UNSUPERVISED POS-TAGGING IMPROVES PARSING ACCURACY AND PARSING EFFICIENCY

Robbert Prins
Alfa-informatica & BCN
University of Groningen
r.p.prins@let.rug.nl

Gertjan van Noord
Alfa-informatica & BCN
University of Groningen
vannoord@let.rug.nl

Abstract

It is shown that a simple POS-tagger can be used to filter the results of lexical analysis of a wide-coverage computational grammar. The reduction of the number of lexical categories not only greatly improves parsing efficiency, but in our experiments also gave rise to a mild increase in parsing accuracy; in contrast to results reported in earlier work on supervised tagging. The novel aspect of our approach is that the POS-tagger does not require any human-annotated data - but rather uses the parser output obtained on a large training set.

1 Introduction

Full parsing of unrestricted texts on the basis of a wide-coverage computational HPSG grammar remains a challenge. In our recent experience in the development of the Alpino system, discussed in section 2, we found that even in the presence of various clever 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 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 in similar systems, e.g. in the English Lingo HPSG system [15].

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 [29], [10], [28]. 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*.

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 (of course, this has been proposed earlier, e.g. in [7]). 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 up to about twenty times as fast, while parsing accuracy does not drop. For somewhat less drastic efficiency gains, we observed an increase in parsing accuracy. Parsing accuracy drops considerably, however, if we only use the best tag for each word (this differs from the conclusion in [10]).

Fourthly, whereas in earlier work evaluation was described e.g. in terms of 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 [8]. This evaluation measure presupposes the availability of a treebank, but is expected to reflect much better the accuracy of the system.

We implemented a standard bigram HMM tagger in which the emission probabilities are directly estimated from a labeled training corpus. A standard POS-tagger attempts to find the best sequence of tags for the given input sentence, or perhaps the n-best sequences of tags for small n. As we discuss later, this is not appropriate for our purposes. Rather, we use an idea from chapter 5.7 of [13] by computing the *a posteriori* probability for each tag. We use a threshold in order to cut away for every position in the input string the most unlikely tags. The same idea is described in [10].

In the following section we shortly describe the Alpino wide-coverage parser of Dutch, with which we performed our experiments. In section 3 we describe the tagger in more detail, as well as our method to use that tagger to filter out unlikely tags. In section 4 we report on the results of experiments in which we include the tagger as a filter component of Alpino. We observed that errors introduced by the POS-filter often were related to subcategorization frames of verbs. We therefore experimented with a set-up in which subcategorization information of verbs was hidden from the POS-filter. In section 5 we discuss a number of 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 [5]. 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 OVIS grammar [26, 27], a lexicalized grammar in the tradition of Head-driven Phrase Structure Grammar [19]. 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 relativey detailed. However, as pointed out in [20], by organizing rules in an inheritance hierarchy, the relevant linguistic generalizations can still be captured. The Alpino grammar currently contains over 250 rules, defined in terms of a few general rule structures and 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.

The formalism supports the use of recursive constraints over feature-structures (using delayed evaluation, [25]). This allowed us to incorporate an analysis of cross-serial dependencies based on argument-inheritance [4] and a trace-less account of extraction along the lines of [3].

2.2 Dependency Structures

The Alpino grammar produces dependency structures compatible with the CGN-guidelines. Within the CGN-project [18], guidelines have been developed for syntactic annotation of spoken Dutch [17], using dependency structures similar to those used for the German Negra corpus [21]. Dependency structures make explicit the dependency relations between constituents in a sentence. Each non-terminal node in a dependency structure consists of a head-daughter and a list of non-head daughters, whose dependency relation to the head is marked. A dependency structure for the sentence

(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

is given in figure 1. Control relations are encoded by means of co-indexing (i.e. the subject of *hebben* is the dependent with index **1**). Note that a dependency structure does not necessarily reflect (surface) syntactic constituency. The dependent *haar nieuwe model gisteren aangekondigd*, for instance, does not correspond to a (surface) syntactic constituent.

2.3 Robust Parsing

The initial design and implementation of the Alpino parser is inherited from the system described in [23], [26] and [24]. 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.

