Automation of Treebank Annotation

Thorsten Brants and Wojciech Skut
Universität des Saarlandes
Computational Linguistics
D-66041 Saarbrücken, Germany
{brants,skut}@coli.uni-sb.de

Abstract

This paper describes applications of stochastic and symbolic NLP methods to treebank annotation. In particular we focus on (1) the automation of treebank annotation, (2) the comparison of conflicting annotations for the same sentence and (3) the automatic detection of inconsistencies. These techniques are currently employed for building a German treebank.

1 Introduction

The emergence of new statistical NLP methods increases the demand for corpora annotated with syntactic structures. The construction of such a corpus (a *treebank*) is a time-consuming task that can hardly be carried out unless some annotation work is automated. Purely automatic annotation, however, is not reliable enough to be employed without some form of human supervision and hand-correction. This interactive annotation strategy requires tools for error detection and consistency checking.

The present paper reviews our experience with the development of automatic annotation tools which are currently used for building a corpus of German newspaper text.

The next section gives an overview of the annotation format. Section 3 describes three applications of statistical NLP methods to treebank annotation. Finally, section 4 discusses mechanisms for comparing structures assigned by different annotators.

2 Annotating Argument Structure

2.1 Annotation Scheme

Unlike most treebanks of English, our corpus is annotated with *predicate-argument structures* and not phrase-structure trees. The reason is the free word order in German, a feature seriously affecting the transparency of traditional phrase structures. Thus local and non-local dependencies are represented in the same way, at the cost of allowing crossing tree branches, as shown in figure 1[1].

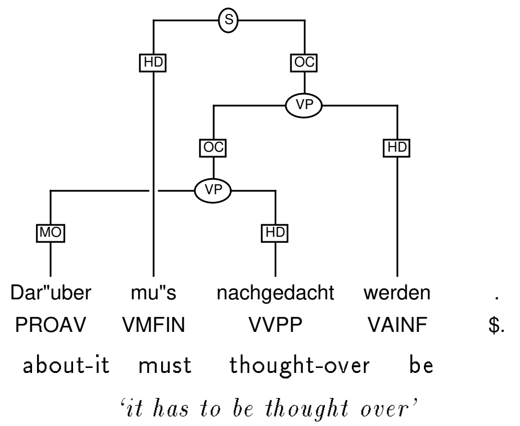


Figure 1: Sample structure from the Treebank

Such a direct representation of the predicate-argument relation makes annotation easier than it would be if additional trace-filler co-references were used for encoding discontinuous constituents. Furthermore, our scheme facilitates automatic extraction of valence frames and the construction of semantic representations.

On the other hand, the predicate-argument structures used for annotating our corpus can still be converted automatically into phrase-structure trees if necessary, cf. (Skut et al., 1997a). For more details on the annotation scheme v. (Skut et al., 1997b).

2.2 The Annotation Mode

In order to make annotation more reliable, each sentence is annotated independently by two annotators. Afterwards, the results are compared, and both annotators have to agree on a unique structure. In

[1]The nodes and edges are labeled with category and function symbols, respectively (see appendix A).

Thorsten Brants and Wojciech Skut (1998) **Automation of Treebank Annotation.** In D.M.W. Powers (ed.) *NeMLaP3/CoNLL98: New Methods in Language Processing and Computational Natural Language Learning,* ACL, pp 49-57.

case of persistent disagreement or uncertainty, the grammarian supervising the annotation work is consulted.

It has turned out that comparing annotations involves significantly more effort than annotation proper. As we do not want to abandon the annotate-and-compare strategy, additional effort has been put into the development of tools supporting the comparison of annotated structures (see section 4).

3 Automation

The efficiency of annotation can be significantly increased by using automatic annotation tools. Nevertheless, some form of human supervision and hand-correction is necessary to ensure sufficient reliability. As pointed out by (Marcus, Santorini, and Marcinkiewicz, 1994), such a *semi-automatic* annotation strategy turns out to be superior to purely manual annotation in terms of accuracy and efficiency. Thus in most treebank projects, the task of the annotators consists in correcting the output of a parser, cf. (Marcus, Santorini, and Marcinkiewicz, 1994), (Black et al., 1996).

