An efficient memory-based morphosyntactic tagger and parser for Dutch

Antal van den Bosch, Bertjan Busser, Sander Canisius, Walter Daelemans

ILK Research Group, Dept. of Communication and Information Technology, Tilburg University, the Netherlands
Center for Dutch Language and Speech, Dept. of Linguistics, University of Antwerp, Belgium

Abstract

We describe TADPOLE, a modular memory-based morphosyntactic tagger and dependency parser for Dutch. Though primarily aimed at being accurate, the design of the system is also driven by optimizing speed and memory usage, using a trie-based approximation of k-nearest neighbor classification as the basis of each module. We perform an evaluation of its three main modules: a part-of-speech tagger, a morphological analyzer, and a dependency parser, trained on manually annotated material available for Dutch – the parser is additionally trained on automatically parsed data. A global analysis of the system shows that it is able to process text in linear time close to an estimated 2,500 words per second, while maintaining sufficient accuracy.

1 Introduction

In this paper we introduce TADPOLE (TAgger, Dependency Parser, and mOrphoLogical analyzEr), a modular morpho-syntactic tagger, analyzer and parser for Dutch. In designing TADPOLE we aim for three partially competing goals: (1) high accuracy, (2) high and preferably linear processing speed, and (3) low memory usage. TADPOLE is particularly targeted at the increasing need for fast, automatic NLP systems applicable to very large (multi-million to billion word) document collections that are becoming available due to the progressive digitization of both new and old textual data. This scale does not fit well with systems that perform exponentially in terms of the length of their input, spending perhaps minutes on single sentences, and neither with linear-time but slow processing system that would take, e.g., a second per word – which would imply more than ten days to process just one million words of text.

Rather than a mix of methods, we opt for a single processing engine to be used in all modules to simplify the software engineering aspects. As the core engine we chose memory-based learning, in particular a fast trie-based approximation of k-nearest neighbor classification, IGTREE (Daelemans, Van den Bosch and Weijters 1997a). Memory-based learning has been shown to produce competitive, state-of-the-art performance in part-of-speech tagging (Daelemans, Zavrel, Berck and Gillis 1996) and morphological analysis (Van den Bosch and Daelemans 1999), and has recently also been employed in a dependency parser (Canisius, Bogers, Van den Bosch, Geertzen and Tjong Kim Sang 2006) with some initial success. IGTREE has been shown to speed up normal k-nearest neighbor classification several orders of magnitude, while retaining much of its generaliza-

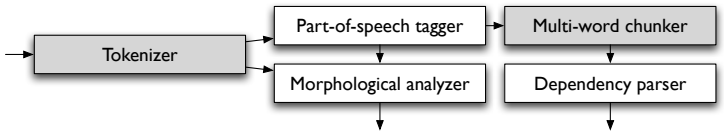

Figure 1: Schematic architecture of TADPOLE. The grey boxes represent non-machine-learning-based modules.

tion accuracy. With IGTREE we aim to reach high processing speed (an aspect of goal 2) and low memory usage (goal 3); the accuracy levels (goal 1) are expected to be lower than those of k-nearest neighbor classification; empirical tests are needed to ascertain the gap.

Linear processing speed (another aspect of goal 2) is straightforwardly achieved with memory-based part-of-speech tagging and morphological analysis; both approaches are fully linear in their default sequence processing method (Daelemans et al. 1996, Van den Bosch and Daelemans 1999). With dependency parsing, however, linearity is an issue. The approach proposed by Canisius et al. (2006) involves a processing step that is quadratic in principle, but linearly bounded, and a deterministic search through the predicted dependency relations.

In this paper we first lay out the architecture of the system in Section 2. We then provide evaluations of the three modules in Section 3, and we evaluate the system globally in Section 4. Related work is discussed in Section 5. We close the paper with a discussion of future work in Section 6.

2 Architecture

The intended function of TADPOLE is to automatically annotate Dutch text with morpho-syntactic information at the word level, and syntactic dependency relations between words at the sentence level. To enable a proper treatment of incoming text, a tokenizer is used for preprocessing. We adopted a rule-based tokenizer that splits punctuation markers from words, using seed lists of common Dutch abbreviations, and that splits sentences according to a set of heuristic rules (Reynaert 2007). Tokenized text is then fed to the part-of-speech tagger and the morphological analyzer. Subsequently, predicted part-of-speech tags are forwarded to the morphological analyzer, which uses the tags to choose among the analyses it has generated for ambiguous words. The tags are also used as input to the dependency parser, which in turn demands that a fixed list of multi-word phrases and all multi-word proper nouns are collated by a straightforward lookup-based multi-word chunker.

