Wide Coverage Parsing with Stochastic Attribute Value Grammars

Gertjan van Noord*
University of Groningen

Robert Malouf†
San Diego State University

Stochastic Attribute Value Grammars (SAVGs) provide an attractive framework for syntactic analysis, because they allow the combination of linguistic sophistication with a principled treatment of ambiguity. The paper introduces a wide-coverage SAVG for Dutch, known as Alpino, and we show how this SAVG can be efficiently applied, using a beam search algorithm to recover parses from a shared parse forest. Unlike previous approaches, this algorithm does not place strict locality restrictions on disambiguation. Experimental results for a number of different corpora suggest that the SAVG framework is applicable for realistically sized grammars and corpora.

1. Introduction

Alpino is a wide-coverage computational analyzer of Dutch which aims at full accurate parsing of unrestricted text, with coverage and accuracy comparable to state-of-the-art parsers for English. The SAVG framework is employed, because it allows linguistic sophistication in combination with a principled treatment of pervasive ambiguity. Building on models proposed by Abney (1997), Johnson et al. (1999), Osborne (2000), Riezler et al. (2002), among others, we show that the SAVG framework can be applied to realistically sized grammars and corpora.

A fundamental problem facing developers of natural language processing systems is that the grammatical constraints created by linguists admit structures in language which no human would recognize. For example, a sentence like *The tourist saw museums* sounds simple enough, but most NLP systems will recognize not only the intended meaning, but also the meaning in which *saw* is a noun, and the entire string is parsed as a determiner *the* followed by a compound noun *tourist saw museums.* This reading is nonsensical, but cannot be ruled out on purely structural grounds without also ruling out the parallel structure in *the circular saw blades.*

A typical architecture for disambiguation uses a probabilistic context free rule system, where estimates of rule probabilities are derived from the frequency with which rules have been encountered in collections of parses which have been disambiguated by hand. With a sufficient quantity of annotated training data and careful selection of stochastic features, such systems perform adequately enough on structural disambiguation tasks to support simple applications.

More sophisticated applications such as open-domain question answering or dialog systems, however, require more sophisticated grammar formalisms like Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994). Furthermore, as grammars become more comprehensive, parsers will find an ever larger number of potential readings for a sentence and effective disambiguation becomes even more important. Since these formalisms involve a more complex flow of information than simple context-free grammars, more complex statistical methods are required to capture the subtle dependencies

* Alfa-informatica PO Box 716 9700AS Groningen Netherlands
† Department of Linguistics and Oriental Languages, San Diego, CA 92182-7727 USA

among grammatical structures.

As Abney (1997) shows, the simple rule frequency methods applied to disambiguating context free parses cannot be used for disambiguating constraint grammar parses, since these methods rely crucially on the statistical independence of context-free rule applications. One solution is provided by Maximum Entropy models (variously also known as Gibbs, logistic, exponential, log-linear models, and multinomial logit models). Maximum entropy models (Berger, Della Pietra, and Della Pietra, 1996; Della Pietra, Della Pietra, and Lafferty, 1997) provide a general purpose machine learning technique for classification and prediction which has been successfully applied to fields as diverse as computer vision and econometrics. In natural language processing, recent years have seen these techniques used for sentence boundary detection, part of speech tagging, to name just a few applications, and they have proven particularly successful as a foundation for Stochastic Attribute Value Grammar formalisms (Abney, 1997; Johnson et al., 1999; Riezler et al., 2002).

A leading advantage of Maximum Entropy models is their flexibility: they allow stochastic rule systems to be augmented with additional syntactic, semantic, and pragmatic features. Suppose we want to construct a probability distribution p over a set of parses $x \in X$, which in turn are characterized by features $f_i(x)$. In the context of a stochastic context-free grammar (SCFG), for example, features $f_i(x)$ might represent the number of times rule i was applied in the derivation of parse x. For SAVGs, the features may be more general, and include in addition information about dependency relations and part of speech tags (see section 4).

Suppose further that we have access to a collection of annotated parses which gives us an empirical distribution $\tilde{p}(x)$ Our task is to construct a model for p which satisfies the constraints imposed by the training data, in the sense that:

$$\mathrm{E}_p[f] = \mathrm{E}_q[f] \tag{1}$$

where $E_p[f]$ is the expected value of the feature vector under the distribution p:

$$\mathrm{E}_p[f] = \sum_{x \in X} p(x) f(x)$$

In general, this problem is ill posed: a wide range of models will fit the constraints in (1). As a guide to selecting one that is most appropriate, we can call on Jaynes' (1957) *Principle of Maximum Entropy*: "In the absence of additional information, we should assume that all events have equal probability." In other words, we should assign the highest prior probability to distributions which maximize the entropy.

$$H(p) = -\sum_{x \in X} p(x) \log p(x) \tag{2}$$

The entropy (2) is a measure of the average surprise or uncertainty in the random variable X given the distribution p. The intuition behind Jaynes' use of entropy is simple: suppose we have two distributions p_1 and p_2 over X which both satisfy the constraints in (1), and $H(p_1) < H(p_2)$. That means that the uncertainty in X under p_1 is less than under p_2, so p_1 must embody some information about X which is not contained in p_2. However, by hypothesis, both distributions satisfy the constraints and so include all the information in the training data. Any 'extra' information in p_1 must have come from somewhere other than the training data, and thus p_1 must reflect some additional a priori assumptions. The Principle of Maximum Entropy is a way of enforcing Occam's Razor, in that it will prefer models which include the fewest constraints which are not directly imposed by the training data.

