more than the bigram model. It must be noted though that after about 40 weeks of training data the accuracy does not improve significantly anymore.

4. Using the Filter in the Parser

4.1. *Experimental Results*

The results so far have been stand-alone filtering results. As the filter is meant to be used to disambiguate the lexical analysis of the Alpino parser, we will now present parsing results and show how these are improved by the incorporation of the filter.

The filter was trained on more than 40 months of newspaper data annotated by the parser. Next, the parser was run on the first 220 sentences of the `cdbl` corpus, which is the newspaper part of the Eindhoven corpus; syntactic annotations of these sentences are available as part of the Alpino Treebank [Alp02]. From those sentences, four sentences were removed due to the fact that the parser *without the tag filter* ran

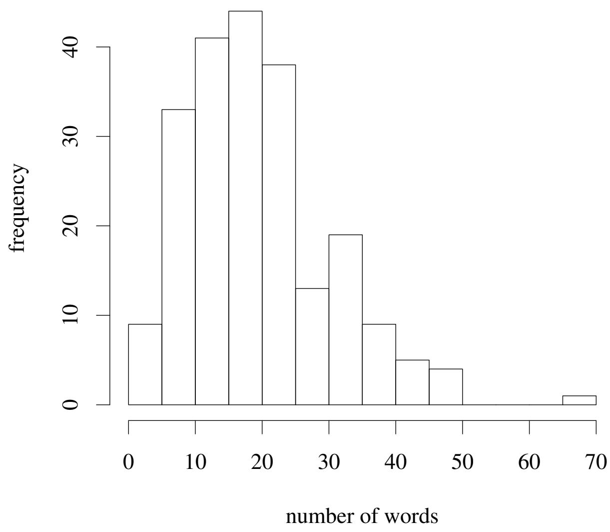

Figure 3. *Distribution of sentence length in the Alpino test set (in number of words)*

out of memory. The test set thus contains 216 sentences (4295 words). In figure 3 we provide a histogram of the distribution of sentence lengths.

Figure 4 plots the 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, leading to very fast parsing (as shown on the left hand side of the graph). Higher accuracy can be attained by using a higher threshold, removing a smaller number of tags and at the cost of a decrease in speed.

It becomes clear from this graph that use of the filter leads to an 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 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 5 we display the differences in efficiency in a different 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 ac-

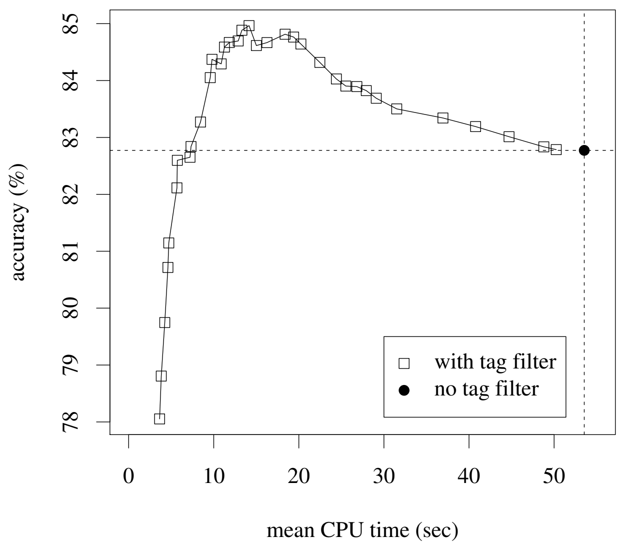

Figure 4. *Parsing results in terms of parser accuracy and mean CPU time, using the filter with different threshold settings ranging from τ=0 to τ=15, and using no filter*

curacy (cf. figure 4). 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.

