Alpino: Wide-coverage Computational Analysis of Dutch

Gosse Bouma, Gertjan van Noord, and Robert Malouf

Alfa-informatica
Rijksuniversiteit Groningen

Abstract

Alpino is a wide-coverage computational analyzer of Dutch which aims at accurate, full, parsing of unrestricted text. We describe the head-driven lexicalized grammar and the lexical component, which has been derived from existing resources. The grammar produces dependency structures, thus providing a reasonably abstract and theory-neutral level of linguistic representation. An important aspect of wide-coverage parsing is robustness and disambiguation. The dependency relations encoded in the dependency structures have been used to develop and evaluate both hand-coded and statistical disambiguation methods.

1 Introduction

For English, tremendous progress has been made in the area of wide-coverage parsing of unrestricted text. Many of the proposed systems are statistical parsers, but systems based on a hand-written grammar exist as well. The aim of Alpino[1] is to provide computational analysis of Dutch with coverage and accuracy comparable to state-of-the-art parsers for English.

The Alpino grammar (described in more detail below) is a lexicalized grammar in the tradition of constructionalist Head-driven Phrase Structure Grammar (Pollard and Sag 1994, Sag 1997). The grammar consists of hand-written, linguistically motivated rules and lexical types. To evaluate the coverage and disambiguation component of the system, a testbench of syntactically annotated material is absolutely crucial. Given the current lack of such material for Dutch, we have started to annotate corpora with dependency structures. Dependency structures provide a convenient level of representation for annotation, and a fairly neutral representation for further processing. The annotation format is taken from the project *Corpus Gesproken Nederlands* (*Corpus of Spoken Dutch*) (Oostdijk 2000). The construction of dependency structures in the grammar and our treebanking efforts are described in section 4. Both the lexicalist nature of the Alpino grammar and the use of dependency structures imply that lexical items must be associated with detailed valency information. For the Alpino lexicon we have extracted this information from the Celex and Parole lexical databases (section 3).

In section 5 we describe Alpino's parsing architecture. Section 6 describes a variety of disambiguation strategies which have been integrated in Alpino. In addition, we report on a number of preliminary disambiguation experiments. We conclude with some remarks on future work.

[1]Alpino is being developed as part of the NWO PIONIER project *Algorithms for Linguistic Processing*, `www.let.rug.nl/~vannoord/alp`

2 Grammar

The Alpino grammar is an extension of the successful OVIS grammar (van Noord, Bouma, Koeling and Nederhof 1999, Veldhuijzen van Zanten, Bouma, Sima'an, van Noord and Bonnema 1999), a lexicalized grammar in the tradition of Head-driven Phrase Structure Grammar (Pollard and Sag 1994). 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 Sag (1997), by organizing rules in an inheritance hierarchy, the relevant linguistic generalizations can still be captured. The Alpino grammar currently contains over 100 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 36 verbal subcategorization types.

The formalism supports the use of recursive constraints over feature-structures (using delayed evaluation, van Noord and Bouma (1994)). This allowed us to incorporate an analysis of cross-serial dependencies based on argument-inheritance (Bouma and van Noord 1998) and a trace-less account of extraction along the lines of Bouma, Malouf and Sag (2001).

3 Lexical Resources

Accurate, wide-coverage parsing of unrestricted text requires a lexical component with detailed subcategorization frames. For lexicalist grammar formalisms, the availability of lexical resources which specify subcategorization frames is even more crucial. In HPSG, for instance, phrase structure rules rely on the fact that each head contains a specification of the elements it subcategorizes for. If such specifications are missing, the grammar will wildly overgenerate.

We have used two existing lexical databases (Celex and Parole) to create a wide-coverage lexicon with detailed subcategorization frames enriched with dependency relations. Celex (Baayen, Piepenbrock and van Rijn 1993) is a large lexical database for Dutch, with rich phonological and morphological information. For use within the CGN project, this database has been extended with dependency frames (Groot 2000). This version of the lexicon contains 11,800 verbal stems, with a total of 21,800 dependency frames. By far the most frequent frames are those for intransitive (4,100) and transitive (6,500) verbs. A fair number of frames occurs more than 100 times, but 300 of the 650 different dependency frame *types* in the database occur only once.