Finding a distribution which satisfies the Principle of Maximum Entropy is effectively a problem in constrained optimization: we want to find a distribution p which maximizes (2) while satisfying the constraints imposed by (1). It can be straightforwardly shown that the maximum entropy distribution has the parametric form:

$$p(x;\theta) = \frac{\exp\left(\sum_i \theta_i f_i(x)\right)}{\sum_{y \in X} \exp\left(\sum_i \theta_i f_i(y)\right)} \tag{3}$$

where θ is a d-dimensional parameter vector.

One complication which makes models of this form difficult to apply to problems in natural language processing is that the event space X is often very large or even infinite, making the denominator in (3) impossible to compute. One modification we can make to avoid this problem is to consider conditional probability distributions instead (Berger, Della Pietra, and Della Pietra, 1996; Johnson et al., 1999). Suppose now that in addition to the event space X and the feature functions f_i, we have also a set of contexts W and a function Y which partitions the members of X. In our SCFG example, W might be the set of possible strings of words, and $Y(w)$ the set of trees whose yield is $w \in W$. Computing the conditional probability $p(x|w;\theta)$ of an event x in context w as

$$p(x|w;\theta) = \frac{\exp\left(\sum_i \theta_i f_i(x)\right)}{\sum_{y \in Y(w)} \exp\left(\sum_i \theta_i f_i(y)\right)} \tag{4}$$

now involves evaluating a much more tractable sum in the denominator.

The value of $f_i(x)$ reflects the frequency of the ith feature in a given parse x. The parameters θ_i (which provide a weight for each feature) can be estimated efficiently by maximizing the pseudo-likelihood of a training corpus (Johnson et al., 1999):

$$L(\theta) = \sum_w \tilde{p}(w) \sum_{x \in Y(w)} \tilde{p}(x|w) \log p(x|w;\theta) \tag{5}$$

The empirical probabilities $\tilde{p}(w)$ and $\tilde{p}(x|w)$ are derived from the training data.

Given the parametric form of an maximum entropy model in (4), fitting an ME model to a collection of training data entails finding values for the parameter vector θ which minimize the Kullback-Leibler divergence between the model $p(x|w;\theta)$ and the empirical distribution $\tilde{p}(x|w)$:

$$D(\tilde{p}||p) = \sum_{w,x} \tilde{p}(x,w) \log \frac{\tilde{p}(x|w)}{p(x|w;\theta)}$$

or, equivalently, which maximize the log likelihood (5). The gradient of the log likelihood function, or the vector of its first derivatives with respect to the parameter θ is:

$$G(\theta) = \sum_{x,y} \tilde{p}(x,y) f(y) - \sum_{x,y} \tilde{p}(x) p(y|x;\theta) f(y)$$

Since the likelihood function (5) is concave over the parameter space, it has a global maximum where the gradient is zero. Unfortunately, simply setting $G(\theta) = 0$ and solving for θ does not yield a closed form solution, so we proceed iteratively. At each step, we adjust an estimate of the parameters $\theta^{(k)}$ to a new estimate $\theta^{(k+1)}$ based on the divergence between the estimated probability distribution $p^{(k)}$ and the empirical distribution $\tilde{p}$. We continue until successive improvements fail to yield a sufficiently large decrease in the divergence.

As a basis for our implementation, we have used PETSc (the "Portable, Extensible Toolkit for Scientific Computation"), a software library designed to ease development of programs which solve large systems of partial differential equations (Balay et al., 2002). PETSc offers data structures and routines for parallel and sequential storage, manipulation, and visualization of very large sparse matrices.

To solve the non-linear optimization required to maximize (5), we used TAO (the "Toolkit for Advanced Optimization"), a library layered on top of the foundation of PETSc for solving non-linear optimization problems (Benson et al., 2002). TAO offers the building blocks for writing optimization programs (such as line searches and convergence tests) as well as high-quality implementations of standard optimization algorithms (including the limited memory variable metric method which we use here).

For parameter estimation, the most expensive operation is computing the probability distribution q and the expectations $\mathrm{E}_p[f]$ for each iteration. In order to make use of the facilities provided by PETSc, we store the training data as a (sparse) matrix F, with rows corresponding to events and columns to features. Then given a parameter vector θ, the unnormalized log probabilities are the matrix-vector product $F\theta$ and the feature expectations are the transposed matrix-vector product $F^T p$. By expressing these computations as matrix-vector operations, we can take advantage of the high performance sparse matrix primitives of PETSc.

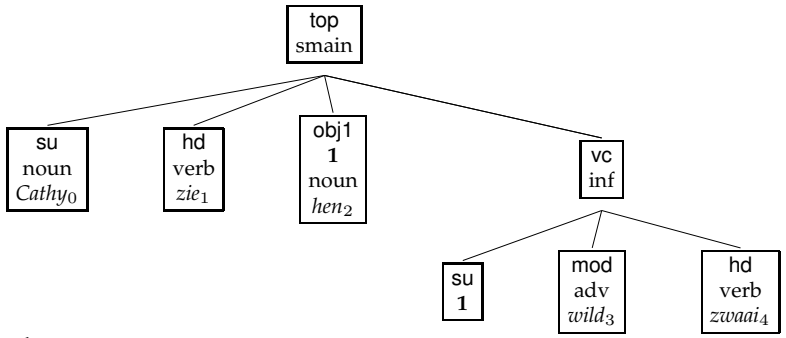
A potential drawback of maximum entropy models is that equation 5 requires access to *all* parses of a given corpus sentence, which is inefficient because a sentence can have an exponential number of parses. Two types of solution for this problem have been proposed. On the one hand, Geman and Johnson (2002); Miyao and Tsujii (2002) present approaches where training data consists of parse forests (or feature forests), rather than sets of parses. Such approaches enforce strong locality requirements on features, whereas in our case features can be arbitrary properties of parses. Geman and Johnson (2002) suggest that it is always possible to localize such arbitrary features in an attribute-value grammar. For some of the features used in Alpino this would dramatically complicate the grammar, and have severe impacts on parsing efficiency. Another type of solution, on which our work is based, is presented in Osborne (2000). Osborne shows that it suffices to provide training with a representative sample of $Y(w)$.

