The Alpino Dependency Treebank

L. van der Beek, G. Bouma, R. Malouf, G. van Noord

Rijksuniversiteit Groningen

Abstract

In this paper we present the Alpino Dependency Treebank and the tools that we have developed to facilitate the annotation process. Annotation typically starts with parsing a sentence with the Alpino parser, a wide coverage parser of Dutch text. The number of parses that is generated is reduced through interactive lexical analysis and constituent marking. A tool for on line addition of lexical information facilitates the parsing of sentences with unknown words. The selection of the best parse is done efficiently with the parse selection tool. At this moment, the Alpino Dependency Treebank consists of about 6,000 sentences of newspaper text that are annotated with dependency trees. The corpus can be used for linguistic exploration as well as for training and evaluation purposes.

1 Introduction

Alpino is a wide-coverage computational analyzer of Dutch that aims at accurate, full parsing of unrestricted text. In section 2 we present the setup of the grammar and the constraint-based formalism that we have adopted. A syntactically annotated corpus is needed to train the grammar and to evaluate its performance. For this purpose we have started to develop the Alpino Dependency Treebank.

The treebank consists of sentences from the newspaper (`cdbl`) part of the Eindhoven corpus (Uit den Boogaard 1975). The sentences are each assigned a dependency structure, which is a relatively theory independent annotation format. The format is taken from the corpus of spoken Dutch (CGN)[1] (Oostdijk 2000), which in turn based its format on the Tiger Treebank (Skut 1997). In section 3 we go into the characteristics of dependency structures and motivate our choice for this annotation format.

Section 4 is the central part of this paper. Here we explain the annotation method as we use it, the tools that we have developed, the advantages and the shortcomings of the system. It starts with a description of the parsing process that is at the beginning of the annotation process. Although it is a good idea to start annotation with parsing (building dependency trees manually is very time consuming and error prone), it has one main disadvantage: ambiguity. For a sentence of average length typically a set of hundreds or even thousands of parses is generated. Selection of the best parse from this large set of possible parses is time intensive.

The tools that we present in this paper aim at facilitating the annotation process and making it less time consuming. We present two tools that reduce the number of parses generated by the parser and a third tool that facilitates the addition of lexical information during the annotation process. Finally a parse selection tool is developed to facilitate the selection of the best parse from the reduced set of parses.

[1]`http://lands.let.kun.nl/cgn`

The Alpino Dependency Treebank is a searchable treebank in an XML format. In section 5 we present examples illustrating how the standard XML query language XPath can be used to search the treebank for linguistically relevant information. In section 6 we explain how the corpus can be used to evaluate the Alpino parser and to train the probabilistic disambiguation component of the grammar. We end with conclusions and some pointers to future work in 7.

2 The grammar

Alpino is a wide-coverage grammar: it is designed to analyze sentences of unrestricted Dutch text. The grammar is based on the OVIS grammar (van Noord et al. 1999), that was used in the Dutch public transportation information system, but both lexicon and grammar have been extensively modified and extended.

The lexicon contains about 100,000 entries at this moment. More than 130 different verbal subcategorization frames are distinguished. Lexical information from the lexical databases Celex (Baayen, Piepenbrock and van Rijn 1993), Parole[2] and CGN (Groot 2000) was used to construct the lexicon. Various unknown word heuristics further extend the lexical coverage of the system.

The grammar should not only cover the question and answer patterns of the public transportation information system, but, in principle, all Dutch syntactic constructions. Therefore the grammar has been greatly extended. At this moment it consists of about 335 rules. These rules are linguistically motivated and describe both the common and the more specific, complex constructions such as verb-raising constructions and cleft sentences. The rules are written in a framework that is based on Head-Driven Phrase Structure Grammar (Pollard and Sag 1994; Sag 1997). Following Sag (1997), we have defined construction specific rules in terms of more general structures and principles.

In the lexicon, each word is assigned a type from a small set of basic lexical types. This type, e.g. *noun* for the word *tafel* (table), specifies the set of lexical features the word has. Nouns for instance have an agreement feature and a feature NFORM, that distinguishes regular nouns from temporal or reflexive nouns. A complementizer in contrast doesn't have those features, but is specified for CTYPE (i.e. complementizer type). These lexical features are represented in feature structures. Fig. 1 shows the feature structures for the word *tafel*, which is a lexical item (*ylex*), not derived from a verb (*ndev*) and not of any special class of nouns such as temporal or reflexive nouns (*norm*).