Dependency Frame	Overlap	Celex only	Parole only	Total
[SU:NP][OBJ1:NP]	1810	1211	240	3261
[SU:NP]	257	1697	42	1996
[SU:NP][PC:PP⟨*pform*⟩]	337	541	273	1151
[SU:NP][OBJ1:NP][PC:PP⟨*pform*⟩]	129	375	308	812
[SU:NP][VC:S⟨subordinate⟩]	103	136	103	342
[SUP:NP⟨het⟩][OBJ1:NP][SU:CP]	7	247	5	259
[SU:NP][OBJ2:NP][OBJ1:NP]	65	171	28	264
[SU:NP][SE:NP][PC:PP⟨*pform*⟩]	65	62	102	229
[SU:NP][SE:NP]	49	137	65	251
[SU:NP][VC:VP]	10	16	37	63

Table 1: Dependency Frames and the number of stems occurring with this frame in both resources, in CGN/Celex only, in Parole only, and the total number of stems with this dependency frame in the Alpino Lexicon.

The Dutch Parole lexicon[2] comes with detailed subcategorization information, including dependency relations. The Parole lexicon is smaller than Celex, with 3,200 verbal stems and a total of 5000 dependency frames. There are 320 different dependency frame types, 190 of which occur only once.

Dependency frames for the Alpino lexicon have been constructed using the dependency information provided by CGN/Celex, Parole, and by entering definitions by hand. The latter has been done mostly for auxiliary and modal verbs: a small class of high-frequent elements which are exceptional in a number of ways. The CGN/Celex dictionary is very large. As the Celex database comes with frequency information, we currently only include those lexical items whose frequency is above a certain threshold. For verbal stems, this means that roughly 50% of the stems in Celex is included in the Alpino lexicon. All verbal stems from the Parole lexicon with a dependency frame covered by the grammar are included.

Currently, for 28 different CGN/Celex dependency frames a definition in the grammar has been provided. This covers over 80% of the verbal dependency frames in the CGN/Celex database, 10,400 of which are sufficiently frequent to be included in the Alpino lexicon. For 15 different dependency frames in the Parole lexicon a definition in Alpino is present. Using these, we extract over 4,100 dependency frames (82% of the total number of dependency frames in the Parole database). An overview of overlap and non-overlap for the most frequent frames extractable from both sources is given in table 1. For transitive and intransitive verbs, we see that over 85% of the stems in Parole are present in CGN/Celex as well. For most other dependency frames, however, the overlap is generally much smaller, and a significant portion of the stems present in Parole is not present in

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

Celex. This suggests that, for more specific subcategorization frames, both resources are only partially complete, and that not even the union of both provides exhaustive coverage.[3]

4 Dependency Structures

Within the CGN-project (Oostdijk 2000), guidelines have been developed for syntactic annotation of spoken Dutch (Moortgat, Schuurman and van der Wouden 2000), using dependency structures similar to those used for the German Negra corpus (Skut, Krenn and Uszkoreit 1997).

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 (1) 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 in (1).

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

The Alpino grammar produces dependency structures compatible with the CGN-guidelines. We believe this is a useful output format for a number of reasons. First of all, annotating a text with dependency structures is relatively straightforward and independent of the particular grammatical framework assumed. Thus, a dependency treebank can be used to debug and test various versions of the Alpino grammar. Second, as we adopt the CGN-guidelines, a considerable amount of annotated material will be available within the near future which can be used for development and testing. Third, it has been suggested that dependency relations provide a convenient level of representation for evaluation of computational grammar based on radically different grammatical theories (Carroll, Briscoe and Sanfilippo 1998). Finally, statistics for dependency relations between head words can be used to develop accurate models for parse-selection (Collins 1999); preliminary experiments are described in section 6.