As for our project, the unavailability of broad-coverage argument-structure and dependency parsers made us adopt a bootstrapping strategy. Having started with completely manual annotation, we are gradually increasing the degree of automation. The corpus annotated so far serves as training material for annotation tools based on statistical NLP methods, and the degree of automation increases with the amount of annotated sentences.

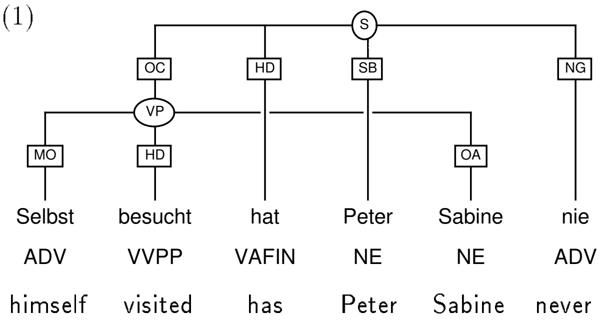
Automatic processing and manual input are combined interactively: the annotator specifies some information, another piece of information is added automatically, the annotator adds new information or corrects parts of the structure, new parts are added automatically, and so on. The size and type of such annotation increments depends on the size of the training corpus. Currently, manual annotation consists in specifying the hierarchical structure, whereas category and function labels as well as simple substructures are assigned automatically. These automation steps are described in the following sections.

3.1 Tagging Grammatical Functions

Assigning grammatical functions to a given hierarchical structure is based on a generalization of standard part-of-speech tagging techniques.

In contrast to a standard probabilistic POS tagger (e.g. (Cutting et al., 1992; Feldweg, 1995)), the tagger for grammatical functions works with lexical

(1)



'Peter never visited Sabine himself'

Figure 2: Example sentence

and contextual probability measures $P_Q(\cdot)$ depending on the category of a mother node (Q). This additional parameter is necessary since the sequence of grammatical functions depends heavily on the type of phrase in which it occurs. Thus each category (S, VP, NP, PP etc.) defines a separate Markov model.

Under this perspective, categories of daughter nodes correspond to the outputs of a Markov model (i.e., like words in POS tagging). Grammatical functions can be viewed as states of the model, analogously to tags in a standard part-of-speech tagger.

Given a sequence of word and phrase categories $T = T_1 \ldots T_k$ and a parent category Q, we calculate the sequence of grammatical functions $G = G_1 \ldots G_k$ that link T and Q as

$$\operatorname*{argmax}_G P_Q(G|T) \tag{1}$$

$$= \operatorname*{argmax}_G \frac{P_Q(G) \cdot P_Q(T|G)}{P_Q(T)}$$
$$= \operatorname*{argmax}_G P_Q(G) \cdot P_Q(T|G)$$

Assuming the Markov property we have

$$P_Q(T|G) = \prod_{i=1}^{k} P_Q(T_i|G_i) \tag{2}$$

and (using a trigram model)

$$P_Q(G) = \prod_{i=1}^{k} P_Q(G_i|G_{i-2}, G_{i-1}) \tag{3}$$

The contexts are smoothed by linear interpolation of unigrams, bigrams, and trigrams. Their weights are calculated by deleted interpolation (Brown et al., 1992).

The structure of a sample sentence is shown in figure 2. Here, the probability of the S node having this particular sequence of children is calculated as

$$\begin{aligned} P_S(G,T) \;=\; & P_S(\mathsf{OC}|\$,\$) \cdot P_S(\mathsf{VP}|\mathsf{OC}) \\ & \cdot P_S(\mathsf{HD}|\$,\mathsf{OC}) \cdot P_S(\mathsf{VAFIN}|\mathsf{HD}) \\ & \cdot P_S(\mathsf{SB}|\mathsf{OC},\mathsf{HD}) \cdot P_S(\mathsf{NE}|\mathsf{SB}) \\ & \cdot P_S(\mathsf{NG}|\mathsf{HD},\mathsf{SB}) \cdot P_S(\mathsf{ADV}|\mathsf{NG}) \end{aligned}$$

($ indicates the start of the sequence).