The feature DT is shared between all types. It contains information about the relations between a word and other words with which it can form a constituent. With the DT values of all words of a sentence a dependency tree is built.[3] This is a structure in which the various dependency relations between words and constituents in

[2] `http://www.inl.nl/corp/parole.htm`

[3] Strictly speaking, dependency trees are graphs, as control relations as well as certain long distance dependency relations are encoded using multiple dominance. The grammar as well as the treebank encode such multiple dominance relations by means of co-indexing of nodes.

$$
{}^{\textit{noun}}\begin{bmatrix}
\textsc{sc} & \langle\rangle \\
\textsc{deverbal} & \mathsf{ndev} \\
\textsc{agr} & \mathsf{sg\&thi\&de\&count} \\
\textsc{lex} & \mathsf{ylex} \\
\textsc{nform} & \mathsf{norm} \\
\textsc{dt} & \begin{bmatrix} \textsc{hwrd} & \mathsf{tafel} \\ \textsc{postag} & \mathsf{noun} \\ \textsc{det} & \langle\rangle \end{bmatrix}
\end{bmatrix}
$$

Figure 1: feature structure for lexical entry *tafel*

a sentence are expressed. More information about dependency structures is found in section 3.

Handwritten grammar rules define how lexical or phrasal items may combine to form larger units. The rules specify for each syntactic structure the type of the mother node, a head daughter and the non-head daughter(s). In addition, the type of structure that they constitute is specified. Almost all structures are headed structures (structures in which one of the daughters can be identified as the head daughter). The class of headed structures is further subdivided in head-complement, head-adjunct, head-filler and head-extra structures according to the function of the non-head daughter.

Furthermore, the grammar rules should specify how the lexical information on the daughter nodes is passed on to the mother node. For different types of features, different inheritance rules apply. It would be extremely time intensive, error prone and opaque if in each rule and for each feature the inheritance had to be specified separately. Therefore, five general principles are formulated that define how feature values are propagated up the tree. Each principle applies to a group of features. For instance, the *Head-feature Principle* states that for all features that are marked as *head features*, the values on the mother node are unified with the values on the *head* daughter. The *Valence, Filler, Extraposition* and *Adjunct and Dependency principle* define similar principles for the subcategorization, extraction, extraposition and modification and dependency tree features respectively. For each syntactic structure that is listed as a headed structure in the grammar, these general principles apply.

Lexical information, construction specific rules and general principles are the three basic components of the grammar. This setup allows the grammarian to formulate simple rules without specifying all the regular feature values on each of the components. The complete rules can be deduced from the simple ones through addition of the information that is conveyed in the general principles. For example, the rule in (1-a) expands to the rule in (1-b), in which the inheritance of the values for VFORM (finite, infinite, participle), subcategorization frame (SC), long distance dependencies (SLASH and EXTRA) and the dependency relations (DT) is specified. In this rule, $\langle H|T\rangle$ is used to denote a list with head H and tail T, and $L\oplus M$ represents the concatenation of the two lists L and M.

(1) a. *head-complement structure:* $v \rightarrow np\ \underline{v}$

b.
$$
{}_{verb}\begin{bmatrix} \text{VFORM} & \boxed{1} \\ \text{SC} & \boxed{2} \\ \text{SLASH} & \boxed{3} \\ \text{EXTRA} & \boxed{4} \oplus \boxed{5} \\ \text{DT} & \boxed{6} \end{bmatrix} \rightarrow \boxed{7}\, {}_{noun}\begin{bmatrix} \text{EXTRA} & \boxed{4} \end{bmatrix} \; {}_{verb}\begin{bmatrix} \text{VFORM} & \boxed{1} \\ \text{SC} & \left\langle \boxed{7} \mid \boxed{2} \right\rangle \\ \text{SLASH} & \boxed{3} \\ \text{EXTRA} & \boxed{5} \\ \text{DT} & \boxed{6} \end{bmatrix}
$$

3 Dependency Trees

The meaning of a word or a sentence is represented in standard HPSG by semantic representations that are added to lexical entries and phrases. Semantic principles define the construction of a semantic structure from these representations. In Alpino we have added the DT features with which we build a dependency tree instead.

Dependency structures represent the grammatical relations that hold in and between constituents. On the one hand they are more abstract than syntactic trees (word order for example is not expressed) and on the other hand they are more explicit about the dependency relations. Indices denote that constituents may have multiple (possibly different) dependency relations with different words. Fig. 2 shows the dependency tree for the sentence *Kim wil weten of Anne komt*. The dependency relations are the top labels in the boxes. In addition, the syntactic category, lexical entry and string position are added to each leaf. The index **1** indicates that *Kim* is the subject of both *wil* (wants) and *weten* (to know).