Grammatical Construction of Dependency Structures. To produce dependency structures with the Alpino grammar, a new level of representation has been added to the grammar. The attribute DT dominates a dependency structure, with attributes for the lexical head (HD) and the various dependents. The value of a dependent attribute can be a dependency structure or a leaf node consisting of a

[3]The less frequent verb stems in Celex (currently not included in Alpino) are almost exclusively assigned the intransitive or transitive dependency frame.

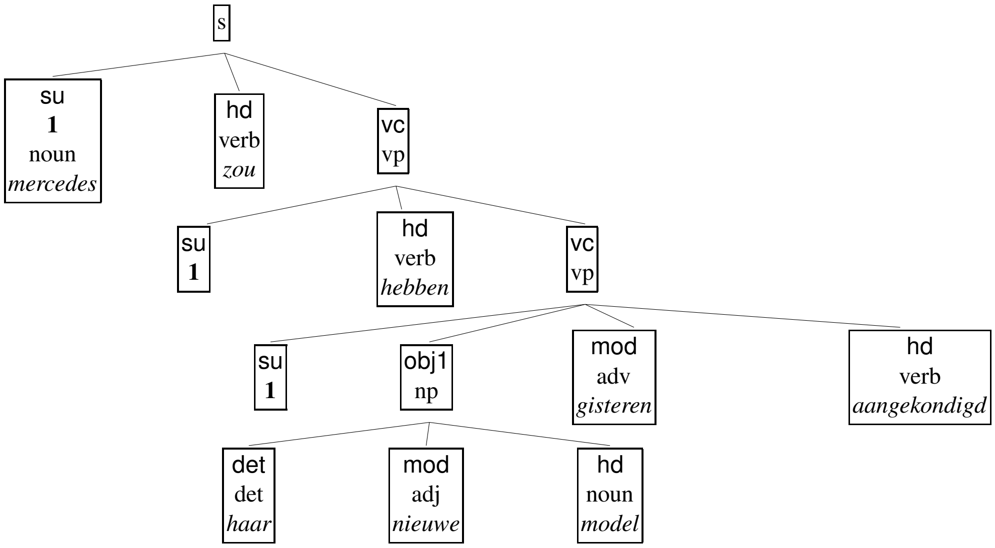

Figure 1: Dependency structure for example (1).

$$
\begin{bmatrix}
\textit{verb} & \\
\text{phon} & \boxed{1} \\
\text{subcat} & \left\langle \begin{bmatrix} \textit{np} & \\ \text{case} & \text{nom} \\ \text{dt} & \boxed{2} \end{bmatrix}, \begin{bmatrix} \textit{np} & \\ \text{case} & \text{acc} \\ \text{dt} & \boxed{3} \end{bmatrix} \right\rangle \\
\text{dt} & \begin{bmatrix} \text{hd} & \begin{bmatrix} \text{pos} & \text{verb} \\ \text{word} & \boxed{1} \end{bmatrix} \\ \text{su} & \boxed{2} \\ \text{obj1} & \boxed{3} \end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\textit{verb} & \\
\text{phon} & \boxed{1} \\
\text{subcat} & \left\langle \begin{bmatrix} \textit{np} & \\ \text{case} & \text{nom} \\ \text{dt} & \boxed{2} \end{bmatrix}, \begin{bmatrix} \textit{np} & \\ \text{case} & \text{dat} \\ \text{dt} & \boxed{3} \end{bmatrix} \right\rangle \\
\text{dt} & \begin{bmatrix} \text{hd} & \begin{bmatrix} \text{pos} & \text{verb} \\ \text{word} & \boxed{1} \end{bmatrix} \\ \text{su} & \boxed{2} \\ \text{obj2} & \boxed{3} \end{bmatrix}
\end{bmatrix}
$$

Figure 2: Schematic lexical entry for transitive verbs taking a direct object (OBJ1), and for transitive verbs taking an indirect object (OBJ2).