A remaining issue is how the model, once it has been learned from the training data, can be applied efficiently. In the approaches of Geman and Johnson (2002); Miyao and Tsujii (2002) features are localized, and therefore an efficient dynamic programming algorithm can be used to extract the best parse from a parse forest. Below, we define a beam-search generalization of such an algorithm, and we show that the algorithm can be used efficiently to recover the best parse even in the presence of non-local features.

In section 2 we introduce the Alpino grammar, and describe how dependency relations are used for evaluation. Section 3 reviews the approach of Osborne, and describes how it has been adapted for the Alpino grammar and corpora. Section 4 describes which features are defined for disambiguation. Section 5 presents the beam search algorithm to recover the best parse from a parse forest. In section 6 a detailed evaluation of the beam search algorithm and the full parsing system is presented.

2. Alpino Grammar

The Alpino grammar is a wide-coverage computational HPSG for Dutch. The grammar takes a 'constructional' approach, with rich lexical representations and a large number of detailed, construction specific rules (about 500). Both the lexicon and the rule component are organized in a multiple-inheritance hierarchy. By relating rules to each other and to more general structures and principles via multiple inheritance, a rule compo-



⟨zie su Cathy⟩
⟨zie obj1 hen ⟩
⟨zie vc zwaai⟩
⟨zwaai mod wild ⟩
⟨zwaai su hen ⟩

Figure 1
Example of CGN dependency structure for the sentence *Cathy zag hen wild zwaaien* (*Cathy saw them wave wildly*), with the associated set of dependencies used for evaluation.

nent can be defined which contains a potentially large number of specific rules, while at the same time the relevant generalizations about these rules are still expressed only once. Beyond considerations of linguistic theory, an important argument in favor of such an implementation is the fact that parsing using a grammar with specific rules appears to be more efficient that parsing on the basis of general rule schemata.

There is a large lexicon (about 100,000 entries, with a number of additional lexical rules to recognize dates, temporal expressions and similar named entities) derived in part from existing lexical resources (Bouma, 2001), stored as a perfect hash finite automaton.[1]

For words which are not in the lexicon, the system applies a number of unknown word heuristics, which 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 attempts to guess a category based on the word's morphological form. If this still does not provide an analysis, then it is assumed that the word is a noun. Finally, lexical ambiguity is reduced by applying a HMM-filter described in Prins and van Noord (2004).

Dependency Structures and Evaluation. The grammar has been augmented to represent dependency structure, based on the guidelines of CGN (Corpus of Spoken Dutch) (Oostdijk, 2000). An example is given in figure 1. The example illustrates the use of coindexing to represent control relations.

The output of the parser is evaluated by comparing the generated dependency structure for a corpus sentence to the dependency structure in a treebank annotated with a compatible CGN dependency structure. For this comparison, we represent the dependency structure as a set of dependency relations. Comparing these sets, we count the number of relations that are identical in the generated parse and the stored structure. This approach is very similar in spirit to the evaluation methodology advocated in Briscoe et al. (2002), although there are differences with respect to the actual dependencies (which we inherit from the CGN guidelines).

Briscoe et al. compute precision and recall on the basis of sets of dependencies, and f-score can be used to combine both metrics in a single score. Because f-score underestimates the importance of missing dependencies, we prefer to express similarity between dependency structures by *concept accuracy* (generalizing the *word accuracy* measure used

[1] Using Jan Daciuk's FSA tools: `http://www.eti.pg.gda.pl/~jandac/fsa.html`

in speech recognition (Boros et al., 1996)) :

$$\text{CA}^i = 1 - \frac{D_f^i}{\max(D_g^i, D_p^i)}$$

D_p^i is the number of relations produced by the parser for sentence i, D_g is the number of relations in the treebank parse, and D_f is the number of incorrect and missing relations produced by the parser.

To compute the accuracy of the parser on a corpus, we can compute mean CA^i. Given that shorter sentences are typically much easier, a more informative measure is the *total* CA score:

$$\text{CA} = 1 - \frac{\sum_i D_f^i}{\max(\sum_i D_g^i, \sum_i D_p^i)}$$

To emphasize the performance of the stochastic parse selection component, we also define *error reduction*:

$$\text{CA}_\kappa = \frac{\text{CA} - \textit{baseline}\ \text{CA}}{\textit{best}\ \text{CA} - \textit{baseline}\ \text{CA}}$$

The lower bound *baseline* CA is the CA obtained by a model which selects a random parse from the set of parses. The upper bound *best* CA is the CA of an ideal model that always picks the best possible parse. For most of the experiments below, we present total CA, error reduction, and the more traditional f-score.

3. Training the model

The Alpino treebank[2] contains dependency structures of all 7,100 sentences (about 145,000 words) of the newspaper (cdbl) part of the Eindhoven corpus (Uit den Boogaard 1975).