Figure 1 schematically illustrates the information flow of the processing modules. Each memory-based module (the white boxes) uses a classification engine that converts its input to a partial output; each conversion step is one classification

of a windowed snapshot of the input sequence into an output label. Sequences of output labels are gathered until the end of the word or sentence, and subsequently converted into a full output (per word for the morphological analyzer, and per sentence for the part-of-speech tagger and dependency parser). Section 3 provides more detailed information on the functioning of each module.

The classifier engine in the three memory-based processing modules is IGTREE (Daelemans et al. 1997a), an algorithm for the top-down induction of decision trees. It compresses a database of labeled examples into a lossless-compression decision-tree structure that preserves the labeling information of all examples, and technically should be named a *trie* according to Knuth (1973). A labeled example is a feature-value vector encoding input (in our case, windowed subsequences of letters, words, or part-of-speech tags) and output (in our case, labels encoding morphological information, part-of-speech tags, or syntactic dependency relation types).

An IGTREE is a hierarchical tree composed of nodes that each represent a partition of the original example database, and are labeled by the most frequent class of that partition. Besides a majority class label, the nodes also hold complete counts of all class labels in the database partition they represent. The root node of the trie thus (1) represents the entire example database, (2) carries the most frequent value as class label, and (3) holds the occurrence counts of all classes in the full training set. In contrast, end nodes (leafs) represent a homogeneous partition of the database in which all examples have the same class label; the node merely stores this label along with the size of the homogeneous partition. Non-ending nodes branch out to nodes at deeper levels of the trie. Each branch represents a test on a feature value; branches fanning out of one node test on values of the same feature.

To attain high compression levels, IGTREE branches out from the root node by testing on the most informative, or most class-discriminative feature first, followed at the next level by the second-most discriminative feature. IGTREE uses information gain (IG) to estimate discriminativeness. The IG of feature i is measured by computing the difference in uncertainty (i.e. entropy) between the situations without and with knowledge of the value of that feature with respect to predicting the class label: $IG_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$, where C is the set of class labels, V_i is the set of values for feature i, and $H(C) = -\sum_{c \in C} P(c) \log_2 P(c)$ is the entropy of the class labels. IGTREE computes the IG of all features once on the full database of training examples, makes a feature ordering once on these computed IG values, and uses this ordering throughout the whole trie.

IGTREE effectively performs a lossless compression of the labeling information of the original example database. As long as the database does not contain fully ambiguous examples (with the same features, but different class labels), the trie produced by IGTREE is able to reproduce the classifications of all examples in the original example database perfectly.

3 Modules

We describe for each of the three IGTREE-based modules how their tasks are encoded into classification tasks, and provide estimates of their generalization performance on unseen words and text.

3.1 Part-of-speech tagging

The approach to part-of-speech tagging taken in TADPOLE was originally introduced by Daelemans et al. (1996). The proposed tagger is a combination of a submodule that disambiguates the tags of words it has seen before, given their context, and a submodule that predicts tags to words it has not seen before. Both taggers process from left to right, and use windowing to represent the local context around the word to be tagged. The left part of the window also includes the joint tagger's previously predicted tags, while in the right part of the window the yet ambiguous tags of the known right neighboring words are incorporated.

The second submodule, the *unknown words* tagger, cannot use the word in focus as a predictive feature since it has not seen it before, but some surface features of the word are represented. Furthermore, both taggers are helped by converting low-frequency words to more generic placeholder strings that retain some of their surface features. Also, the unknown words tagger is not trained on the full training set, but rather on a subset of low-frequency words in their context in the training set, as they are the most representative of actual unseen words, which will tend to occur in the same frequency band. In detail, the features for the two subtaggers are the following:

- For the *known words* tagger: the focus word and its immediate left and right neigboring words, the three preceding predicted tags, and the two still ambiguous tags to the right.
- For the *unknown words* tagger: the first two letters and the last three letters of the focus word; binary features marking whether the word is capitalized, contains a hyphen, or one or more numbers; its immediate left and right neighboring words; the three preceding predicted tags, and the two still ambiguous tags at the right.

When trained on a substantial training corpus, often less than 10% (or even less than 5%) of words in new text will not have occurred in the training corpus. Hence, the first submodule, the *known words* tagger, is responsible for a major part of the work. Yet, the remaining work for the unknown word tagger is harder. For the TADPOLE part-of-speech tagger we opted to use IGTREE for the known words tagger, but use TRIBL for the unknown words tagger. TRIBL is a hybrid between the fast approximation IGTREE and the slower IB1-IG algorithm that implements k-nearest neighbor in its unabridged form (Daelemans, Van den Bosch and Zavrel 1997b)[1]; it builds a trie structure for the most informative features, and

[1] IGTREE, TRIBL, and IB1-IG are included in the TiMBL software package, version 5.1, available from `http://ilk.uvt.nl/timbl`.