POS-tag and word only.

The construction of dependency structures is driven by the lexicon. For each subcategorization type recognized in the lexical hierarchy a mapping between elements on the list-valued feature which specifies basic subcategorization properties (SUBCAT) and attributes of DT is defined. Two examples are given in figure 2. The leftmost feature structure exemplifies a finite, transitive verb. The value of DT of the nominative NP on subcat is identical to the value of the SU dependent. Similarly, the value of DT of the accusative NP on subcat is identical to the value of the OBJ1 dependent. The rightmost feature structure exemplifies a finite, transitive verb for which the object is assigned to the OBJ2 (secondary object) dependency relation. In some cases, the addition of dependency structures leads to more fine-grained distinctions. For instance, PP-arguments can be linked to PC (*prepositional complement*) or LD (*locative or directional complement*), where the distinction between these two is primarily semantic in nature. Therefore, verbs taking a prepositional complement are assigned a subcategorization frame that differs from the frame assigned to verbs taking such a LD complement.

In HEAD-COMPLEMENT structures, the DT attribute can simply be shared between head daughter and mother. In HEAD-MODIFIER structures, the dependency structure of the modifier is added to the list-valued MOD dependent of the head.

Dependency Treebanks. For development and evaluation purposes, we have started to annotate various sample text fragments with dependency structures.

The annotation process typically starts by parsing a sentence with the Alpino grammar. This produces a (often large) number of possible analyses. The annotator picks the analysis which best matches the correct analysis. To facilitate selection of the best parse among a large number of possibilities, the HDRUG environment has been extended with a graphical tool based on the SRI TreeBanker (Carter 1997) which displays all fragments of the input which are a source of ambiguity. By disambiguating these items (usually a much smaller number than the number of readings), the annotator can quickly pick the most accurate parse.

For example, the sentence *Jan zag het meisje* 'Jan saw the girl' has (in principle) two readings corresponding to the dependency structures in figure 3. The readings of a sentence are represented as a set of sets of dependency paths, as in figure 4. From these sets of paths, the parse selection tool computes a set of *maximal discriminants* which can be used to select among different analyses. In this case, the path 's:hd = v *zag*' is shared by all the analyses and so is not a useful discriminant. On the other hand, the path 's:obj1:hd = n *meisje*' does distinguish between the readings but it is not maximal, since it is subsumed by the path 's:obj1 = np *het meisje*' which is shorter and makes exactly the same distinctions. The maximal discriminants are presented to the annotator, who may mark any of them as either good (the correct parse must include it) or bad (the correct parse may not include it). In this simple example, marking any one of the maximal discriminants as good or bad is sufficient to uniquely identify the correct parse. For more complex sentences, several choices will have to be made to select a single best parse. To help the annotator, when a discriminant is marked as bad or good, the following

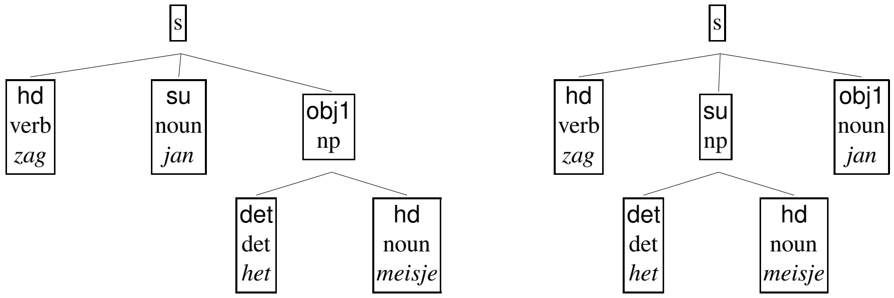

Figure 3: Dependency structures for two readings of *Jan zag het meisje*.