While the treebank contains correct dependency structures for the sentences in the corpus, these structures deliberately abstract away from syntactic details. If we want our disambiguation model to be sensitive to arbitrary aspects of a parse, then the training data should contain the full parse of each sentence as produced by the grammar. To construct these full parses, we use the grammar to parse a given sentence of the training corpus, and then select the parse(s) with the correct dependency structure.

This solution faces two problems. First, the parser might not in all cases be able to produce a parse with the correct dependency structure. Figure 2 illustrates this for the Alpino grammar and treebank. For longer sentences, a considerable proportion cannot be parsed fully correctly. The second problem is that, for longer sentences, it might take too long to find the correct parse even if it exists. This is illustrated in figure 3. For sentences with more than 20 tokens, it becomes unfeasible to enumerate all parses.

Both problems are addressed in Osborne (2000). Osborne suggests mapping the accuracy of a given parse to the probability $\tilde{p}(x|w)$ of that parse in the training data. Thus, rather than adding a parse if its corresponding dependency structure is correct, and ignoring a parse otherwise, we add a parse to the training data with an associated probability that is determined by the quality of that parse, where the quality of a parse is given by the concept accuracy of its dependency structure. Thus, if a parse has a CA of 85%, we add the parse to the training data marked with a weight of 0.85. Next, each sentence is assigned a weight proportional to the sum of the weights of its parses. This gives sentences with higher quality parses more weight than those for which the parser was unable to find a parse. Finally, these parse and sentence weights are renormalized so that $\tilde{p}(w)$ and $\tilde{p}(x|w)$ are proper distributions, and the parameters of the model are then selected to maximize the pseudo-likelihood (5).

[2] http://www.let.rug.nl/~vannoord/trees/

Figure 2
Proportion of sentences for which the parser finds (among all its parses) a fully correct dependency structure, per sentence length.

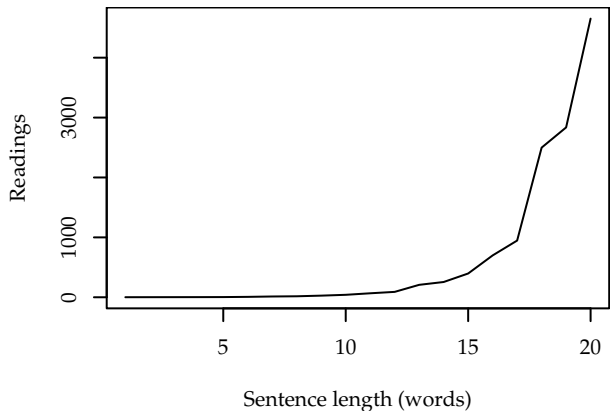


Figure 3
Mean number of parses per sentence length.

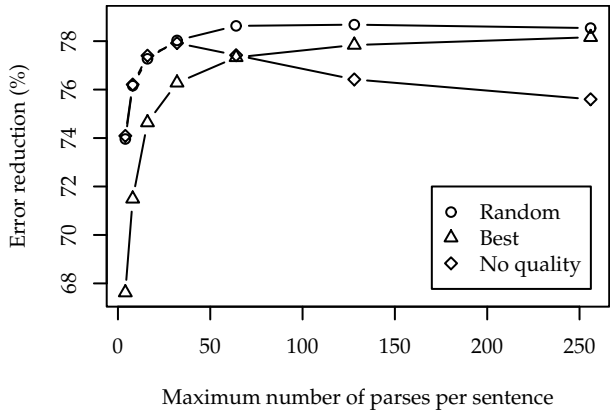


Figure 4
Error reduction versus the maximum number of parses per sentence used for training. Results obtained using ten-fold cross-validation on Alpino treebank.

Secondly, Osborne suggests that we need not have access to *all* parses for a given sentence, but that an "informative sample" is sufficient. The feature weights chosen by maximizing (5) depend only on the expected values of the features in the training data. So, any sub-sample of the parses in the training data which yields unbiased estimates of the feature expectations should result in as accurate a model as the complete set of parses. In initial experiments, we simply used the first n parses for a given sentence (we typically used $n = 250$, to fit within the constraints of a 2Gb core memory system). Since the order of parses is not randomized, somewhat better results can be obtained if we collect $m \gg n$ sentences (say $m = 1000$), from which we then take a random sample of size n. Due to memory problems, three sentences had to be removed from this corpus. In total, 3,270,554 parses were found for 7,136 sentences. For 2,614 sentences, the system found the maximum number of 1,000 parses. The lower bound baseline CA for this set is 58.52%, and the upper bound is 88.29%. Note that the upper bound depends strongly on the maximum number of parses.

In figure 4 the relative performance of models is shown for different values of n. The models all use the same features, feature frequency cutoff, and penalty; details are described below. The result of Osborne is confirmed that concentrating on the best parses (results labeled 'best') only hurts performance. The results labeled 'no quality' are obtained if all best parses are assigned a score of 1, and a random selection of other parses is assigned a score of 0.

4. Feature Selection

Feature Templates. In this section, we describe the features that the disambiguation model uses to distinguish between parses. These features are organized by means of feature templates. The templates are listed in table 1.

Template **r1** signals the application of a particular grammar rule, given by the rule identifier **Rule**. The **r2** template indicates that the **Int**'th daughter of a node constructed by rule **RuleM** is constructed by rule **RuleD**. In template **mf**, **Cat1** and **Cat2** are atomic identifiers for major categories. For noun phrases, the identifiers represent case marking and whether the NP is headed by a proper noun, pronoun, or common noun. An **mf** feature indicates that **Cat1** precedes **Cat2** in the *mittelfeld*, i.e., left of the head in a verb-phrase. This feature is useful, for instance, to learn that, for common noun phrases, the

Table 1
Feature templates and the number of features instantiating this template, for the final model.

r1(Rule)	324
r2(RuleM,Int,RuleD)	2679
mf(Cat1,Cat2)	727
f1(Pos)	451
f2(Word,Pos)	2553
dep23(ArgPos,Rel,Pos)	478
dep34(ArgWord,ArgPos,Rel,Pos)	6241
dep35(ArgWord,ArgPos,Rel,Word,Pos)	6880
h1(Ident)	44
misc boolean features	12

indirect object tends to precede the direct object.