The predictions of the tagger are correct in approx. 94% of all cases. During the annotation process this is further increased by exploiting a precision/recall trade-off (cf. section 3.5).

3.2 Tagging Phrasal Categories

The second level of automation is the recognition of phrasal categories, which frees the annotator from typing phrase labels. The task is performed by an extension of the grammatical function tagger presented in the previous section.

Recall that each phrasal category defines a different Markov model. Given the categories of the children nodes in a phrase, we can run these models in parallel. The model that assigns the most probable sequence of grammatical functions determines the category label to be assigned to the parent node.

Formally, we calculate the phrase category Q (and at the same time the sequence of grammatical functions $G = G_1 \ldots G_k$) on the basis of the sequence of daughters $T = T_1 \ldots T_k$ with

$$\underset{Q}{\operatorname{argmax}} \; \max_{G} P_Q(G|T).$$

This procedure can also be performed using one large (combined) Markov model that enables a very efficient calculation of the maximum.

The overall accuracy of this approach is 95%.

3.3 Tagging Hierarchical Structure

The next automation step is the recognition of syntactic structures. In general, this task is much more difficult than assigning category and function labels, and requires a significantly larger training corpus than the one currently available. What can be done at the present stage is the recognition of relatively simple structures such as NPs and PPs.

(Church, 1988) used a simple mechanism to mark the boundaries of NPs. He used part-of-speech tagging and added two flags to the part-of-speech tags to mark the beginning and the end of an NP.

Our goal is more ambitious in that we mark not only the phrase boundaries of NPs but also the complete structure of a wider class of phrases, starting with APs, NPs and PPs.

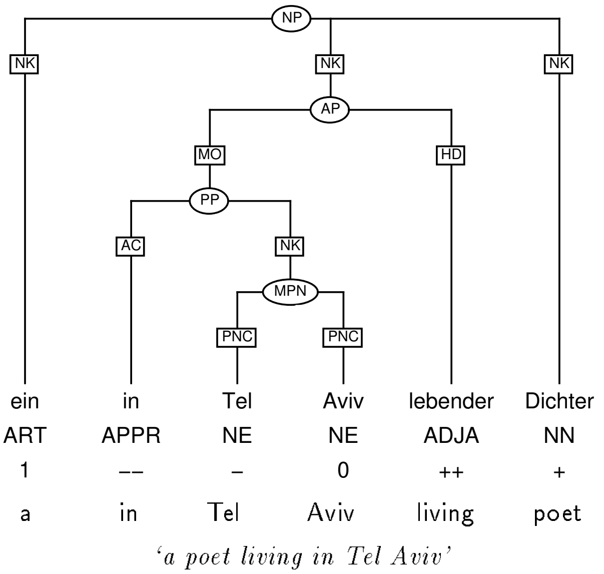


Figure 3: Structural tags

(Ratnaparkhi, 1997) uses an iterative procedure to assign two types of tags (*start X* and *join X*, where X denotes the type of the phrase) combined with a process to build trees.

We go one step further and assign simple structures in one pass. Furthermore, the nodes and branches of these tree chunks have to be assigned category and function labels.

The basic idea is to encode structures of limited depth using a finite number of tags. Given a sequence of words $\langle w_0, w_1, \ldots w_n \rangle$, we consider the structural relation r_i holding between w_i and w_{i-1} for $1 \leq i \leq n$. For the recognition of NPs and PPs, it is sufficient to distinguish the following seven values of r_i which uniquely identify sub-structures of limited depth.

$$r_i = \begin{cases} 0 & \text{if} \quad parent(w_i) = parent(w_{i-1}) \\ + & \text{if} \quad parent(w_i) = parent^2(w_{i-1}) \\ ++ & \text{if} \quad parent(w_i) = parent^3(w_{i-1}) \\ - & \text{if} \quad parent^2(w_i) = parent(w_{i-1}) \\ -- & \text{if} \quad parent^3(w_i) = parent(w_{i-1}) \\ = & \text{if} \quad parent^2(w_i) = parent^2(w_{i-1}) \\ 1 & \quad \text{else} \end{cases}$$

If more than one of the conditions above are met, the first of the corresponding tags in the list is assigned. A structure tagged with these symbols is shown in figure 3.