The main advantage of this format is that it is relatively theory independent, which is important in a grammar engineering context. A second advantage is that the format is similar to the format CGN uses (and that they in turn based on the Tiger Treebank), which allowed us to base our annotation guidelines on theirs (Moortgat, Schuurman and van der Wouden 2001). The third and last argument for using dependency structures is that it is relatively straightforward to perform evaluation of the parser on dependency structures: one can compare the automatically generated dependency structure with the one in the treebank and calculate statistical measures such as F-score based on the number of dependency relations that are identical in both trees (Carroll, Briscoe and Sanfilippo 1998).

4 The annotation process

The annotation process is roughly divided into two parts: we first parse a sentence with the Alpino parser and then select the best parse from the set of generated parses. Several tools that we have developed and implemented in Hdrug, a graphical environment for natural language processing (van Noord and Bouma 1997), facilitate the two parts of the annotation process. In section 4.1 we present an interactive lexical analyzer, a constituent marker and a tool for temporary addition of lexical information. The parse selection tool is described in in section 4.2.

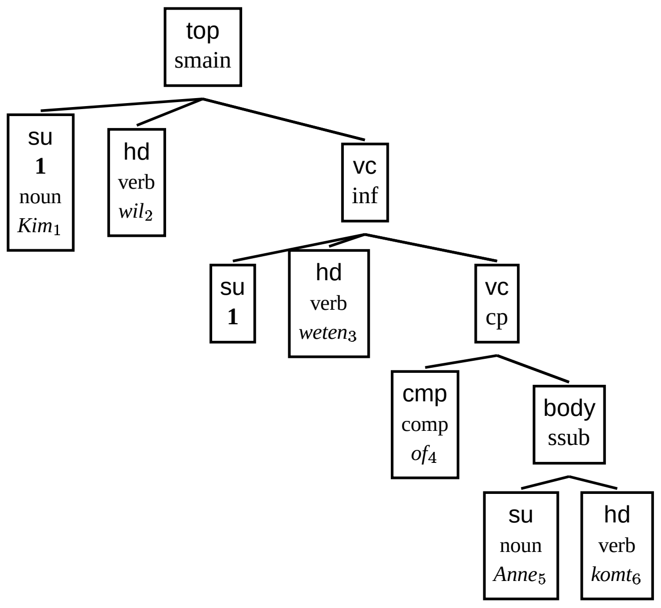

Figure 2: Dependency tree voor de zin *Kim wil weten of Anne komt*

4.1 Parsing

The annotation process typically starts with parsing a sentence from the corpus with the Alpino parser. This is a good method, since building up dependency trees manually is extremely time consuming and error prone. Usually the parser produces a correct or almost correct parse. If the parser cannot build a structure for a complete sentence, it tries to generate as large a structure as possible (e.g. a noun phrase or a complementizer phrase). The main disadvantage of parsing is that the parser produces a large set of possible parses (see fig.3). This is a well known problem in grammar development: the more linguistic phenomena a grammar covers, the greater the ambiguity per sentence. Because selection of the best parse from such a large set of possible parses is time consuming, we have tried to reduce the set of generated parses. The interactive lexical analyzer and the constituent marker restrict the parsing process which results in reduced sets of parses. A tool for on line addition of lexical information makes parsing of sentences with unknown words more accurate and efficient.

4.1.1 Interactive lexical analysis

The interactive lexical analyzer is a tool that facilitates the selection of lexical entries for the words in a sentence. It presents all possible lexical entries for all words in the sentence to the annotator. He or she may mark them as correct, good

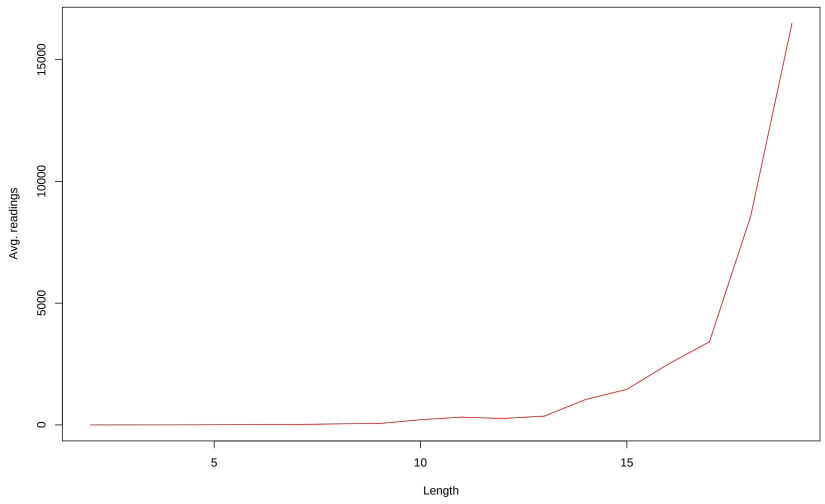

Figure 3: Number of parses generated per sentence by the Alpino parser

or bad.