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

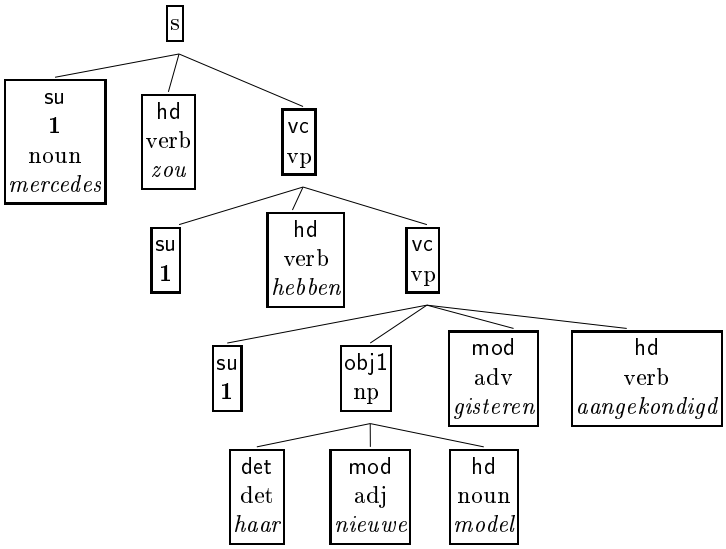

Figure 1: Dependency structure example

For sentence (2), for instance, the lexicon produces 83 tags. Some of those tags are obviously wrong. For example, one of the tags for the word `hebben` is `verb(hebben,pl,part_sbar_transitive(door))`. The tag 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 tag 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 56 tags. The remaining 27 tags are input to the bigram tagger described below. The tagger (using the default settings) will remove 10 of these. After the filter has applied, feature structures are associated with each of the remaining 17 tags. Often, a single tag is mapped to multiple feature structures. The remaining 17 filtered tags give rise to 81 feature structures.

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 [23]. We generalized some of the techniques described there to take into account relational constraints, which are delayed until sufficiently instantiated [25].

As described in [26] and [24], 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.

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 these best parse trees 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 [5] 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 [2] and [11], and applied to stochastic constraint-based grammars by [1] and [14]. 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 a small treebank of about 1300 sentences with associated dependency structures.

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. [1]

3 Using a POS-tagger as a filter

3.1 Training and test data for the POS-tagger

Recall that the tagged training corpus is not annotated by humans, but rather by the parser. Thus, we have run the parser on a large training set, and collected the sequences of lexical categories that were used by the best parse (according to the parser).

Of course, the training set thus produced contains errors, but since the POS-tagger is used to select the best tags out of the tags suggested by the parser, it makes sense to train it on the parser's output as well.

In our experiments discussed below, we used as our corpus the first six months of 1997 of the Dutch newspaper 'de Volkskrant', except that we kept apart some 5,783 sentences (91,857 words) which are used for the stand-alone tests described below. The remaining 517,492 sentences were fed to the parser, with a time-out of 60 CPU-seconds per sentence, and with the robustness component disabled (in such a way that only those sentences were used for which the parser found a complete parse). 324,575 sentences (4,879,085 words) were parsed successfully; the corresponding tag sequences are used to train our bigram model.

[1] Note that this procedure differs from best-first parsing (e.g. [6]) since in our case only the parse selection phase is best-first; the construction of the parse-forest finds all parses.

n	tags/word	accuracy (%)
1	1	70.9
5	1.14	74.8
15	1.36	76.7
25	1.49	77.5
50	1.69	78.4
100	1.91	79.1
200	2.11	79.7
300	2.23	80.0
∞	3.32	100

Table 1: Filter results using the n-best paths approach

We implemented a standard bigram HMM tagger, described e.g. in chapter 10.2 of [16]: an HMM in which each state corresponds to a tag, and in which emission probabilities are directly estimated from a labeled training corpus. For each sentence, the filter is given as input the set of tags found by the lexical analysis component of Alpino. The task of the filter consists of the removal of all unlikely tags. We have experimented with a few techniques to determine which tags are unlikely.