Task	Full tag	Main tag
Known words	96.8	98.7
Unknown words	76.4	84.3
All words	96.5	98.6

Table 1: Percentages of correctly tagged test words, overall (bottom line) and split into known words and unknown words, on the full tag and on the main tag only.

performs k-nearest neighbor classification on the remaining features. For building the tagger, the Mbt wrapper was used[2].

The data used for training the TADPOLE tagger consists of a broad selection of available manually annotated part-of-speech tagged corpora for Dutch tagged with the Spoken Dutch Corpus tagset (Van Eynde 2004): The approximately nine-million word of the transcribed Spoken Dutch Corpus itself (Oostdijk, Goedertier, Van Eynde, Boves, Martens, Moortgat and Baayen 2002), the ILK corpus with approximately 46 thousand part-of-speech tagged words, the D-Coi corpus with approximately 330 thousand words, and the 754-thousand word Eindhoven corpus (Uit den Boogaart 1975) which has been automatically retagged with the Spoken Dutch Corpus tagset. Together this accounts for 10,979,827 manually-checked part-of-speech tagged words, all using the same rich tagset of 316 tags.

We split this 10 million-word corpus randomly (at the sentence level) into a 90% training set and a 10% test set. The performance of the tagger on known words and unknown words in the test set, as well as on all test words, is listed in Table 1. Not surprisingly, the tagger has significantly more trouble tagging unknown words. The Spoken Dutch Corpus tagset makes a distinction between the main tag (a traditional 12-tag distinction) and the morphosyntactic subtags, which are not always used in higher-level applications; the generalization accuracy on the main tag reaches a respectable 98.6%.

In the overall tagging accuracy, the influence of the unknown-word tagger is of course related to the amount of unknown words in the text to be tagged. In the 10% test set, about 98.8% of all tokens is also present in the 90% training set, but this test is a sentence-level partition of the same texts as the training set is drawn from. Typically, coverage of tokens in a randomly selected text from outside the (genres of the) training set will be somewhat lower, as illustrated by the following two examples. A first random text, offering general instructions on Unix, containing many foreign words and command line fragments, is covered by 89.8%. The second text, the full text of the novel *Het boetekleed*, a Dutch translation of Ian McEwen's *Atonement*, is covered by 97.9%.

[2]Mbt, version 2.0.1: `http://ilk.uvt.nl/mbt`

instance number	left context					focus letter	right context					TASK
1	_	_	_	_	_	a	b	n	o	r	m	A
2	_	_	_	_	a	b	n	o	r	m	a	0
3	_	_	_	a	b	n	o	r	m	a	l	0
4	_	_	a	b	n	o	r	m	a	l	i	0
5	_	a	b	n	o	r	m	a	l	i	t	0
6	a	b	n	o	r	m	a	l	i	t	e	0
7	b	n	o	r	m	a	l	i	t	e	i	0
8	n	o	r	m	a	l	i	t	e	i	t	0+Da
9	o	r	m	a	l	i	t	e	i	t	e	N_A*
10	r	m	a	l	i	t	e	i	t	e	n	0
11	m	a	l	i	t	e	i	t	e	n	_	0
12	a	l	i	t	e	i	t	e	n	_	_	0
13	l	i	t	e	i	t	e	n	_	_	_	0
14	i	t	e	i	t	e	n	_	_	_	_	m
15	t	e	i	t	e	n	_	_	_	_	_	0

Table 2: Instances with morphological analysis classifications derived from abnormaliteiten, analyzed as [abnormaal]$_A$[iteit]$_{N\_A*}$[en]$_m$.

3.2 Morphological analysis

We take the task of analyzing the morphology of Dutch words to include (1) segmenting a wordform into its morphemes; (2) labeling each morpheme with its function (e.g. a stem with a certain part-of-speech tag, or being a derivational affix, or an inflection), and (3) identifying all spelling changes between the wordform and its underlying morphemes (Van den Bosch and Daelemans 1999). We draw our examples from the CELEX lexical database (Baayen, Piepenbrock and van Rijn 1993), which features a full morphological analysis for 363,690 of them. We took each wordform and its associated analysis, and created task examples using a windowing approach, which transforms each wordform into as many examples as it has letters. Each example focuses on one letter, and includes a fixed number of left and right neighbor letters, chosen here to be five. Consequently, each example spans eleven letters, which is also the average word length in the CELEX database.

To illustrate the construction of examples, Table 2 displays the 15 examples derived from the Dutch example word abnormaliteiten (abnormalities) and their associated classes. The class of the first example is "A", which means that the morpheme starting in a is an adjective ("A"). This morpheme continues up to the eighth example, which is labeled with "0+Da", meaning that at that position, an a is deleted from the underlying morpheme. The coding thus tells that the first morpheme is the adjective abnormaal. The second morpheme, iteit, has class "N_A*". This complex tag indicates that when iteit attaches right to an adjective (encoded by "A*"), the new combination becomes a noun ("N_"). Finally, the third morpheme is en, which is a plural inflection (labeled "m" in CELEX).