- Correct Parse must include it
- Good Parse may include it
- Bad Parse may not include it

One *correct* mark for a particular lexical entry automatically produces *bad* marks for all other entries for the same word. The parser uses the reduced set of entries to generate a significantly smaller set of parses in less processing time.

4.1.2 Constituent Marking

The annotator can mark a piece of the input string as a constituent by putting square brackets around the words. The type of constituent can be specified after the opening bracket. The parser will only produce parses that have a constituent of the specified type at the string position defined in the input string. Even if the parse cannot generate the correct parse, it will produce parses that are likely to be close to the best possible parse, because they do oblige to the restrictions posed on the parses by the constituent marker.

Constituent marking has some limitations. First, the specified constituent borders are defined on the syntactic tree, not the dependency tree (dependency structures are an extra layer of annotation that is added to the syntactic structure). Using the tool therefore requires knowledge of the Alpino grammar and the syntactic trees that it generates.

Second, specification of the constituent type is necessary in most cases, especially for disambiguating prepositional phrase attachments. As shown in fig. 4, a

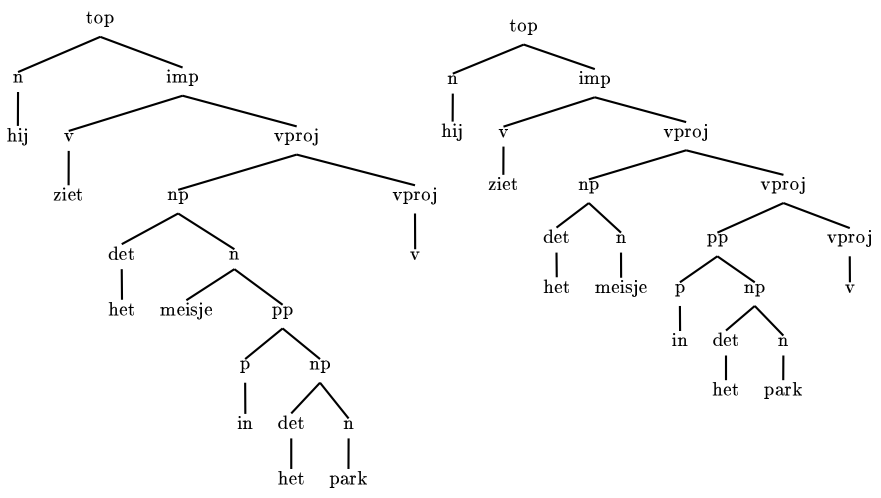

Figure 4: PP attachment ambiguity in Alpino

noun phrase and a prepositional phrase can form a constituent on different levels. The two phrases can form either a noun phrase or a verbal projection with an empty verb (which is used in the grammar to account for verb second). The first structure corresponds to a dependency structure with a noun phrase internal prepositional modifier, the second corresponds to a dependency tree in which the prepositional phrase is a modifier on the sentence level. Marking the string `het meisje in het park` as a constituent without further specification does not disambiguate between the two readings: in both readings the string is a constituent. One has to specify that the string should be a noun phrase, not a verbal projection. This specification of the constituent type requires even more knowledge of the grammar. If one specifies a constituent type that cannot be formed at the denoted string position, the parser treats the specification as an illegal character, skips it and generates partial parses only.

4.1.3 Addition of lexical information

Alpino is set up as a broad coverage parser. The goal is to build an analyzer of unrestricted text. Therefore a large lexicon has been created and extensive unknown word heuristics have been added to the grammar. Still, it is inevitable that the parser will come across unknown words that it cannot handle yet. Verbs are used with extra or missing arguments, Dutch sentences are mingled with foreign words, spelling mistakes make common words unrecognizable. In most cases, the parser will either skip such a word or assign an inappropriate category to it. The only way to make the system correctly use the word, is to add a lexical entry for it in the lexicon.