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 4: Dependency paths for *Jan zag het meisje* (* indicates a maximal discriminant).

inference rules are applied to further narrow the possibilities (Carter 1997):

- If a discriminant is bad, any parse which includes it is bad.
- If a discriminant is good, any parse which does not include it is bad.
- If a discriminant is only included in bad parses, it must be bad.
- If a discriminant is included in all the undecided parses, it must be good.

This allows users to focus their attention on discriminants about which they have clear intuitions. Their decisions about these discriminants combined with the rules of inference can then be used to automatically make decisions about less obvious discriminants.

If the parse selected by the annotator is fully correct, the dependency structure for that parse is stored as XML in the treebank. If the best parse produced by the grammar is not the correct parse as it should be included in the treebank, the dependency structure for this parse is sent to the Thistle editor.[4] The annotator can now produce the correct parse manually.

We have started to annotate various smaller fragments using the annotation tools described above. The largest fragments consist of two sets of sentences ex-

[4]LT Thistle (Calder 2000), `www.ltg.ed.ac.uk/software/thistle/`, is an editor and display engine for linguistic data-structures which supports XML.

tracted from the Eindhoven corpus (Uit den Boogaart 1975). The CDBL10 treebank currently consists of the first 519 sentences of ten words or less from section CDBL (newspaper text). The CDBL20 treebank consists of the first 252 sentences with more than 10 but no more than 20 words.

Evaluation. Evaluation of coverage and accuracy of a computational grammar usually is based on some metric which 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, Carroll et al. (1998) propose to annotate sentences with triples of the form ⟨*head-word, dependency relation, dependent head-word*⟩. For instance, for the example in (1) we might obtain:

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

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.

5 Robust Parsing

The initial design and implementation of the Alpino parser is inherited from the system described in van Noord (1997), van Noord et al. (1999) and van Noord (2001). However, a number of improvements have been implemented which are described below.

The construction of a dependency structure on the basis of some input proceeds in a number of steps, described below. 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. For sentence (1), 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 (1), 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 conditions hold. For sentence (1), the filter removes 56 tags. After the filter has applied, feature structures are associated with each of these tags. Often, a single tag is mapped to multiple feature structures. The remaining 27 filtered tags give rise to 89 feature structures.

An important aspect of lexical analysis is the treatment of unknown words. The system applies a number of heuristics for unknown words. Currently, these heuristics attempt to deal with numbers and number-like expressions, capitalized words, words with missing diacritics, words with 'too many' diacritics, compounds, and proper names.

If such heuristics still fail to provide an analysis, then the system guesses a tag by inspecting the suffix of the word. A list of suffixes is maintained which predict the tag of a given word. If this still does not provide an analysis, then it is assumed that the word is a noun.

In addition to the treatment of unknown words, the robustness of the system is enhanced by the possibility to skip tokens of the input. Currently this possibility is employed only for certain punctuation marks. Even though punctuation is treated both in the lexicon and the grammar, the syntax of punctuation is irregular enough to warrant the possibility to ignore punctuation. For instance, quotation marks may appear almost anywhere in the input. The corpus contains:

(2)	De	z.g.	"	speelstraat	,	die	hier	en	daar	al	bestaat	?
	The	so-called	"	play-street	,	that	here	and	there	already	exists	?

Apparently, the author intended to place `speelstraat` within quotes, but the second quote is not present. During lexical analysis, tags are optionally extended to include neighbouring words which are classified as 'skipable'.

Creating Parse Forests. The Alpino parser takes the result of lexical analysis as its input, and produces a *parse forest*: a compact representation of all parse trees. The Alpino parser is a left-corner parser with selective memoization and goal-weaking. It is a variant of the parsers described in van Noord (1997). We generalized some of the techniques described there to take into account relational constraints, which are delayed until sufficiently instantiated (van Noord and Bouma 1994).