This way we generated a database of 3,209,064 examples. Within these examples, 3,806 different class labels occur. The most frequently occurring class label is "0", occurring in 69.3% of all instances. The three most frequent non-null labels are "N" (6.9%), "V" (4.2%), and "A" (1.3%).

When a wordform is listed in CELEX as having more than one possible morphological labeling (e.g., a morpheme may be N or V, the inflection -en may be plural for nouns or infinitive for verbs), these labels are joined into ambiguous classes ("N/V"). Ambiguity in syntactic and inflectional tags occurs in 3.6% of all morphemes in our CELEX data. When the morphological analyzer generates more than one analysis based on these ambiguous classes, it asks for the part-of-tagger to break the tie – hence the arrow from the tagger to the analyzer in Figure 1. We created a translation table between combinations of CELEX main tags and inflectional markers such as "m" on the one hand, and the CGN tags of the part-of-speech tagger on the other hand, to allow matching the CGN tags to the ambiguous analyses. We observed that when the tagger is correct and the analyzer generates the appropriate analyses, the CGN tags predicted by the tagger, with their main tag and the morpho-syntactic subtags, always provide sufficient matches to disambiguate between ambiguous analyses. If due to an error of either module no match is possible to break the tie, a random choice is made.

To evaluate the morphological analyzer, we split the CELEX database randomly in a 90% training set (of 362,690 words, or 2,888,197 examples) and a 10% test set (of 36,369 words, or 320,867 examples). When trained on the full 90% training set, IGTREE correctly segments 79.0% of test words; e.g., it would segment abnormaliteiten correctly into [abnormal][iteit][en]. Also taking into account spelling changes and morpheme types (stems with part-of-speech, affixes, inflections, e.g. [abnormaal]$_A$[iteit]$_{N\_A*}$[en]$_m$), 56.3% of all test words are fully correctly analyzed. These generalization accuracies, obtained on a random 10% of CELEX words, can be seen as approximations of the analyzer's performance on unknown words in free text. Performing a coverage check similar to the one in the previous section, we observe that CELEX covers about 98.3% of the tokens in the test material of the tagger, 83.9% of the Unix instruction document, and 98.1% of the word tokens in *Het boetekleed*. As IGTREE performs a lossless compression of the training set, the analysis or alternate analyses of any word that is also in CELEX will be flawlessly retrieved; hence, the effective accuracy of the analyzer on a text such as the novel is at least 98.1%, and possibly around 99%, as we estimated that about 56.3% of unknown words receives a correct analysis.

3.3 Dependency parsing

In the TADPOLE approach to dependency parsing, IGTREE is trained to predict (directed) labeled dependency relations between a head and a dependent. For each token in a sentence, examples are generated where this token is a potential dependent of each of the other tokens in the sentence. To prevent explosion of the number of classification cases to be considered for a sentence, we restrict the maximum distance between a token and its potential head. We selected this distance

so that 95% of the dependency relations in the training data are covered, which is at a maximum distance of eight words. The label that is predicted for each classification case serves two different purposes at once: 1) it signals whether the token is a dependent of the designated head token, and 2) if the instance does in fact correspond to a dependency relation in the resulting parse of the input sentence, it specifies the type of this relation as well.

The features we used for encoding instances for this classification task correspond to a rather simple description of the head-dependent pair to be classified. For both the potential head and dependent, there are features encoding a 1-1-1 window of words and part-of-speech tags predicted by our tagger; in addition, there are two spatial features: a relative position feature, encoding whether the dependent is located to the left or to the right of its potential head, and a distance feature that expresses the number of tokens between the dependent and its head.

Thus, dependency parsing is first broken down into classifications at the level of word-to-word dependency relations. In a second step these relations need to be gathered per sentence to form a dependency tree. A dependency tree is regarded as a set of dependency relations connecting a head and a dependent. For a set of such relations to form a valid dependency tree, some constraints should be satisfied: 1) each token can only be linked as a dependent to maximally one head token (though a token may be a head to more than one dependent), and 2) dependency relations should not form a cycle. As long as these two constraints are satisfied, a dependency tree can be treated as a set of dependency relations without losing any information.

Naively applying this approach results in a number of practical issues however, which may also negatively affect the performance. First, the classification task as formulated gives rise to a highly skewed class distribution in which examples that correspond to a dependency relation are largely outnumbered by "negative" examples. Second, there is a quadratic increase of instances to be classified as sentence length increases, that is, a sentence of n tokens translates to $n(n-1)$ classification cases.