Adding new words to the lexicon costs time: one has to write the entry, save the

top:hd	=	v *wil*
top:su	=	n *Kim*
top:vc:hd	=	v *weet*
top:vc:su	=	n *Kim*
top:vc:vc:cmp	=	comp *of*
top:vc:vc:body:hd	=	v *kom*
top:vc:vc:body:su	=	n *Anne*

Figure 5: Set of dependency paths for the sentence *Kim wil weten of Anne komt*

new lexicon and reload it. It would be far more efficient to add all new words one comes across during an annotation session at once, avoiding spurious reloadings. Furthermore, not all unknown words the parser finds should be added to the lexicon. One would want to use misspelled words and verbs with an incorrect number of arguments only once to build a parse with.

Alpino has temporary, on line addition of lexical information built in for this purpose. Unknown words can temporarily be added to the lexicon with the command `add_tag` or `add_lex`. Like the words in the lexicon, this new entry should be assigned a feature structure. `add_tag` allows the user to specify the lexical type as the second argument. However types may change and especially for verbs it is sometimes hard to decide which of the subcategorization frames should be used. For that reason the command `add_lex` allows us to assign to unknown words the feature structure of a similar word, that could have been used on that position. The command `add_lex stoel tafel` for instance assigns the feature structure of fig. 1 to the word *stoel*. The command `add_lex zoen slaap` assigns *zoen* all feature structures of *slaap*, including imperative and 1st person singular present for all sub-categorization frames of *slapen*. The lexical information is automatically deleted when the annotation session is finished.

4.2 Selection

Although the number of parses that is generated is strongly reduced through the use of different tools, the parser usually still produces a set of parses. Selection of the best parse (i.e. the parse that needs the least editing) from this set of parses is facilitated by the parse selection tool. This design of this tool is based on the SRI Treebanker (Carter 1997).

The parse selection takes as input a set of dependency paths for each parse. A dependency path specifies the grammatical relation of a word in a constituent (e.g. head (hd) or determiner (det)) and the way the constituent is embedded in the sentence. The representation of a parse as a set of dependency paths is a notational variant of the dependency tree. The set of dependency triples that corresponds to the dependency tree in fig. 2 is in fig. 5.

From these sets of dependency paths the selection tool computes a (usually

s:hd	=	v *zag*	s:hd	=	v *zag*
*s:su	=	np *jan*	*s:su	=	np *het meisje*
*s:obj1	=	np *het meisje*	s:su:det	=	det *het*
s:obj1:det	=	det *het*	s:su:hd	=	n *meisje*
s:obj1:hd	=	n *meisje*	*s:obj1	=	np *jan*

Figure 6: Two readings of the sentence *Jan zag het meisje* represented as sets of dependency paths. An '*' indicates a maximal discriminant

much smaller) set of maximal discriminants. This set of maximal discriminants consists of the triples with the shortest dependency paths that encode a certain difference between parses. In example 6 the triples *s:su:det = det het* and *s:su = np het meisje* always co-occur, but the latter has a shorter dependency path and is therefore a maximal discriminant. Other types of discriminants are lexical and constituent discriminants. Lexical discriminants represent ambiguities that result form lexical analysis, e.g. a word with an uppercase first letter can be interpreted as either a proper name or the same word without the upper case first letter. Constituent discriminants define groups of words as constituents without specifying the type of the constituent.

The maximal discriminants are presented to the annotator, who can mark them as either good (parse must include it) or bad (parse may not include it). The parse selection tool then automatically further narrows down the possibilities using four simple rules of inference. This allows users to focus on discriminants about which they have clear intuitions. Their decisions about these discriminants combined with the rules of inference can then be used to make decisions about the less obvious discriminants.

1. If a discriminant is bad, any parse which includes it is bad
2. If a discriminant is good, any parse which doesn't include it is bad
3. If a discriminant is only included in bad parses, it must be bad
4. If a discriminant is included in all the undecided parses, it must be good

The discriminants are presented to the annotator in a specific order to make the selection process more efficient. The highest ranked discriminants are always the lexical discriminants. Decisions on lexical discriminants are very easy to make and greatly reduce the set of possibilities.

After this the discriminants are ranked according to their power: the sum of the number of parses that will be excluded after the discriminant has been marked *bad* and the number of parses that will be excluded after it has been marked *good*. This way the ambiguities with the greatest impact on the number of parses are resolved first.

The parse that is selected is stored in the treebank. If the best parse is not fully correct yet, it can be edited in the Thistle (Calder 2000) tree editor and then stored again. A second annotator checks the structure, edits it again if necessary and stores it afterwards.