3.2 Using the most likely sequence

The optimal sequence of tags for a given sentence is defined as:

$$\hat{t}_{1,n} = \arg\max_{t_{1,n}} \; P(t_{1,n}|w_{1,n}) = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

The Viterbi algorithm is used to compute this most probable tag sequence. In a first experiment, we simply assumed that a tag is removed if it is not part of the most probable tag sequence. This results in most of the tags being discarded, and leads to low tagging accuracy (first line of Table 1). The table shows the average number of tags per word after the application of the filter and the resulting tagger accuracy. An accuracy of 70.9% suggests that it is not possible to rely on the best sequence.

One might wonder why the results of this tagger are so poor, whereas in the literature tagging is supposed to obtain at least an accuracy level of 95%. This is caused by the size of the tag set (more than 25K different tags). If the tagger would simply use the most frequent tag for each given word (in isolation of its context), then we would obtain a tagger accuracy of about 50%. This should be contrasted with typical taggers in which this base-line is reported to be around 90%.

3.3 Using the n-best sequences

Since using only one sequence leads to low accuracy results, the set of accepted tags is extended to include the tags that make up the n-best sequences. For different values of n Table 1 shows the results. A word is assumed to be tagged correctly if the correct tag is not filtered by the tagger. An accuracy level of 80%, attained by considering the 300 best taggings, is not good enough, since it would still reduce the grammar's chances of finding the right parse too much. Increasing accuracy by considering even more sequences will lead to more ambiguity at the same time, and makes the Viterbi algorithm very slow. Since the number of possible tag sequences increases exponentially with sentence length, we have also experimented with dynamically chosen values of n; these experiments were not very successful either, and for longer sentences (i.e. larger values of n) the Viterbi algorithm itself becomes too slow.

τ	tags/word	accuracy (%)
0	1.00	86.5
1	1.10	89.8
2	1.23	92.6
3	1.37	94.6
4	1.51	96.1
5	1.66	97.0
6	1.81	97.7
7	1.97	98.3
8	2.11	98.7
∞	3.32	100

Table 2: Filter results using forward-backward method

3.4 Using forward and backward probabilities

To increase accuracy and decrease ambiguity, the idea of selecting tags based on the most likely sequences must be abandoned. Instead, probabilities have to be computed for individual tags, to compare tags that are assigned to the same word 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 [13] in the context of speech recognition. The same technique is described in [10].

The 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. We report on experiments with various values of τ.

The results using forward and backward probabilities to compute the likeliness of individual tags are given as Table 2, showing the average number of remaining tags per word and tagger accuracy percentages for various threshold levels τ. The method is a significant improvement over the n-best sequences approach. Considering the large number of different tags in the tag set and the fact that the tagger only uses bigram probabilities, the accuracy percentages might come across as relatively high. However, it must be noted that the tagger is not a completely self-supporting tagging system, but a filter that receives for each word a set of candidate tags from which a selection has to be made.

4 Incorporating the POS-tagger in Alpino

4.1 Evaluation Procedure

The treebank used in the experiments with the Alpino parser is the CDBL (newspaper) part of the Eindhoven corpus [12], which we are currently annotating with dependency structures, according to the guidelines specified in [17]. These dependency structures are similar to those used in the German

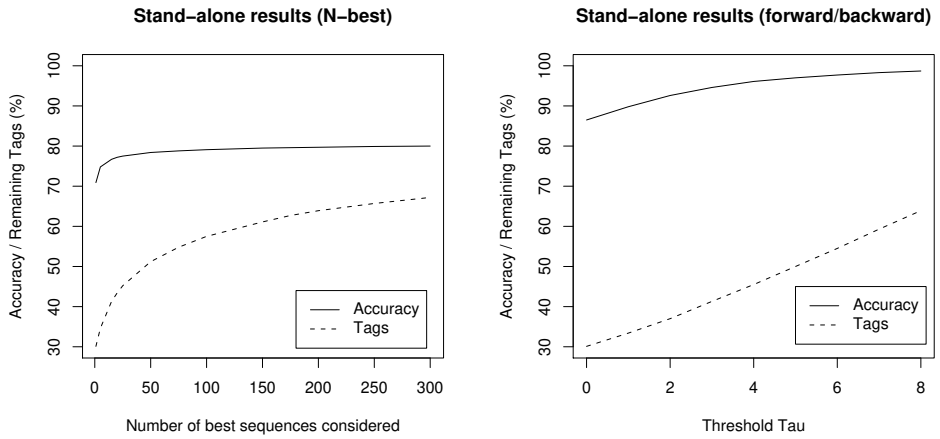

Figure 2: Stand-alone results for N-best and forward/backward techniques

Negra corpus [21]. CDBL220 refers to the first 220 sentences of the CDBL part, which are all annotated. The average sentence length is 20 words.