4.2. *Discussion*

It is perhaps surprising that the use of the filter can actually *improve* the accuracy of the overall parser. One important reason is related to the strategy of the parser described earlier. The parser always prefers a single parse over a sequence of partial parses. Although this strategy implements a fairly natural tendency, it turns out that in the context of very unlikely lexical categories the strategy implies that very unlikely full parses are constructed e.g. for elliptical utterances. As an example, consider:

(5) In Amsterdam vanavond
cash[V+imp]/in[Prep] Amsterdam tonight
Cash Amsterdam tonight!
In Amsterdam - tonight

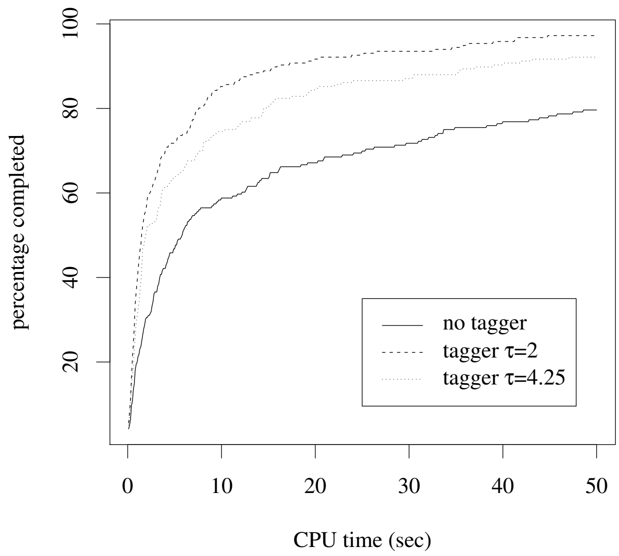

Figure 5. *Comparison of proportion of sentences that receive a parse within given amount of CPU time*

This sentence is naturally analyzed as an elliptical utterance, consisting of two parts: a prepositional phrase `in Amsterdam` followed by an adverbial. However, the word `in` can also be used as a verb. In this particular case, the parser can create a full parse in which `in` is indeed analyzed as an imperative verb form. In the integrated system, the POS-filter will filter out the possibility that `in` is a verb. As a result, no full parse is possible. In this particular case, the resulting parse (consisting of two parts) will be better than the unlikely single parse. In many cases, improvements of the integrated parser are due to similar phenomena.

The *coverage* of a parser can be defined as the proportion of sentences which receive a full parse. It can be argued that coverage is actually not a very useful metric for parser evaluation. The fact that the parser finds a full parse for a given sentence is only relevant if that parse is in some sense *correct*. But one important reason that the use of a lexical analysis filter improves parsing accuracy is given by the *reduction* of coverage. If the parser has many different lexical categories to choose from, then often it is able to find a full parse (albeit a ridiculous one). However, if the parser is forced to work with a limited set of lexical categories it becomes much harder to find such a full parse. In the majority of cases, however, the resulting partial parse is much closer to the gold standard. In figure 6 coverage and accuracy are plotted for different threshold settings.

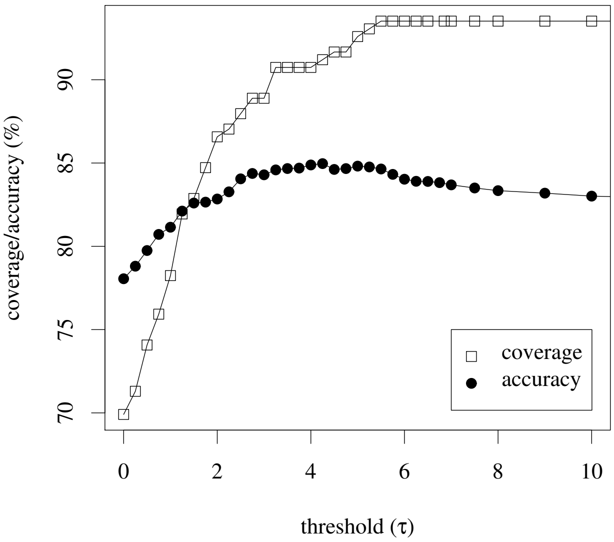

Figure 6. *Coverage (proportion of sentences that receive a full parse) and accuracy for different threshold settings*

5. Conclusions and Future Work

In this article we have proposed a simple technique to incorporate an n-gram POS-tagger in the lexical analysis phase of a wide-coverage parser. Since the POS-tagger does not need any human annotated data, the proposal is easily adaptable to other parsing systems. In our experiments, we found that the use of the tagger greatly reduced parsing times, and in addition gave rise to an increase of parsing *accuracy*. We showed that the accuracy increase was at least partly caused by the specific approach to robustness implemented in our system. More generally, the effect on accuracy of the incorporation of a POS-tagger as proposed in this article is dependent on the quality of the baseline system. We believe, however, that the positive effect on parsing efficiency is less dependent on the particularities of the baseline system. If the technique proposed in this article is applied to other wide-coverage parsing systems, we do expect important improvements in parsing efficiency.