5 Querying the treebank

The results of the annotation process are stored in XML. XML is widely in use for storing and distributing language resources, and a range of standards and software tools are available which support creation, modification, and search of XML documents. Both the Alpino parser and the Thiste editor output dependency trees encoded in XML.

As the treebank grows in size, it becomes increasingly interesting to explore it interactively. Queries to the treebank may be motivated by linguistic interest (i.e. which verbs take inherently reflexive objects?) but can also be a tool for quality control (i.e. find all PPs where the head is not a preposition).

The XPath standard[4] implements a powerful query language for XML documents, which can be used to formulate queries over the treebank. XPath supports conjunction, disjunction, negation, and comparison of numeric values, and seems to have sufficient expressive power to support a range of linguistically relevant queries. Various tools support XPath and can be used to implement a query-tool. Currently, we are using a C-based tool implemented on top of the LibXML library.[5]

The XML encoding of dependency trees used by Thistle (and, for compatibility, also by the parser) is not very compact, and contains various layers of structure that are not linguistically relevant. Searching such documents for linguistically interesting patterns is difficult, as queries tend to get verbose and require intimate knowledge of the XML structure, which is mostly linguistically irrelevant. We therefore transform the original XML documents into a different XML format, which is much more compact (the average filesize reduces with 90%) and which provides maximal support for linguistic queries.

As XML documents are basically trees, consisting of elements which contain other elements, dependency trees can simply be represented as XML documents, where every node in the tree is represented by an element `node`. Properties are represented by attributes. Terminal nodes (leaves) are nodes which contain no daughter elements. The XML representation of (the top of) the dependency tree given in figure 2 is given in figure 7.

The transformation of dependency trees into the format given in figure 7 is not only used to eliminate linguistically irrelevant structure, but also to make explicit information which was only implicitly stored in the original XML encoding. The indices on root forms that were used to indicate their string position are removed and the corresponding information is added in the attributes `start` and `end`. Apart from the root form, the inflected form of the word as it appears in

[4]`www.w3.org/TR/xpath`
[5]`www.xmlsoft.org/`

```
<node rel="top" cat="smain" start="0" end="6" hd="2">
  <node rel="su" pos="noun" cat="np" index="1"
        start="0" end="1" hd="1" root="Kim" word="Kim"/>
  <node rel="hd" pos="verb"
        start="1" end="2" hd="2" root="wil" word="wil"/>
  <node rel="vc" cat="inf" start="2" end="6" hd="3">
     ....
  </node>
</node>
```

Figure 7: XML encoding of dependency trees.

the annotated sentence is also added. Words are annotated with part of speech (`pos` information, whereas phrases are annotated with category (`cat`) information. A drawback of this distinction is that it becomes impossible to find all NPs with a single (non-disjunctive) query, as phrasal NPs are `cat="np"` and lexical NPs are `pos="noun"`. To overcome this problem, category information is added to non-projecting (i.e. non-head) leaves in the tree as well. Finally, the attribute `hd` encodes the string position of the lexical head of every phrase. The latter information is useful for queries involving discontinuous consituents. In those cases, the start and end positions may not be very informative, and it can be more interesting to be able to locate the position of the lexical head.

We now present a number of examples which illustrate how XPath can be used to formulate various types of linguistic queries. Examples involving the use of the `hd` attribute can be found in Bouma and Kloosterman (2002).

Objects of prepositions are usually of category NP. However, other categories are not completely excluded. The query in (2) finds the objects within PPs.

(2) `//node[@cat="pp"]/node[@rel="obj1"]`

The double slash means we are looking for a matching element anywhere in the document (i.e. it is an ancestor of the top element of the document), whereas the single slash means that the element following it must be an immediate daughter of the element preceding it. The `@`-sign selects attributes. Thus, we are looking for nodes with dependency relation `obj1`, immediately dominated by a node with category `pp`. In the current state of the dependency treebank, 98% (5,892 of 6,062) of the matching nodes are regular NPs. The remainder is formed by relative clauses (*voor wie het werk goed kende, for who knew the work well*), PPs (*tot aan de waterkant, till on the waterfront*), adverbial pronouns (see below), and phrasal complements (*zonder dat het een cent kost, without that it a penny costs*).