Evaluation of coverage and accuracy of a computational grammar usually compares tree structures (such as recall and precision of (labelled) brackets or bracketing inconsistencies (crossing brackets) between test item and parser output). As is well-known, such metrics have a number of drawbacks. Therefore, [8] propose to annotate sentences with triples of the form ⟨*head-word, dependency relation, dependent head-word*⟩. For instance, for the example in (2) we obtain:

⟨zou su mercedes⟩	⟨zou vc hebben⟩	
⟨hebben su mercedes⟩	⟨hebben vc aangekondigd⟩	
⟨aangekondigd su mercedes⟩	⟨aangekondigd obj1 model⟩	⟨aangekondigd mod gisteren⟩
⟨model mod nieuwe⟩	⟨model det haar⟩	

Dependency relations between head-words can be extracted easily from the dependency structures in our treebank, as well as from the dependency structures constructed by the parser. It is thus straightforward to compute precision, recall, and f-score on the set of dependency triples. In the experiments described below, the accuracy of the Alpino parser is expressed in terms of this f-score.

4.2 Experiment with POS-filter in Alpino

In Table 3 we summarize the experiments in which the POS-filter is applied as a preprocessing component in Alpino. The POS-filter used the forward and backward probabilities to filter out unlikely tags, as described in the previous section. We experimented with various values of τ. In the table, the first row describes the reference system in which no POS-filter is applied. In that case, all tags are used by the parser, and parsing is very slow. The accuracy is 74.79%. As can be concluded from the table, a threshold value of $\tau = 2$ already performs (slightly) better than the reference system, with a sharp decrease in parsing times.

4.3 Ignoring subcategorization information

Inspection of the errors made by the filter indicated problems with subcategorization information. Since the bigram model uses only a limited history, this information is not always used properly. By

τ	tags/word	CPU (msec)	Accuracy (%)
NONE	3.53	76620	74.79
0	1.05	1421	68.05
1	1.16	2341	71.88
2	1.32	4077	74.89
3	1.46	6620	75.63
4	1.62	10894	75.15
5	1.77	15702	74.94

Table 3: Incorporating the POS-filter in Alpino: results on cdbl220 corpus. (CPU times are averages per sentence)

removing the extra information from the tags, the algorithm can make a better decision in these cases. So, both in training and during the application of the filter, we map each verbal tag to its *class*, by removing the subcategorization specification. For instance, the verb *hebben* in (2) is assigned the following tags:

```
verb(inf,aux_psp_hebben) verb(inf,pred_transitive) verb(inf,transitive)
verb(pl,aux_psp_hebben)  verb(pl,pred_transitive)  verb(pl,transitive)
```

This set is mapped to two classes, `verb(inf)` and `verb(pl)`. If the tagger finds that a class is too unlikely, then all tags that were mapped to that class are removed. Similarly, if a class survives the filter, then all tags which were mapped to that class will be available during parsing. The transformation clearly makes tagging much easier by making the number of different tags much smaller (about 1400 tags remain). This class-based approach typically removes less tags than the previous approach. In Table 4 we show the results. In figure 3 we compare the system without a POS-filter with two systems including the POS-filter, either with or without subcategorization information. As can be seen from these figures, the accuracy of Alpino is improved if subcategorization from verbs is ignored. If $\tau = 2$ then Alpino is almost ten times faster than the reference system (without POS-tagger), and the corresponding accuracy is higher too: 76.40% vs. 74.79%.