One issue that may arise when considering each potential dependency relation as a separate classification case is that inconsistent trees are produced. For example, a token may be predicted to be a dependent of more than one head. To recover a valid dependency tree from the separate dependency predictions, a simple inference procedure is performed. Consider a token for which the dependency relation is to be predicted. For this token, a number of classification cases have been processed, each of them indicating whether and if so how the token is related to one of the other tokens in the sentence. Some of these predictions may be negative, i.e. the token is not a dependent of a certain other token in the sentence, others may be positive, suggesting the token is a dependent of some other token.

If all classifications are negative, the token is assumed to have no head, and consequently no dependency relation is added to the tree for this token. If one of the classifications is non-negative, suggesting a dependency relation between this token as a dependent and some other token as a head, this dependency relation is added to the tree. Finally, there is the case in which more than one prediction

is non-negative. By definition, at most one of these predictions can be correct; therefore, only one dependency relation should be added to the tree. To select the most-likely candidate from the predicted dependency relations, the candidates are ranked according to the classification confidence of the base classifier that predicted them, and the highest-ranked candidate is selected for insertion into the tree. For example, if in the sentence Ik hoor haar zingen, *I hear her singing*, the word haar is classified as relating to hoor in the "OBJ1" relation (direct object) with confidence 8, and to zingen in the "DET" relation (determiner) with confidence 5, the first prediction is selected, and the second discarded.

As a measure of confidence for the predictions made by IGTREE we divide the tree-node counts assigned to the majority class by the total counts assigned to all classes. Though this confidence measure is rather crude, and should not be confused with any kind of probability, it tends to work quite well in practice (Canisius et al. 2006).

The base classifier in our parser is faced with a classification task with a highly skewed class distribution, i.e. instances that correspond to a dependency relation are largely outnumbered by those that do not. In practice, such a huge number of negative instances usually results in classifiers that tend to predict fairly conservatively, resulting in high precision, but low recall. In the approach introduced above, however, it is better to have high recall, even at the cost of precision. A missed relation by the base classifier can never be recovered by the inference procedure. Also, due to the constraint that each token can only be a dependent of one head, excessive prediction of dependency relations can still be corrected by the inference procedure. An effective method for increasing the recall of a classifier is downsampling of the training data. In downsampling, instances belonging to the majority class (in this case the negative class) are removed from the training data, so as to obtain a more balanced distribution of negative and non-negative instances.

Canisius et al. (2006) describe the effect of systematically removing an increasingly larger part of the negative instances from the training data. They report that downsampling helps to improve recall, at the cost of precision, but indeed improving the dependency parser, with a maximal performance at downsampling rate $1:2$ (i.e. twice as many negative examples as positive ones). Note that downsampling is naturally restricted to the training data; the test data is not downsampled as the labeling is not known yet.

As training material for our parser we used all manually annotated data available in the Alpino Treebank[3] (Van der Beek, Bouma, Malouf and Van Noord 2001), amounting to 262,452 words, converted to 2,959,456 pairwise examples, and subsequently downsampled to 726,440 examples. We also collected data that is automatically parsed by the Alpino parser (Malouf and Van Noord 2004), available in significantly larger quantities than manually annotated data. We added several millions words of automatically parsed text from Wikipedia pages, newspaper articles, and the full Eindhoven corpus except a portion taken out as test set

[3]Alpino Treebank: `http://www.let.rug.nl/~ vannoord/trees/`.

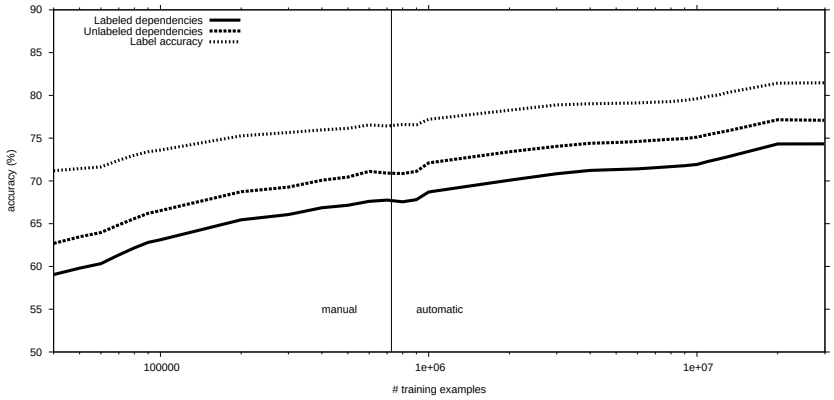

Figure 2: Dependency parsing learning curves in terms of correctly labeled dependencies, unlabeled dependencies, and label accuracy.

(see below). We converted this Alpino output to the column format used in the CoNLL-X Shared Task (Buchholz and Marsi 2006), replacing the part-of-speech information generated by Alpino by the output of TADPOLE's tagger described in Subsection 3.1. Also in this process, in special cases (particularly with multi-word units and coordinations without a conjunction) multiple heads in the original treebank are discarded, keeping only the leftmost head.