The CGN annotation guidelines distinguish between three possible dependency relations for PPs: complement, modifier, or 'locative or directional complement' (a more or less obligatory dependent containing a semantically meaningful preposition which is not fixed). Assigning the correct dependency relation is difficult, both for the computational parser and for human annotators. The following query finds the head of PPs introducing locative dependents:

(3)
```
//node[@rel="hd" and ../@cat="pp" and ../@rel="ld"]
```

Here, the double dots allow us to refer to attributes of the dominating XML element. Thus, we are looking for a node with dependency relation *hd*, which is *dominated* by a PP with a *ld* dependency relation. Here, we exploit the fact that the mother node in the dependency tree corresponds with the immediately dominating element in the XML encoding as well.

Comparing the list of matching prepositions with a general frequency list reveals that about 6% of the PPs are locative dependents. The preposition *naar* (*to, towards*) typically introduces locative dependents (50% (74 out of 151) of its usage), whereas the most frequent preposition (i.e. *van, of*) does introduce a locative in only 1% (15 out of 1496) of the cases.

In PPs containing an impersonal pronoun like *er* (*there*), the pronoun always precedes the preposition. The two are usually written as a single word (*eraan, there-on*). A further peculiarity is that pronoun and preposition need not be adjacent (*In Delft wordt* **er** *nog* **over** *vergaderd In Delft, one still talks about it*). The following query finds such discontinuous phrases:

(4)
```
//node[@cat="pp" and
./node[@rel="obj1"]/@end < ./node[@rel="hd"]/@start ]
```

Here, the '$<$'-operator compares the value of the end position of the object of the PP with the start position of the head of the PP. If the first is strictly smaller than the second, the PP is discontinuous. The corpus contains 133 discontinuous PPs containing an impersonal pronoun vs. almost 322 continuous pronoun-preposition combinations, realized as a single word, and 17 cases where these are realized as two words. This shows that in almost 25% of the cases, the preposition + impersonal pronoun construction is discontinuous.

6 Evaluation and training

The treebank is primarily constructed to evaluate the performance of the parser. We do this by comparing the dependency structures that the parser generates to the dependency structures that are stored in the treebank. For this purpose we do not use the representation of dependency structures as trees, but the alternative notation as sets of dependency paths that we already saw in the previous section. Comparing these sets we can count the number of relations that are identical in both the best parse that the system generated and the stored structure. From these counts precision, recall and F-score ($2 * precision * recall/precision + recall$) can be calculated. We also use the measure *accuracy*, which is defined as follows:

$$\textit{accuracy}=1-\frac{D_f}{max(D_t,D_s)}$$

D_s is the set of dependency relations of the best parse that the system generated. D_t is the set of dependency relations of the parse that is stored in the treebank. D_f is the number of incorrect or missing relations in D_s.

The annotated corpus is also used to train the stochastic module of the Alpino grammar that is used to rank the various parses for an example sentence according to their probability. This ranking is done in two steps: first, we construct a model of what a ”best parse” is. For this step, the annotated corpus is of crucial importance. Second, we evaluate parses of previously unseen sentences by this model and select as most probable parse the parse that best suits the constraints for ”best parse”.

The model for the probability of parses is based on the probabilities of features. These features should not be confused with the features in an HPSG feature structure. The features in this stochastic parsing model are chosen by the grammarian and in principle they can be any characteristic of the parse that can be counted. Features that we use at present are grammar rules, dependency relations and unknown word heuristics. We calculate the frequencies of the features in our corpus and assign weights to them proportional to their probability. This is done in the first step, the training step. In the second step, evaluation of a previously unseen parse, we count for each feature the number of times that it occurs in the parse and multiply that by its weight. The sum of all these counts is a measure for the probability of this parse. We will now describe in more detail the Maximum Entropy model that we use for stochastic parsing (Johnson et al. 1999), first focusing on the training step and then turning to parse evaluation.

The training step of the maximum entropy model consists of the assignment of weights to features. These weights are based on the probabilities of those features. To calculate these probabilities, we need a stochastic training set. We generate such a training set by first parsing each sentence in the corpus using the Alpino parser. The dependency structures of the parses that are generated by the parser (also the incorrect ones) are compared to the correct one in the corpus and evaluated following the above described evaluation method. The parses are then assigned a frequency proportional to the evaluation score.

Given the set of features (characteristics of parses) and the stochastic training set, we can calculate which features are likely to be included in a parse and which features are not. This tendency can be represented by assigning weights to the features. A large positive weight denotes a preference for the model to use a certain feature, whereas a negative weight denotes a dispreference. Various algorithms exist that guarantee to find the global optimal settings for these weights so that the probability distribution in the training set is best represented (Malouf 2002).