The template **f1(Pos)** represents a word in the derivation with POS-tag **Pos**. Template **f2** represents the fact that **Word** was assigned POS-tag **Pos**. The **dep23, dep34** and **dep35** templates refer to the dependency structure, where **Pos, ArgPos** are simplified POS-tags, **Rel** is a dependency relation (subject, object, determiner, modifier, ...), and **Word, ArgWord** are base forms of words. There are about 20 different simplified POS-tags such as noun, verb, prep, etc. **dep23** indicates that a word with POS-tag **Pos** has a dependent of type **Rel**, headed by a word with POS-tag **ArgPos**. Such a feature makes it possible to learn that, typically, verbs have nouns as objects. The **dep34** template explicitly includes the base form of the head word of the argument. Such features make it possible to learn that, typically, prepositional phrases headed by 'of' are attached to nouns. In the **dep35** template the head word itself is part of the feature too.

The **h1** template indicates that the analysis contains a word with a POS-tag proposed by a particular unknown word heuristic. In addition, there are a number of boolean features. One such feature indicates whether or not, in a coordinated structure, the conjuncts are parallel. The heuristic used to determine parallelism simply is: conjunction is parallel if each conjunct is constructed by the same rule or if each conjunct is a lexical entry. Another feature indicates whether a temporal noun is used adverbially or nominally. Three other boolean features are used to indicate whether in an extraction construction (wh-question, topicalization, relative clause), the subject is extracted or not, and whether such an extraction is local or not. Another boolean feature indicates whether an extraposed relative clause modifies the closest NP to the left, or not.

Frequency cutoff. The templates generate a potentially huge number of features, which we reduce by deleting features whose frequency falls below a cutoff point c. A feature f is *relevant* for a sentence w in the training data, if there are two parses $y_1, y_2 \in Y(w)$ such that $f(y_1) \neq f(y_2)$ (the frequency of that feature must be different for at least one pair of parses). We take a feature into account if it is relevant for more than c sentences. The effects of various cutoffs are given in Table 2. As can be observed, using a small subset of all available features does not hurt performance very much. In the remaining experiments, we have used a frequency cutoff of 2.

Penalty for feature weights. Ratnaparkhi (1998) suggests that frequency cutoff might improve models by reducing over-fitting. However, a more effective countermeasure against over-fitting is the use of a penalized likelihood function for parameter estima-

Table 2
The effect of removing infrequent features.
The first column indicates the cutoff threshold. These models use a Gaussian penalty with $\sigma^2 = 1000$. Results obtained using ten-fold cross-validation on Alpino treebank.

cutoff	# features	F-score	CA%	CA_κ%
-	285,497	78.80	77.69	78.75
1	35,850	78.70	77.59	78.35
2	20,391	78.65	77.53	78.14
3	15,546	78.60	77.49	77.97
5	9,619	78.58	77.47	77.90
10	5,748	78.53	77.40	77.65
20	3,658	78.33	77.21	76.92
50	2,120	78.05	76.94	75.88

Table 3
The effect of Gaussian penalty.
These models use frequency cutoff of 2. Results obtained using ten-fold cross-validation on Alpino treebank.

σ^2	F-score	CA%	CA_κ%	iterations
1	73.41	72.07	57.46	5
10	76.48	75.26	69.52	12
100	78.01	76.89	75.65	30
1000	78.65	77.53	78.14	80
10000	78.42	77.32	77.34	160
100000	77.81	76.72	75.05	250
none	77.52	76.43	73.97	275

tion (Chen and Rosenfeld, 1999; Johnson et al., 1999). Rather than maximizing the likelihood (5) to estimate the parameters θ_i, we instead maximize a penalized likelihood:

$$L'(\theta) = L(\theta) - \frac{1}{2\sigma^2}\sum_i \theta_i^2$$

This Gaussian regularization term penalizes extreme parameter values, which reduces the 'effective degrees of freedom' of the model and in turn tends to reduce over-fitting (Hastie, Tibshirani, and Friedman, 2001). The variance σ^2 is a smoothing parameter which sets the relative influence of the likelihood and the regularization term: larger values of σ^2 results in less smoothing of the parameters θ_i. The results of using a Gaussian penalty are given in Table 3. The penalty can improve accuracy and in addition often makes training converge faster. In further experiments, a value of $\sigma^2 = 1000$ is assumed.

In summary, a Gaussian penalty is used for more accurate models, and a feature frequency cutoff is used for more compact models.

Comparison. The disambiguation model solves more than 78% of the disambiguation problem (by ten-fold cross-validation on the Alpino treebank). It is hard to directly

r1: s → np vp	r2: vp → vp pp	r3: np → n
r4: np → det n	r5: np → np pp	r6: pp → p np
r7: vp → v np		
l1: a → "I"	l2: v → "see"	l3: det → "a"
l4: n → "man"	l5: p→ "at"	l6: n→ "home"

Figure 5
Sample grammar

compare these results with others, due to differences in language, grammar, training data and test data. Riezler et al. (2002) report error reductions between 32% and 36%. Osborne (2000) evaluates the model rather crudely by comparing how often the model picks out the best possible parse (ignoring the quality of that parse itself, and ignoring the number of parses).