In general, a model such as an HMM can be enriched by including more information in a single state. For example, in the case of n-gram models a single state could be made to represent three, instead of two, previous tags. The already small difference between the performance of the bigram and the trigram tagger (see figure 1) suggest that this is not likely to give much improvement. Perhaps a better way of extending the states is by adding information that is more linguistically informed. This is feasible, because during the construction of training material, the full parse tree is available.

Based on frequent errors made by the filter we have experimented with adding specific contextual information that humans would find helpful in making the relevant distinctions. So far, this gives a small improvement in accuracy; the question remains how much and what kind of contextual information can be added effectively. Since there is no bound on the amount of training material that we can produce for the tagger, sparse data may be less of a problem in our setup.

Acknowledgements

This research was carried out within the framework of the PIONIER Project *Algorithms for Linguistic Processing*, funded by NWO (Dutch Organization for Scientific Research) and the University of Groningen. The article is an extended and revised version of [PRI 01]. We would like to thank Mark-Jan Nederhof and the anonymous reviewers for helpful comments.

6. References

[ABN 97] ABNEY S. P., "Stochastic attribute-value grammars", *Computational Linguistics*, vol. 23, 1997, p. 597–618.

[Alp02] "The Alpino Treebank 1.0", University of Groningen, 11 2002, CDROM; also available via http://www.let.rug.nl/˜vannoord/trees/.

[BER 96] BERGER A., DELLA PIETRA S., DELLA PIETRA V., "A maximum entropy approach to natural language processing", *Computational Linguistics*, vol. 22, num. 1, 1996, p. 39–72.

[BIK 97] BIKEL D. M., MILLER S., SCHWARTZ R., WEISCHEDEL R., "Nymble: a high-performance learning name-finder", *Proceedings of ANLP-97*, 1997, p. 194–201.

[BOO 75] DEN BOOGAART P. C. U., *Woordfrequenties in geschreven en gesproken Nederlands*, Oosthoek, Scheltema & Holkema, Utrecht, 1975, Werkgroep Frequentie-onderzoek van het Nederlands.

[BOU 98] BOUMA G., VAN NOORD G., "Word order constraints on verb clusters in German and Dutch", HINRICHS E., NAKAZAWA T., KATHOL A., Eds., *Complex Predicates in Nonderivational Syntax*, p. 43–72, Academic Press, New York, 1998.

[BOU 01a] BOUMA G., MALOUF R., SAG I., "Satisfying Constraints on Adjunction and Extraction", *Natural Language and Linguistic Theory*, vol. 19, 2001, p. 1–65.

[BOU 01b] BOUMA G., VAN NOORD G., MALOUF R., "Wide Coverage Computational Analysis of Dutch", DAELEMANS W., SIMA'AN K., VEENSTRA J., ZAVREL J., Eds., *Computational Linguistics in the Netherlands, CLIN 2000*, Amsterdam, 2001, Rodopi, p. 45–59.

[CAR 96] CARROLL J., BRISCOE E., "Apportioning Development Effort in a Probabilistic LR Parsing System through Evaluation", *Proceedings of the ACL SIGDAT Conference on Empirical Methods in Natural Language Processing*, University of Pennsylvania, Philadelphia, PA, 1996, p. 92–100.

[CAR 98a] CARABALLO S. A., CHARNIAK E., "New Figures of merit for best-first probabilistic chart parsing", *Computational Linguistics*, vol. 24, num. 2, 1998.

[CAR 98b] CARROLL J., BRISCOE T., SANFILIPPO A., "Parser Evaluation: A Survey and a New Proposal", *Proceedings of the first International Conference on Language Resources and Evaluation (LREC)*, Granada, Spain, 1998, p. 447–454.

[CHA 96] CHARNIAK E., CARROLL G., ADCOCK J., CASSANDRA A., GOTOH Y., KATZ J., LITTMAN M., MCCANN J., "Taggers for Parsers", *Artificial Intelligence*, vol. 85, num. 1-2, 1996.

[COL 99] COLLINS M., "Head-Driven Statistical Models for Natural Language Parsing", PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1999.