Once the weights for the features are set, we can use them in the second step: parse evaluation. In this step we calculate the probability of a parse for a new, previously unseen, sentence. In maximum entropy modeling, the probability of a parse *x* given sentence *y* is defined as

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

The number of times feature *i* with weights λ_i occurs in a parse is denoted by f_i. For each parse the normalization factor Z(*x*) is the same. Since we only want to calculate which parse is the most likely one and we do not need to know the precise probability of each parse, we only have to maximize

$$\sum_i \lambda_i f_i(y)$$

The accuracy of this model depends primarily on two factors: the set of features that is used and the size of the training set (see for instance Mullen 2002). Therefore it is important to expand the Alpino Dependency Treebank in order to improve the accuracy.

7 Conclusions

A treebank is very important for both evaluation and training of a grammar. For the Alpino parser, no suitable treebank existed. For that reason we have started to develop the Alpino Dependency Treebank by annotating a part of the Eindhoven corpus with dependency structures. As the treebank grows in size, it becomes more and more attractive to use it for linguistic exploration as well, and we have developed an XML format which supports a range of linguistic queries.

To facilitate the time consuming annotation process, we have developed several tools: interactive lexical analysis and constituent marking reduce the set of parses that is generated by the Alpino parser, the tool for addition of lexical information makes parsing of unknown words more efficient and the parse selection tool facilitates the selection of the best parse from a set of parses. In the future, constituent marking could be made more user friendly. We could also look into ways of further reducing the set of maximal discriminants that is generated by the parse selection tool.

The treebank currently contains over 6,000 sentences.[6] Much effort will be put in extending the treebank to at least the complete cdbl newspaper part of the Eindhoven corpus, which contains 7,150 sentences.

References

Baayen, R. H., Piepenbrock, R. and van Rijn, H.(1993), *The CELEX Lexical Database (CD-ROM)*, Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA.

Bouma, G. and Kloosterman, G.(2002), Querying Dependency Treebanks in XML, *Proceedings of the Third international conference on Language Resources and Evaluation (LREC)*, Gran Canaria, Spain, pp. 1686–1691.

Calder, J.(2000), Thistle and interarbora, *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, Saarbrücken, pp. 992–996.

[6]Available on `http://www.let.rug.nl/~vannoord/trees`

Carroll, J., Briscoe, T. and Sanfilippo, A.(1998), Parser evaluation: A survey and a new proposal, *Proceedings of the first International Conference on Language Resources and Evaluation (LREC)*, Granada, Spain, pp. 447–454.

Carter, D.(1997), The treebanker: A tool for supervised training of parsed corpora, *Proceedings of the ACL Workshop on Computational Environments For Grammar Development And Linguistic Engineering*, Madrid.

Groot, M.(2000), Lexiconopbouw: microstructuur. Internal project report Corpus Gesproken Nederlands, see lands.let.kun.nl/cgn.

Johnson, M., Geman, S., Canon, S., Chi, Z. and Riezler, S.(1999), Estimators for stochastic ”unification-based” grammars, *Proceedings of the 37th Meeting of the ACL*, College Park, Maryland, pp. 535–541.

Malouf, R.(2002), A comparison of algorithms for maximum entropy parameter estimation, *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002)*, Taipei.

Moortgat, M., Schuurman, I. and van der Wouden, T.(2001), CGN syntactische annotatie. Internal project report Corpus Gesproken Nederlands, see lands.let.kun.nl/cgn.

Mullen, T.(2002), *An investigation into Compositional Features and Feature Merging for Maximum Entropy-Based Parse Selection*, PhD thesis, Rijksuniversiteit Groningen.

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

Pollard, C. and Sag, I.(1994), *Head-driven Phrase Structure Grammar*, University of Chigago / CSLI.

Sag, I.(1997), English relative clause constructions, *Jourmal of Linguistics* **33**(2), 431–484.

Skut, W., Krenn, B. and Uszkoreit, H.(1997), An annotation scheme for free word order languages, *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington,DC.

Uit den Boogaard, P.(1975), *Woordfrequenties in geschreven en gesproken Nederlands*, Oosterhoek, Scheltema & Holkema, Utrecht. Werkgroep Frequentieonderzoek van het Nederlands.

van Noord, G. and Bouma, G.(1997), Hdrug, A flexible and extendible development environment for natural language processing, *Proceedings of the EACL/ACL workshop on Environments for Grammar Development, Madrid*.

van Noord, G., Bouma, G., Koeling, R. and Nederhof, M.-J.(1999), Robust grammatical analysis for spoken dialogue systems, *Journal of Natural Language Engineering* **5**(1), 45–93.