As described in van Noord et al. (1999) and van Noord (2001), the parser can be instructed to find all occurrences of the start category *anywhere in the input*. This feature is added to enhance robustness as well. 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 large chunks of the input. A best-first search procedure then picks out the best sequence of such chunks. Depending on the application, such chunks might be very useful. In the past, we successfully employed this strategy in a spoken dialogue system (Veldhuijzen van Zanten et al. 1999).

beam	cdbl10		cdbl20	
	accuracy (%)	speed (msec)	accuracy (%)	speed (msec)
1	79.99	190	73.63	740
2	80.66	270	74.59	1470
4	81.11	350	75.07	2350
8	81.22	530	75.35	3630
16	81.36	590	75.31	5460
32	81.36	790	74.98	7880
∞	81.36	640	-	-

Table 2: Effect of beam-size on accuracy and efficiency of parse selection

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 section 6 we describe some initial experiments with a variety of parse evaluation functions. 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. In order to be able to *guarantee* that this search procedure indeed finds the best parse tree, a certain monotonicity requirement should apply to this evaluation function: if a (partial) tree s is better than s', then a tree t which contains s should be better than t' which is just like t except it has s' instead of s. However, instead of relying on such a requirement, 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 risc 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. Currently, we find that a value of $b = 4$ is a good compromise between accuracy and efficiency. In table 2 the effect of various values for b is presented for two development treebanks. The grammar assigns on average about 33 parse trees per sentence for the `cdbl10` corpus. This number increases rapidly for longer sentences: for the `cdbl20` corpus it is at least 340.[5]

[5] This is the average number after creating all parse trees for each sentence with a maximum of 1000 parse trees per sentence.

6 Disambiguation

The best-first unpack strategy described in section 5 depends on a parse evaluation function which assigns scores to (partial) parse trees. We have experimented with a number of disambiguation techniques on the `cdbl10` and `cdbl20` development treebanks described earlier.

Penalty rules. The simplest disambiguation method consists of hand-written 'penalty' rules which implement a variety of preferences. Each such penalty rule describes a partial parse tree. For a given parse tree, the system computes how often a sub-tree matches with a penalty rule, giving rise to the total penalty of that parse. The following lists characterizes some of the penalty rules:

- complementation is preferred over modification
- subject topicalization is preferred over object topicalization
- long distance dependencies are dis-preferred
- certain rules are dis-preferred (e.g. rules which coordinate categories without an explicit coordinator)
- certain lexical entries are dis-preferred (e.g. the preposition readings for the words `aan, bij, in, naar, op, uit, voor, tussen` are preferred over the adjectival, noun and/or verb readings).
- certain guesses for unknown words are preferred over others

As can be concluded from the preliminary results presented in table 3, it appears to be the case that about 60% of the disambiguation problem can be solved using this very simple technique.

Dependency relations We also experimented with statistical models based on dependency relations encoded in the dependency structure. The model assigns a probality to a parse by considering each dependency relation. For this purpose, dependency relations d are 5-tuples $d = \langle w_h, p_h, r, w_a, p_a \rangle$ where w_h is the head word, p_h is the corresponding part-of-speech tag taken from a small set of part-of-speechs $\{v, n, a, adv, p, \ldots\}$, r is the name of the relation taken from a small set of relation names $\{su, obj1, obj2, vc, mod, det, \ldots\}$; w_a is the argument word, and p_a is its part of speech.

The probability of a parse y given a sentence x might then be defined as:

$$p(y|x) = \frac{1}{Z(x)} \prod_{d \in y} p(r, w_a, p_a | w_h, p_h)$$

For disambiguation, the normalizing factor $Z(x)$ is the same for every parse of a given sentence and can be ignored.