5. Recovery of best parse

The construction of a dependency structure on the basis of some input proceeds in a number of steps. After lexical analysis, a parse forest is constructed using a left-corner parser. If no single analysis of the input is possible, the parser constructs a parse forest for a sequence of analyses (van Noord, 1997).

From the parse forest, the best parse must be selected, based on the disambiguation model described in the previous section. In order to select the best parse from a parse forest, we assume a parse evaluation function which assigns a score to each parse. The parse evaluation function simply applies the disambiguation model described in previous sections, by counting the frequency of each of the features. The frequency of each feature is then multiplied with the corresponding weight, and finally these products are then all summed to arrive at a number indicating the (relative) quality of the parse.

A naive algorithm constructs all possible parses, assigns each one a score, and then selects the best one. In the approach we take here, a parse is selected from the parse forest by a best-first search. This requires the parse evaluation function to be extended to partial parses.

The left-corner parser constructs a parse forest, using the technique explained in detail in section 4 of van Noord (1997). In this approach, the parse forest is a tree substitution grammar, which derives exactly all derivation trees of the input sentence. Each tree in the tree substitution grammar is a left-corner spine. An example should clarify this.

Example. Consider the simple grammar and lexicon presented in figure 5, where terminals are written within double quotes, and each rule is prefixed by a rule identifier. We use a context-free grammar for ease of exposition, but since we are actually constructing derivation trees, rather than parse trees, the technique immediately generalizes for attribute-value grammars.

The sentence *I see a man at home* has the two parse trees and corresponding derivation trees given in figure 6. The left-corner parser constructs the parse forest given in figure 7. Such a parse forest consists of a set of pairs, where each pair is an index and a set of partial derivation trees (left-corner spines). Each left-corner spine is a tree, where all non-terminal nodes as well as the left-most terminal node are rule names, and where all other terminal nodes are indexes. Full derivation trees can be constructed by composing the partial derivation trees together, with the condition that a node labeled by an

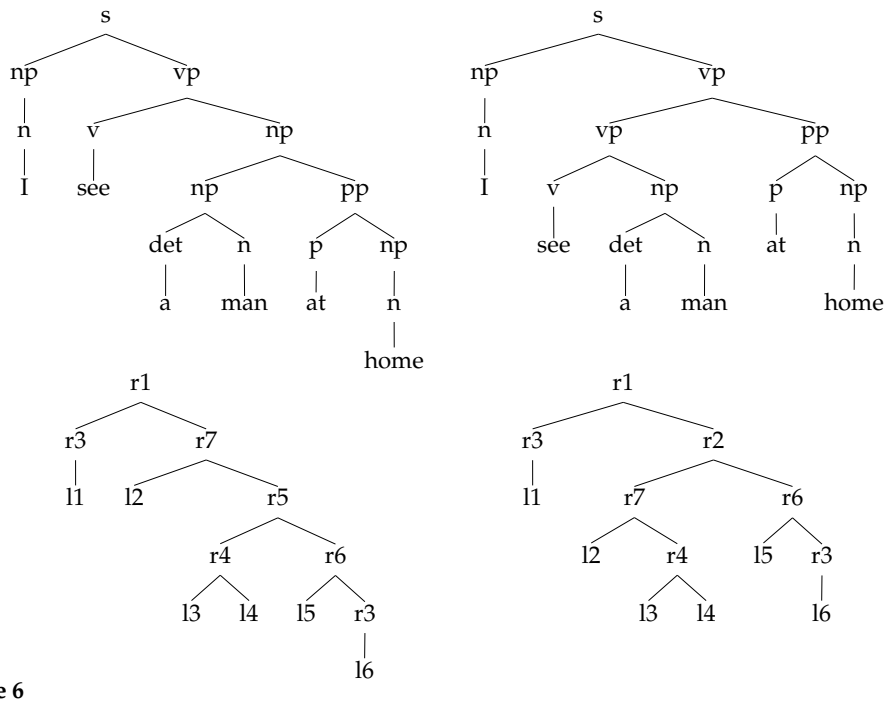


Figure 6
Two parse trees and corresponding derivation trees for *I see a man at home*

index should be substituted by a partial derivation tree associated with that index. The index associated with the start symbol is given (in the example, the start index is nt0).

Parse recovery. An algorithm which recovers a derivation tree for a given index is given in figure 8. It is closely related to the algorithm presented in Geman and Johnson (2002) for selecting best parses from LFG packed representations.

The algorithm first topologically sorts the indexes, where an index i precedes j if a tree associated with i is required in a possible derivation tree for j (line 1). The algorithm iterates over the indexes in this ordering, constructing larger derivation trees on the basis of derivation trees created earlier. To create a derivation tree for a specific index i, the algorithm iterates over all trees associated with i (line 2). In such a tree, there are

nt0	nt1	nt2	nt3	nt4	nt5	nt6
r1 r3 nt1 l1	r2 r7 r7 nt3 l2 nt4 l2 nt2	r4 l3 nt5	r6 l5 nt6	r5 r4 nt3 l3 nt5	l4	r3 l6

Figure 7
The parse forest of *I see a man at home.*

The parse forest consists of a number of indexes associated with sets of partial derivation trees. The derivation trees are left-corner spines where all non-terminal nodes and the left-most terminal node are rule-names. Each of the other terminal nodes is labeled with an index, indicating that one of the partial derivation trees associated with that index can be plugged in here.

```
RECOVER(start, indexes)
1  for each i in TOP-SORT(indexes)
2       do for each sub ∈ i.trees
3            do I ← indexes in sub
4               for each j ∈ I
5                    do SUBS(j, j.best)
6               if sub is better than i.best
7                 then i.best ← sub
8  return start.best
```