Figure 2 displays the learning curves of three commonly used evaluation metrics (Buchholz and Marsi 2006), viz. labeled and unlabeled dependency relation accuracy, and the accuracy on the label per word. The test set consists of 2,530 sentences (47,471 words) taken from the manually parsed section of the Eindhoven corpus (the *cdbl* part); this is newspaper text with relatively long sentences with many subclauses and quotations. The vertical line at 726,400 downsampled pairwise examples marks the transition of manually labeled material to automatically parsed data. Despite a dip in performance in all three evaluation metrics, the curves suprisingly return to their trajectories, and continue to rise – albeit at a sub-loglinear rate with increasing amounts of training data. The exact scores of the parser, trained on a current maximum of 29,778,197 examples, and tested on the aforementioned manually parsed test set, are displayed in Table 3. At best, the parser identifies and labels dependency relations between words at an accuracy of 74.3.

4 Speed and memory usage analysis

Thus far we have not reported on speeds and memory usage, except in passing when comparing the morphological analyzer to IB1-IG. Three design goals of TADPOLE relate to speed and memory: we want the system to be fast, as linear as possible in the length of the input, and costing as little memory as possible.

Aspect	% Correct assignment	
	Only manual data	Automatic data added
Labeled dependencies	67.3	74.3
Unlabeled dependencies	70.6	77.1
Label accuracy	76.3	81.5

Table 3: Percentages of correctly assigned dependencies, with and without labeling, and the accuracy on labels only, trained on the maximal amount of training data, tested on newspaper texts, before and after the addition of automatically parsed training data.

Module	Memory (Mb)	1000 words/s
Part-of-speech tagging	23.3	10.1
Morphological analyzer	2.9	6.7
Dependency parser	68.9	7.6
Tokenizer (rule-based)	Perl	81.9
MWU chunker (rule-based)	Perl	120.3
Total	95.1 + Perl	2.5

Table 4: Amount of memory used, and numbers of words processed by the five modules at maximal training set sizes. Bottom line sums the amount of memory, and aggregates the speeds.

We measured the speed of our classifiers in terms of the number of words they processed per second, and the bytesize of the IGTREES[4]. Table 4 summarizes the measurements taken at the maximal sizes of the training sets used in the previous section to estimate the generalization accuracies of each module. The table also lists the speed of the rule-based tokenizer and multi-word chunker for completeness, as these modules do cost some memory[5] and time. As can be seen in the table, the parser consumes most memory, being trained also on the largest amount of training examples (nearly 30 million). The part-of-speech tagger consumes a fair bit of memory as well, due to the TRIBL-based *unknown words* tagger.

Disregarding the fast rule-based preprocessing modules, the tagger is the fastest module with about 10,160 words per second, while the morphological analyzer is the slowest, processing about 6,715 words per second. Given a single processor, the aggregated speed with which TADPOLE can process text with all three modules is about 2,488 words per second. This number assumes single-CPU, full streaming performance.

One remaining design goal is to include a parser with preferably linear performance. We measured the speed and accuracy of the parser on different sentence

[4]The hardware used for testing is equipped with Dual Core AMD Opteron 880 2,412 Mhz processors.
[5]They are implemented as Perl scripts and require the Perl executable at runtime.

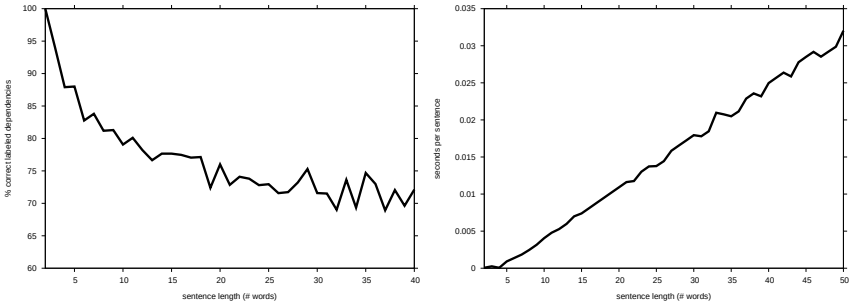

Figure 3: Generalization accuracies (left) and seconds per sentence (right) of the dependency parser trained on maximal amounts of data, measured per sentence length from 2 to 50.

lengths found in our test set. Figure 3 shows both, measured separately for all sentence lengths from 2 to 50. As the left graph of Figure 3 shows, sentences shorter than length 20 are parsed at above-average performance levels. The right graph of Figure 3 shows a perhaps more unexpected linear relation between the length of a sentence and the average time it takes to parse it. Earlier we noted that for each sentence pairwise examples are generated ($n(n-1)$, to be exact), but we also constrained this (also with test sentences) to pairs of words within a range of eight words from each other, as 95% of all relations in the training corpus occur within that range. This fixed constraint bounds the number of examples per sentence, making the relation between the sentence length and the number of examples effectively linear.