Due to the occurrence of reentrancies, dependency structures are generally not trees but graphs. Therefore, the product above gives poor results because it will have an unjustified bias against such reentrancies (a reentrancy gives rise to an additional dependency relation). For this reason, we have chosen to score parse trees by determining the *mean* value of $-\log p$ for each tuple; this improved results considerably. The probability of a dependency is calculated as follows:

$$p(r, w_a, p_a | w_h, p_h) = p(r|w_h, p_h) * p(p_a|w_h, p_h, r) * p(w_a|w_h, w_p, r, p_a)$$

The three components are each calculated using a linear back-off strategy, where the weights are determined by frequency and diversity (formula 2.66 of (Collins 1999)). The quantities we use for backing off are given in the following table:

back-off level	$p(r\|w_h, p_h)$	$p(p_a\|w_h, p_h, r)$	$p(w_a\|w_h, w_p, r, p_a)$
1	$p(r\|p_h)$	$p(p_a\|p_h, r)$	$p(w_a\|w_p, r, p_a)$
2	$p(r)$	$p(p_a\|r)$	$p(w_a\|r, p_a)$
3		$p(p_a)$	$p(w_a\|p_a)$
4			$p(w_a)$

Because the size of the treebanks we have currently available is much too small to estimate these quantities accurately, we have chosen to do our estimation using unsupervised learning. We have parsed a large corpus ('de Volkskrant' newspaper text: first four months of 1997) using the penalty rules described in the previous section as our disambiguator. This corpus contains about 350,000 sentences and 6,200,000 words. We only used those sentences that the system could analyse as a single constituent, and within a reasonable amount of time. This meant that we could use the results of about 225,000 sentences. We estimated the quantity p using the best parse (according to the penalty rules) for each of these sentences. Collecting the 225,000 dependency structures took about one month of CPU-time (using the high-performance computing cluster of the University of Groningen).

As can be concluded from table 3, such a model performs much better than the baseline. Moreover, a combined model in which we simply add the rule penalties to the quantity p performs better than either model in isolation.

Log-linear models. While the model described in the previous section offers good performance and conceptual simplicity, it is not without problems. In particular, the strategies for dealing with reentrancies in the dependency structures and for combining scores derived from penalty rules and from dependency relation statistics are ad hoc. Log-linear models, introduced to natural language processing by Berger, Della Pietra and Della Pietra (1996) and Della Pietra, Della Pietra and Lafferty (1997), and applied to stochastic constraint-based grammars by Abney (1997) and Johnson, Geman, Canon, Chi and Riezler (1999), offer the potential to solve both of these problems. Given a conditional log-linear model, the probability of a sentence x having the parse y is:

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

	cdbl10			cdbl20		
technique	precision	recall	f-score	precision	recall	f-score
baseline	62.3	63.3	62.8	58.5	59.6	59.0
log linear	76.0	76.6	76.3	66.3	67.6	66.0
penalties	78.6	79.3	78.9	73.1	73.3	73.2
dependency rel's	78.9	79.7	79.3	69.7	71.1	70.4
heur. + dep-rel's	80.9	81.7	81.3	74.6	75.4	75.0
maximum	89.1	90.0	89.6	83.2	84.1	83.7

Table 3: Preliminary results on the `cdbl10` and `cdbl20` development treebanks for a number of disambiguation techniques. The *baseline* row lists the percentages obtained if we select for each sentence a random parse tree from the parse forest. The *maximum* row lists the percentages obtained if we take for each sentence the best parse tree. These two numbers thus indicate the lower and upper bounds for parse selection.

As before, 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 $f_i(x,y)$. What makes log-linear models particularly well suited for this application is that the property functions may be sensitive to any information which might be useful for disambiguation. Possible property functions include syntactic heuristics, lexicalized and backed-off dependency relations, structural configurations, and lexical semantic classes. Using log-linear models, all of these disparate types of information may be combined into a single model for disambiguation. Furthermore, since standard techniques for estimating the weights λ_i from training data make no assumptions about the independence of properties, one need not take special precautions when information sources overlap.

The drawback to using log-linear models is that accurate estimation of the parameters λ_i requires a large amount of annotated training data. Since such training data is not yet available, we instead attempted unsupervized training from unannotated data. We used the Alpino parser to find all parses of the 82,000 sentences with ten or fewer words in the 'de Volkskrant' newpaper corpus. Using the resulting collection of 2,200,000 unranked parses, we then applied Riezler et al.'s (2000) 'Iterative Maximization' algorithm to estimate the parameters of a log-linear model with dependency tuples as described in the previous section as property functions. The results, given in table 3, show some promise, but the performance of the log-linear model does not yet match that of the other disambiguation strategies. Current work in this area is focused on expanding the set of properties and on using supervised training from what annotated data is available to bootstrap the unsupervised training from large quantities of newspaper text.