[Del 97] DELLA PIETRA S., DELLA PIETRA V., LAFFERTY J., "Inducing features of random fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, 1997, p. 380–393.

[GAR 87] GARSIDE R., LEECH G., SAMPSON G., *The Computational Analysis of English: A Corpus-Based Approach*, Longman, 1987.

[JEL 98] JELINEK F., *Statistical Methods for Speech Recognition*, MIT Press, 1998.

[JOH 99] JOHNSON M., GEMAN S., CANON S., CHI Z., RIEZLER S., "Estimators for Stochastic "Unification-based" Grammars", *Proceedings of the 37th Annual Meeting of the ACL*, College Park, Maryland, 1999, p. 535–541.

[KIE 99] KIEFER B., KRIEGER H.-U., CARROLL J., MALOUF R., "A Bag of Useful Techniques for Efficient and Robust Parsing", *Proceedings of the 37th Meeting of the ACL*, College Park, MD, 1999, p. 473–480.

[MAN 99] MANNING C. D., SCHÜTZE H., *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge Mass., 1999.

[MAR 93] MARCUS M. P., SANTORINI B., MARCINKIEWICZ M. A., "Building a Large Annotated Corpus of English: The Penn Treebank", *Computational Linguistics*, vol. 19, num. 2, 1993.

[MOO 00] MOORTGAT M., SCHUURMAN I., VAN DER WOUDEN T., "CGN Syntactische Annotatie", 2000, internal report Corpus Gesproken Nederlands.

[NOO 94] VAN NOORD G., BOUMA G., "Adjuncts and the Processing of Lexical Rules", *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, Kyoto, 1994, p. 250–256, cmp-lg/9404011.

[NOO 97] VAN NOORD G., "An Efficient Implementation of the Head Corner Parser", *Computational Linguistics*, vol. 23, num. 3, 1997, p. 425–456, cmp-lg/9701004.

[NOO 99] VAN NOORD G., BOUMA G., KOELING R., NEDERHOF M.-J., "Robust Grammatical Analysis for Spoken Dialogue Systems", *Journal of Natural Language Engineering*, vol. 5, num. 1, 1999, p. 45–93.

[NOO 01] VAN NOORD G., "Robust Parsing of Word Graphs", JUNQUA J.-C., VAN NOORD G., Eds., *Robustness in Language and Speech Technology*, Kluwer Academic Publishers, Dordrecht, 2001.

[OOS 00] OOSTDIJK N., "The Spoken Dutch Corpus: Overview and first evaluation", GRAVILIDOU M., CARAYANNIS G., MARKANTONATOU S., PIPERIDIS S., STAINHAOUER G., Eds., *Proceedings of the Second International Conference on Language Resources and*

Evaluation (LREC), 2000, p. 887–894.

[POL 94] POLLARD C., SAG I., *Head-driven Phrase Structure Grammar*, University of Chicago / CSLI, 1994.

[PRI 01] PRINS R., VAN NOORD G., "Unsupervised POS-Tagging Improves Parsing Accuracy and Parsing Efficiency", *Proceedings of the International Workshop on Parsing Technologies 2001*, Beijing, China, 2001, Tsinghua University Press, p. 154–165.

[SAG 97] SAG I., "English Relative Clause Constructions", *Journal of Linguistics*, vol. 33, num. 2, 1997, p. 431–484.

[SKU 97] SKUT W., KRENN B., USZKOREIT H., "An annotation scheme for free word order languages", *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, DC, 1997.

[VOU 98] VOUTILAINEN A., "Does Tagging Help Parsing? A Case Study on Finite State Parsing", *Finite-state Methods in Natural Language Processing*, Ankara, 1998.

[WAU 95] WAUSCHKUHN O., "The Influence of Tagging on the Results of Partial Parsing in German Corpora", *Fourth International Workshop on Parsing Technologies*, Prague/Karlovy Vary, Czech Republic, 1995, p. 260–270.

[ZAN 99] VAN ZANTEN G. V., BOUMA G., SIMA'AN K., VAN NOORD G., BONNEMA R., "Evaluation of the NLP Components of the OVIS2 Spoken Dialogue System", VAN EYNDE F., SCHUURMAN I., SCHELKENS N., Eds., *Computational Linguistics in the Netherlands 1998*, Rodopi Amsterdam, 1999, p. 213–229.