5 Related research

Most if not all related work on morpho-syntactic analysis, tagging, and parsing on Dutch has focused on these tasks in isolation. Schone and Jurafsky (2000) describe an unsupervised approach to computational morphological analysis, using CELEX as a gold standard. Their knowledge-free method analyzes words in a large corpus above a frequency threshold of 10. Matching these analyses to the ones in CELEX, they report F-scores on correctly identified morphemes of around 79.6. Without a direct comparison, we can safely say that our supervised system vastly outperforms this system, even if we would only look up analyses from CELEX (which their system is obviously not allowed to).

Van Halteren, Zavrel and Daelemans (2001) provide generalization accuracies of various tagging systems trained on Dutch data annotated using the Wotan tagset, a predecessor of, and comparable to, the CGN tagset. Using additional learning methods (hidden markov models, transformation-based learning, and maximum-entropy tagging) and combinations of these taggers in ensemble architectures, but using only the 754-thousand-words Eindhoven corpus, the best cross-validated ac-

curacy reported is 93.3%, and 96.4% using a reduced version of the tagset, "Wotan-Lite"; this is the performance of a stacked ensemble of classifiers. In contrast, with about 10 million words of training data we attain about the same accuracy (96.5%) in a similar experiment with a tagset that is at least as rich as Wotan, but using a single classifier.

Buchholz and Marsi (2006) provide an overview of systems who competed in the CoNLL-X Shared Task, which also used a part of the manually annotated Alpino treebank, split in training data (195,069 words, 13,349 sentences) and test data (5,463 words, 386 sentences). For the best system (McDonald, Lerman and Pereira 2006) a labeled dependency score of 79.2 is reported, clearly superior to our 74.3 (obtained with more training data, tested on a different test set). Yet, this best performing system is a more complicated two-stage discriminative parser that first performs unlabeled parsing, and then assigns labels, and runs in cubic time as opposed to our linear parser.

An obvious competitor to our parser is the original Alpino parser (Malouf and Van Noord 2004) which it hopes to emulate. Probably the best parser for Dutch, Alpino is a typical modern example of a rule-based approach that has hybridized with a stochastic, data-driven approach. After a rule-based core generates possible parses for a given sentence (possibly hundreds or thousands), a stochastic component searches in this space of possibilities for the most likely parse, where the statistics are derived from the Alpino treebank.

Alpino has been evaluated with various metrics; Malouf and Van Noord (2004) argue for using an adapted form of *concept accuracy* to estimate the correctness of the dependency labeling. The labeled dependencies accuracy metric of the CoNLL-X shared task (Buchholz and Marsi 2006), used in this paper, has the same aim; both metrics essentially compute $\#correct/\#total$, i.e., the number of correctly assigned relations divided by the total number of relations. The difference between the two metrics is that Alpino generates a limited amount of non-terminal nodes in its trees, which necessitates their metric, where in our case the number of generated relations will never be larger than the number of tokens, hence the simple labeled dependency accuracy metric suffices. Given this, we cannot currently compare our parsers to Alpino. Still, it is interesting to contrast some results obtained on the same or similar test sets. On a similar test set to ours, composed of news articles, Alpino is reported to attain a concept accuracy of 87.9%, which is markedly higher than our 74.3% accuracy on labeled dependencies. On a small corpus of quesions, Alpino attains a concept accuracy of 88.7%; a test of our parser on this corpus yields a labeled dependency accuracy of 78.7%. Clearly, our parser lags behind Alpino in terms of accuracy.

6 Discussion

We have described the TADPOLE system, a robust modular morphological analyzer, part-of-speech tagger, and dependency parser for Dutch. Including the classification engine, the complete system costs about 95 Mb of memory, and has an estimated processing speed of close to 2,500 words per second, assuming a com-

mon processor type and full streaming performance. The tagger is estimated to be about 96.5% correct on unseen text (98.6% in terms of main tags). The morphological analyzer can segment about 79.0% of unseen words correctly, and can produce a completely correct analysis with part-of-speech tags and spelling changes for 56.3% of unseen words. The coverage of the tagger and the morphological analyzer is quite high; a random novel text is covered at about 98% of all tokens. In the case of the morphological analyzer this means that it is able to losslessly reproduce correct analyses for at least these 98% tokens. The dependency parser, feeding on tags generated by the part-of-speech tagger, generates dependency relations between pairs of words at an accuracy rate of about 74.3%. The parser is observed to parse in linear time in function of the length of the input; although it has a quadratic component in the example generation process, this process is constrained by a threshold that makes the number of examples linear in the length of the sentence.