7 Conclusions

Alpino aims at providing a wide-coverage, accurate, computational grammar for Dutch. The linguistic component of the system consists of a lexicalist feature-based grammar for Dutch, a wide-coverage and detailed lexicon, and a method for constructing dependency treebanks. The parser contains a lexical analysis module and a method for reconstructing parses from a parse forest using beam search, which allows the linguistic knowledge to be applied efficiently and robustly to unrestricted text. Finally, we have presented preliminary experiments aimed at providing accurate disambiguation.

In the near future, we hope to address a number of additional issues. The valency information in the lexicon is in many ways incomplete. We hope to obtain a more complete lexicon by acquiring dependency frames from corpora. Lexical analysis currently uses hand-written filter rules to reduce the number of tags for lexical items. An obvious alternative is to use a corpus-based part-of-speech tagger to arrive at the relevant filters. Finally, the work on disambiguation can profit from the availability of more annotated material. This suggests that our efforts at creating a dependency treebank may lead to improved results in the future.

References

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

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

Berger, A., Della Pietra, S. and Della Pietra, V.(1996), A maximum entropy approach to natural language processing, *Computational Linguistics* **22**(1), 39–72.

Bouma, G. and van Noord, G.(1998), Word order constraints on verb clusters in German and Dutch, *in* E. Hinrichs, T. Nakazawa and A. Kathol (eds), *Complex Predicates in Nonderivational Syntax*, Academic Press, New York, pp. 43–72.

Bouma, G., Malouf, R. and Sag, I.(2001), Satisfying constraints on adjunction and extraction, *Natural Language and Linguistic Theory* **19**, 1–65.

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

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.

Collins, M.(1999), *Head-Driven Statistical Models for Natural Language Parsing*, PhD thesis, University Of Pennsylvania.

Della Pietra, S., Della Pietra, V. and Lafferty, J.(1997), Inducing features of random fields, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**, 380–393.

Groot, M.(2000), Lexiconopbouw: microstructuur. Internal report Corpus Gesproken Nederlands.

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

Moortgat, M., Schuurman, I. and van der Wouden, T.(2000), CGN syntactische annotatie. Internal report Corpus Gesproken Nederlands.

Oostdijk, N.(2000), The Spoken Dutch Corpus: Overview and first evaluation, *Proceedings of 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 Chicago / CSLI.

Riezler, S., Prescher, D., Kuhn, J. and Johnson, M.(2000), Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and em, *Proceedings of the 38th Annual Meeting of the ACL*, Hong Kong, pp. 480–487.

Sag, I.(1997), English relative clause constructions, *Journal 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 Boogaart, P. C.(1975), *Woordfrequenties in geschreven en gesproken Nederlands*, Oosthoek, Scheltema & Holkema, Utrecht. Werkgroep Frequentie-onderzoek van het Nederlands.

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

van Noord, G.(2001), Robust parsing of word graphs, *in* J.-C. Junqua and G. van Noord (eds), *Robustness in Language and Speech Technology*, Kluwer Academic Publishers, Dordrecht.

van Noord, G. and Bouma, G.(1994), Adjuncts and the processing of lexical rules, *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, Kyoto, pp. 250–256. cmp-lg/9404011.

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.

Veldhuijzen van Zanten, G., Bouma, G., Sima'an, K., van Noord, G. and Bonnema, R.(1999), Evaluation of the NLP components of the OVIS2 spoken dialogue system, *in* F. van Eynde, I. Schuurman and N. Schelkens (eds), *Computational Linguistics in the Netherlands 1998*, Rodopi Amsterdam, pp. 213–229.