Figure 8
Algorithm RECOVER

```
RECOVER-WITH-BEAM(start, indexes)
1  for each i in TOP-SORT(indexes)
2       do for each sub ∈ i.trees
3            do I_1 ... I_k ← indexes in sub
4               for each (t_1 ... t_k) ∈
                I_1. best × ... × I_k. best
5                    do for i ← 1 to k
6                            do SUBS(I_i, t_i)
7                          ADD(b,best,sub.best)
8  return best element of start.best
```

Figure 9
Algorithm RECOVER-WITH-BEAM

a number of nodes labeled with an index. For each of these, the corresponding best derivation tree (discovered in a previous iteration) is substituted at the corresponding node (line 5). Then, a score is computed for the resulting tree. This involves mapping the derivation tree to a full parse tree, counting the occurrences of all features, multiplying these counts with the corresponding feature weights, and summing the results. If the new tree has a higher score than the best tree associated with i so far, then the tree is stored. Finally, the algorithm returns the best parse associated with the start node.

In order to be able to *guarantee* that this search procedure indeed finds the best parse, a monotonicity requirement should apply to the parse evaluation function. However, instead of relying on such a requirement (some non-local features discussed in the previous section would indeed violate this 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 risk of missing the best parse (but the result will typically still be a relatively 'good' parse); if the beam is increased, then the amount of computation increases as well. The definition is presented in figure 9.

The algorithm works in a similar fashion as before, but instead of keeping track of the single best parse for each index, we maintain at most b best parses for each index. Because an index is associated with a set of trees, the algorithm iterates over all combinations of sub-trees (line 4), and then substitutes each one of those sub-trees (line 5). The procedure ADD will add a given tree t to a set of trees T if either there are less than

b trees in T, or t is better than at least one of the trees in T. In the latter case, the worst scoring tree is removed from T.

Comparison. Rather than computing the parse-forest first, and then compute the best parse from this forest, one might attempt to apply the disambiguation component immediately during parsing. First note, though, that a naive application of this idea is problematic for attribute-value grammars. Even if a particular component of a parse has a very low score, it might be that attribute value constraints, enforced by the context of that component parse, will rule out all competing, better scoring parses. Therefore, we should only eliminate a candidate component parse, if there are better parses *with the same span and the same attribute-value structure.* But if the parser builds a parse-forest, and uses packing, then it will in such cases continue parsing with a single parse anyway.

In Nederhof (2003) two types of algorithm are discussed to find the best parse. The first type of algorithm is closely related to Dijkstra's algorithm to find the shortest path in a directed graph; its application to probabilistic context-free grammars is known as Knuth's algorithm. It consists of an agenda-driven, chart parsing algorithm in which the agenda is ordered in such a way that promising items are processed before others. This type of algorithm is applicable provided the scores are *superior.* This roughly implies that:

- the score of a parse is equal or lower than the score of each of its components
- if a component c_1 has a higher score than c_2, then it should be the case that all parses which contain c_1 have a higher score than all corresponding parses which have c_2 instead of c_1.

Both conditions are not applicable. The first condition is violated because the maxent feature weights are logits rather than log probabilities, and so can be either positive or negative. The second condition is violated, in general, since we allow various non-local features.

The second type of algorithm discussed by Nederhof is closely related to the Viterbi algorithm, and to the DAG-SHORTEST-PATHS algorithm as described in Cormen, Leiserson, and Rivest (1990), as well as to the algorithm for finding the best parse in a parse forest presented in Geman and Johnson (2002). This type of algorithm works provided the second condition above applies. Our algorithm can be described as a generalization of the second type. The generalization consists in allowing the b best candidates for each component, to compensate for the effect of global features which violate condition 2 above.

6. Experimental Results

Beam Search. In table 4 the effect of various values for b is presented for a number of different treebanks. In the first columns, we list the results on a random sample of sentences from the treebank of up to fifteen words. In the next few columns, we list the result on a random sample of sentences from the treebank of up to thirty words. In the final columns, a random sample of the treebank is used without a restriction on sentence length. Per column, we list the F-score, concept accuracy, CPU requirements, and the number of sentences for which the parser could not find an analysis due to memory limitations (in such cases the accuracy obviously is dropped too, since no correct result is constructed). As can be seen from the table, increasing the beam size slightly improves results, but for larger values memory problems cause a severe drop of accuracy. Also, the beam can remain rather small. This is probably due to the fact that most of our

Table 4
Effect of beam-size on accuracy and efficiency of parse selection.
Sentences from random 10% of Alpino Treebank. The left part of the table displays results for sentences up to 15 words; the central part for sentences up to 30 words; and the right part for all sentences. We normalize the parse selection times with respect to the variant with $b = \infty$ (CPU=1), ignoring sentences for which the algorithm ran out of memory.

beam	≤ 15			≤ 30			all		
	CA%	CPU	out	CA%	CPU	out	CA%	CPU	out
1	90.33	0.66	0	86.69	0.23	0	84.82	0.14	0
2	90.62	0.72	0	87.08	0.28	0	85.18	0.18	0
4	90.71	0.79	0	87.19	0.37	0	85.36	0.28	0
8	90.85	0.87	0	87.32	0.50	0	85.49	0.39	0
16	90.85	0.98	0	87.37	0.70	0	85.60	0.56	0
32	90.85	1.17	0	87.11	1.04	1	84.87	0.90	4
∞	90.85	1	0	80.16	1	32	69.59	1	74

features are rather local in nature, as well as to the fact that the basic units of the parse forest are relatively large in size. Currently, Alpino uses $b = 4$ by default.