In future work we aim to prolong the learning curve of the dependency parser, as much more training data is still available. If the learning curve does not flatten too much it may be possible in the long run to develop a linear-time memory-based emulation of the Alpino parser. We may introduce some extra internal flow of information, such as from the morphological analyzer to the unknown-words module of the part-of-speech tagger. Other future work involves the incorporation of other modules into TADPOLE such as a named-entity recognizer, a semantic role labeler, and a co-reference module, so that the abbreviation will stand for TAgger, Dependency Parser, and Other Language Engines.

Acknowledgements

We gratefully acknowledge the contributions to the different modules of Sabine Buchholz (the tokenizer), Jakub Zavrel (the tagger), Ko van der Sloot (the Timbl software), and Ton Weijters (the IGTREE algorithm). Thanks also to the anonymous reviewers for valuable suggestions. We are indebted to Gertjan van Noord and his co-workers for their invaluable work on the Alpino parser. This work was funded by NWO, The Netherlands Organisation for Scientific Research, as part of the IMIX Program and the Vici project “Implicit Linguistics”.

References

Baayen, R. H., Piepenbrock, R. and van Rijn, H.(1993), *The CELEX lexical data base on CD-ROM*, Linguistic Data Consortium, Philadelphia, PA.

Buchholz, S. and Marsi, E.(2006), CoNLL-X shared task on multilingual dependency parsing, *Proceedings of CoNLL-X, the Tenth Conference on Computational Natural Language Learning*, New York, NY.

Canisius, S., Bogers, T., Van den Bosch, A., Geertzen, J. and Tjong Kim Sang, E.(2006), Dependency parsing by inference over high-recall dependency predictions, *Proceedings of the Tenth Conference on Computational Natural Language Learning, CoNLL-X*, New York, NY.

Daelemans, W., Van den Bosch, A. and Weijters, A.(1997a), IGTree: using trees for compression and classification in lazy learning algorithms, *Artificial Intelligence Review* **11**, 407–423.

Daelemans, W., Van den Bosch, A. and Zavrel, J.(1997b), A feature-relevance heuristic for indexing and compressing large case bases, *in* M. Van Someren and G. Widmer (eds), *Poster Papers of the Ninth European Conference on Machine Learing*, University of Economics, Prague, Czech Republic, pp. 29–38.

Daelemans, W., Zavrel, J., Berck, P. and Gillis, S.(1996), MBT: A memory-based part of speech tagger generator, *in* E. Ejerhed and I. Dagan (eds), *Proceedings of the Fourth Workshop on Very Large Corpora*, ACL SIGDAT, pp. 14–27.

Knuth, D. E.(1973), *The art of computer programming*, Vol. 3: Sorting and searching, Addison-Wesley, Reading, MA.

Malouf, R. and Van Noord, G.(2004), Wide coverage parsing with stochastic attribute value grammars, *Proceedings of the IJCNLP-04 Workshop Beyond Shallow Analyses - Formalisms and statistical modeling for deep analyses*.

McDonald, R., Lerman, K. and Pereira, F.(2006), Multilingual dependency analysis with a two-stage discriminative parser, *in* L. Màrquez and D. Klein (eds), *Proceedings of the Tenth Conference on Computational Natural Language Learning, CoNLL-X*, New York, NY, USA.

Oostdijk, N., Goedertier, W., Van Eynde, F., Boves, L., Martens, J., Moortgat, M. and Baayen, H.(2002), Experiences from the spoken dutch corpus project, *in* M. González Rodriguez and C. Paz Suárez Araujo (eds), *Proceedings of the third International Conference on Language Resources and Evaluation*, pp. 340–347.

Reynaert, M.(2007), Sentence-splitting and tokenization in D-Coi, *Technical Report ILK 07-03*, ILK Research Group.

Schone, P. and Jurafsky, D.(2000), Knowledge-free induction of inflectional morphologies, *Proceedings of the North American Chapter of the Association of Computational Linguistics*, Pittsburgh, PA, USA.

Uit den Boogaart, P.(1975), *Woordfrequenties in geschreven en gesproken Nederlands*, Oosthoek, Scheltema & Holkema, Utrecht.

Van den Bosch, A. and Daelemans, W.(1999), Memory-based morphological analysis, *Proceedings of the 37th Annual Meeting of the ACL*, Morgan Kaufmann, San Francisco, CA, pp. 285–292.

Van der Beek, L., Bouma, G., Malouf, R. and Van Noord, G.(2001), The alpino dependency treebank, *Selected Papers from the Twelfth Computational Linguistics in the Netherlands Meeting, CLIN-2001*, Rodopi, Amsterdam.

Van Eynde, F.(2004), Part of speech tagging en lemmatisering van het Corpus Gesproken Nederlands, *Technical report*, Centrum voor Computerlinguïstiek, K.U. Leuven.

Van Halteren, H., Zavrel, J. and Daelemans, W.(2001), Improving accuracy in word class tagging through combination of machine learning systems, *Computational Linguistics* **27**(2), 199–230.