5 Future Work

We showed that a simple POS-tagger can be used to filter the results of lexical analysis of a wide-coverage computational grammar. The reduction of the number of lexical categories not only greatly improves parsing efficiency, but in our experiments also gave rise to a mild increase in parsing accuracy; in contrast to results reported in earlier work on supervised tagging. The novel aspect of our approach

τ	tags/word	CPU (msec)	Accuracy (%)
NONE	3.53	76620	74.79
0	1.46	3125	73.33
1	1.58	5221	75.30
2	1.70	7846	76.40
3	1.81	11054	76.29
4	1.94	26415	76.09
5	2.09	34611	75.68

Table 4: POS-filter in Alpino, if subcategorization information is ignored; cdbl220 corpus. (CPU times are averages per sentence)

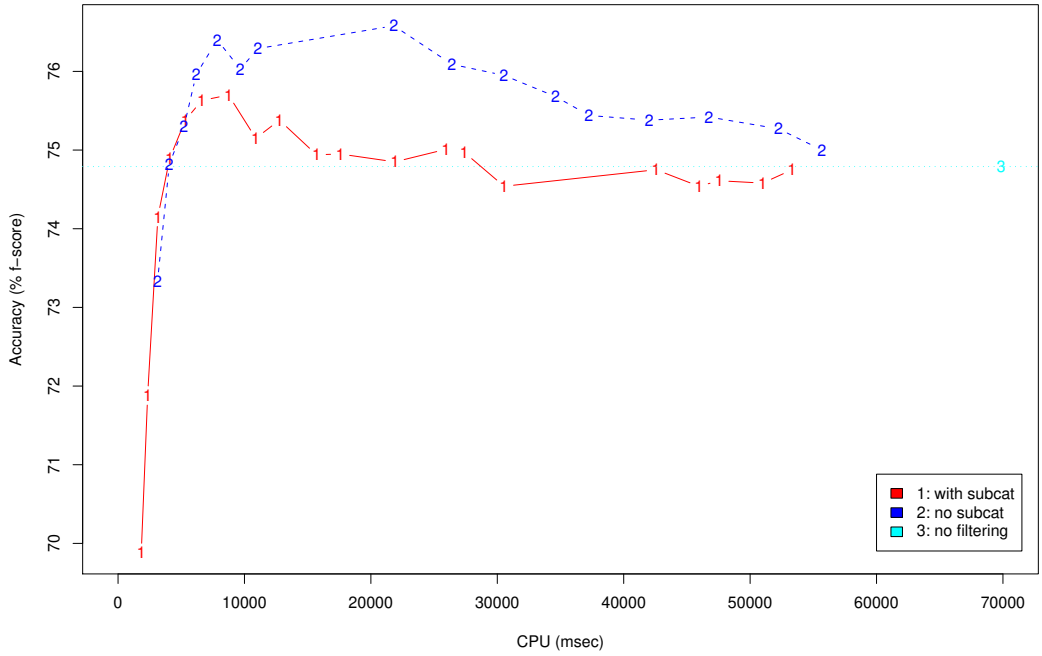

Figure 3: POS-filtering for Alpino

is that the POS-tagger does not require any human-annotated data - but rather uses the parser output obtained on a large training set.

The bigram HMM POS-tagger implemented here is perhaps the simplest POS-tagger proposed in the literature. Therefore, an obvious extension of this work would be to experiment with POS-taggers which have been shown to perform better than the bigram HMM. We did some preliminary experiments with a trigram model, but we were not yet able to improve upon the bigram results.

The context of the work reported in this paper concerns finite-state approximation and grammar specialization techniques. In finite-state approximation, a finite-state grammar is often derived directly from an underlying (typically context-free) grammar. In the approach that we would like to pursue, a finite-state approximation is derived in an indirect way by means of the application of the underlying grammar to a large training corpus. The finite-state approximation is then extracted from the annotated training corpus. The experiment described in this paper is a simple implementation of this technique. The Alpino grammar is applied to a large corpus, and from the parse results we derive a bigram HMM (a stochastic finite-state automaton with a very simple topology). In the future we hope to experiment with finite-state learning techniques that are capable of learning more complex stochastic finite-state automata [9, 22].

Acknowledgments. 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.

References

[1] Steven P. Abney. Stochastic attribute-value grammars. *Computational Linguistics*, 23:597–618, 1997.

[2] Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–72, 1996.

[3] Gosse Bouma, Rob Malouf, and Ivan Sag. Satisfying constraints on adjunction and extraction. *Natural Language and Linguistic Theory*, 19:1–65, 2001.