Full System. In table 5, the accuracy of the full system on the Alpino treebank is given in the first row, using ten-fold cross-validation. The accuracy is much higher than in tables 2 and 3, because we are not limited anymore to the maximum of 1000 parses per sentence. Error reduction on the Alpino treebank is lower for the full system than reported in those earlier tables, because in the full system the POS-tagger of Prins and van Noord (2004) is employed to reduce lexical ambiguities, which solves many of the 'easy' disambiguation decisions.

The Alpino treebank is, strictly speaking, a development set on which we optimized both the grammar and lexicon somewhat, as well as the various tuning parameters for training (frequency cut-off, Gaussian penalty, beam size). Therefore, we provide results on two other test sets as well (using the model trained on the Alpino treebank). To this end, we annotated the first 500 sentences of the Trouw 2001 newspaper, as found in the TwNC newspaper corpus.[3] Another test set consists of all the Dutch questions from the CLEF Question Answering competition. As can be seen in the table, the results on these test sets are even better. A potential explanation is the fact that these sets are easier because of shorter mean sentence length.

Note that in these experiments, all sentences receive at least one analysis (except for one sentence of the Alpino set, for which the parser ran out of memory), because if no single analysis of the input is possible, a sequence of partial analyses will be constructed.

Wide coverage parsing is understood here in the sense that not only the parser should produce some parse for arbitrary, free, input text, but moreover should produce the correct or mostly correct parse. We suggest our results warrant the conclusion that the SAVG framework is applicable for wide-coverage parsing in this sense. Recently, Alpino has indeed been used for wide-coverage parsing in the context of question answering, word-sense disambiguation, pronoun resolution, preparation of training material for enhanced POS-tagging and corpus linguistics (Villada Moirón, 2004; Prins, 2004; Bouma, 2003; Bouma, 2004).

[3] `http://wwwhome.cs.utwente.nl/~druid/TwNC/TwNC-main.html`

Table 5
Accuracy on development set and test sets for full system.
The table lists the number of sentences, mean sentence length, F-score, CA and error reduction.

corpus	sents	length	F-sc	CA%	CA_{κ}%
Alpino	7136	19.7	85.78	84.66	72.30
CLEF	446	11.3	90.99	88.68	74.04
Trouw	500	17.0	88.94	87.86	80.51

Acknowledgments

This research was supported by the PIONIER project *Algorithms for Linguistic Processing* funded by *Nederlandse Organisatie voor Wetenschappelijk Onderzoek* (NWO). Helpful remarks from Miles Osborne and Gosse Bouma gratefully acknowledged.

References

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

Balay, Satish, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. 2002. PETSc users manual. Technical Report ANL-95/11–Revision 2.1.2, Argonne National Laboratory.

Benson, Steven J., Lois Curfman McInnes, Jorge J. Moré, and Jason Sarich. 2002. TAO users manual. Technical Report ANL/MCS-TM-242–Revision 1.4, Argonne National Laboratory.

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

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

Boros, M., W. Eckert, F. Gallwitz, G. Görz, G. Hanrieder, and H. Niemann. 1996. Towards understanding spontaneous speech: Word accuracy vs. concept accuracy. In *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP 96)*, Philadelphia.

Bouma, Gerlof. 2003. Doing Dutch pronouns automatically in optimality theory. In *Proceedings of the EACL workshop on the Computational Treatment of Anaphora*, Budapest.

Bouma, Gosse. 2001. Extracting dependency frames from existing lexical resources. In *Proceedings of the NAACL Workshop on WordNet and Other Lexical Resources: Applications, Extensions and Customizations*, Somerset, NJ. Association for Computational Linguistics.

Bouma, Gosse. 2004. Question answering for Dutch using dependency relations. Technical report, University of Groningen.

Briscoe, Ted, John Carroll, Jonathan Graham, and Ann Copestake. 2002. Relational evaluation schemes. In *Proceedings of the Beyond PARSEVAL Workshop at the 3rd International Conference on Language Resources and Evaluation*, pages 4–8, Las Palmas, Gran Canaria.

Chen, Stanley F. and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Computer Science Department, Carnegie Mellon University.

Cormen, Leiserson, and Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge Mass.

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

Geman, Stuart and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the ACL*.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning Theory*. Springer Verlag, New York.

Jaynes, E.T. 1957. Information theory and statistical mechanics. *Physical Review*, 106,108:620–630.

Johnson, Mark, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the ACL*.

Nederhof, Mark-Jan. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

van Noord, Gertjan. 1997. An efficient implementation of the head corner parser. *Computational Linguistics*, 23(3):425–456.

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

Osborne, Miles. 2000. Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the Eighteenth International Conference on Computational Linguistics (COLING 2000)*.

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

Prins, Robbert. 2004. Beyond N in Ngram tagging. Technical report, University of Groningen.

Prins, Robbert and Gertjan van Noord. 2004. Reinforcing parser preferences through tagging. *Traitement Automatique des Langues*. in press.

Ratnaparkhi, Adwait. 1998. *Maximum entropy models for natural language ambiguity resolution*. Ph.D. thesis, University of Pennsylvania.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the ACL*.

Villada Moirón, Begoña. 2004. Discarding noise in an automatically acquired lexicon of support verb constructions. In *LREC 2004*.