[4] Gosse Bouma and Gertjan van Noord. Word order constraints on verb clusters in German and Dutch. In Erhard Hinrichs, Tsuneko Nakazawa, and Andreas Kathol, editors, *Complex Predicates in Nonderivational Syntax*, pages 43–72. Academic Press, New York, 1998.

[5] Gosse Bouma, Gertjan van Noord, and Robert Malouf. Wide coverage computational analysis of Dutch. 2001. To appear in Proceedings of CLIN-2000. Available from `http://www.let.rug.nl/~vannoord/`.

[6] Sharon A. Caraballo and Eugene Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2), 1998.

[7] J. Carroll and E. Briscoe. 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*, pages 92–100, University of Pensylvania, Philadelphia, PA, 1996.

[8] John Carroll, Ted Briscoe, and Antonio Sanfilippo. Parser evaluation: A survey and a new proposal. In *Proceedings of the first International Conference on Language Resources and Evaluation (LREC)*, pages 447–454, Granada, Spain, 1998.

[9] Rafael C.Carrasco and Jose Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications (Informatique Théorique et Applications)*, 33:1–19, 1999.

[10] E. Charniak, G. Carroll, J. Adcock, A. Cassandra, Y. Gotoh, J. Katz, M. Littman, and J. McCann. Taggers for parsers. *Artificial Intelligence*, 85(1-2), 1996.

[11] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–393, 1997.

[12] P. C. Uit den Boogaart. *Woordfrequenties in geschreven en gesproken Nederlands.* Oosthoek, Scheltema & Holkema, Utrecht, 1975. Werkgroep Frequentie-onderzoek van het Nederlands.

[13] Frederick Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1998.

[14] Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 535–541, College Park, Maryland, 1999.

[15] B. Kiefer, H.-U. Krieger, J. Carroll, and R. Malouf. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics*, pages 473–480, College Park, MD, 1999.

[16] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge Mass., 1999.

[17] Michael Moortgat, Ineke Schuurman, and Ton van der Wouden. CGN syntactische annotatie, 2000. internal report Corpus Gesproken Nederlands.

[18] Nelleke Oostdijk. The Spoken Dutch Corpus: Overview and first evaluation. In *Proceedings of Second International Conference on Language Resources and Evaluation (LREC)*, pages 887–894, 2000.

[19] Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar*. University of Chicago / CSLI, 1994.

[20] Ivan Sag. English relative clause constructions. *Journal of Linguistics*, 33(2):431–484, 1997.

[21] Wojciech Skut, Brigitte Krenn, and Hans Uszkoreit. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, DC, 1997.

[22] Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. 17th International Conf. on Machine Learning*, pages 975–982. Morgan Kaufmann, San Francisco, CA, 2000.

[23] Gertjan van Noord. An efficient implementation of the head corner parser. *Computational Linguistics*, 23(3):425–456, 1997. cmp-lg/9701004.

[24] Gertjan van Noord. Robust parsing of word graphs. In Jean-Claude Junqua and Gertjan van Noord, editors, *Robustness in Language and Speech Technology*. Kluwer Academic Publishers, Dordrecht, 2001.

[25] Gertjan van Noord and Gosse Bouma. Adjuncts and the processing of lexical rules. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pages 250–256, Kyoto, 1994. cmp-lg/9404011.

[26] Gertjan van Noord, Gosse Bouma, Rob Koeling, and Mark-Jan Nederhof. Robust grammatical analysis for spoken dialogue systems. *Journal of Natural Language Engineering*, 5(1):45–93, 1999.

[27] Gert Veldhuijzen van Zanten, Gosse Bouma, Khalil Sima'an, Gertjan van Noord, and Remko Bonnema. Evaluation of the NLP components of the OVIS2 spoken dialogue system. In Frank van Eynde, Ineke Schuurman, and Ness Schelkens, editors, *Computational Linguistics in the Netherlands 1998*, pages 213–229. Rodopi Amsterdam, 1999.

[28] Atro Voutilainen. Does tagging help parsing? a case study on finite state parsing. In *Finite-state Methods in Natural Language Processing*, Ankara, 1998.

[29] Oliver Wauschkuhn. The influence of tagging on the results of partial parsing in German corpora. In *Fourth International Workshop on Parsing Technologies*, pages 260–270, Prague/Karlovy Vary, Czech Republic, 1995.