In addition, we encode the POS tag t_i assigned to w_i. On the basis of these two pieces of information we define *structural tags* as pairs $S_i = \langle r_i, t_i \rangle$. Such tags constitute a finite alphabet of symbols describing the structure and syntactic category of phrases of depth ≤ 3.

The task is to assign the most probable sequence of structural tags ($\langle S_0, S_1, ..., S_n \rangle$) to a sequence of part-of-speech tags $\langle T_0, T_1, ..., T_n \rangle$.

Given a sequence of part-of-speech tags $T = T_1 \ldots T_k$, we calculate the sequence of structural tags $S = S_1 \ldots S_k$ such that

$$\operatorname*{argmax}_S P(S|T) \tag{4}$$

$$= \operatorname*{argmax}_S \frac{P(S) \cdot P(T|S)}{P(T)}$$

$$= \operatorname*{argmax}_S P(S) \cdot P(T|S)$$

The part-of-speech tags are encoded in the structural tag (t), so S uniquely determines T. Therefore, we have $P(T|S) = 1$ if $T_i = t_i$ and 0 otherwise, which simplifies calculations:

$$\operatorname*{argmax}_S P(S) \cdot P(T|S) \tag{5}$$

$$= \operatorname*{argmax}_S \prod_{i=1}^{k} P(S_i|S_{i-2}, S_{i-1}) P(T_i|S_i)$$

As in the previous models, the contexts are smoothed by linear interpolation of unigrams, bigrams, and trigrams. Their weights are calculated by deleted interpolation.

This *chunk tagging* technique can be applied to treebank annotation in two ways. Firstly, we could use it as a preprocessor; the annotator would then complete and correct the output of the chunk tagger.

The second alternative is to combine this chunking with manual input in an interactive way. Then the annotator has to determine the boundaries of the sub-structure that is to be build by the program.

Obviously, the second solution is favorable since the user supplies information about chunk boundaries, while in the preprocessing mode the tagger has to find both the boundaries and the internal structure of the chunks.

The assignment of structural tags is correct in more than 94% of the cases. For detailed results see section 3.6.3.

3.4 Interaction and Alternation

To illustrate the interaction of manual input and the automatic annotation techniques described above, we show the way in which the structure in figure 3 is constructed. The current version of the annotation tool supports automatic assignment of category and phrase labels, so the user has to specify the hierarchical structure step by step[2].

The starting point is the plain string of words together with their part-of-speech tags. The annotator first selects the words *Tel Aviv* and executes the command "group" (this is all done with the mouse). Then the program inserts the category label MPN (multi-lexeme proper noun) and assigns the grammatical function PNC (proper noun component) to both words (cf. sections 3.2 and 3.1).

Having completed the first sub-structure, the annotator selects the newly created MPN and the preposition *in*, and creates a new phrase. The tool automatically inserts the phrase label PP and the grammatical functions AC (adpositional case marker) and NK (noun kernel component). The following two steps are to determine the components of the AP and, finally, those of the NP.

At any time, the annotator has the opportunity to change and correct entries made by the program.

This interactive annotation mode is favorable from the point of view of consistency checking. The first reason is that the annotation increments are rather small, so the annotator corrects not an entire parse tree, but a fairly simple local structure. The automatic assignment of phrase and function labels is generally more reliable than manual input because it is free of typically human errors (see the precision results in (Brants, Skut, and Krenn, 1997)). Thus the annotator can concentrate on the more difficult task, i.e., building complex syntactic structures.

The second reason is that errors corrected at lower levels in the structure facilitate the recognition of structures at higher levels, thus many wrong readings are excluded by confirming or correcting a choice at a lower level.

The partial automation of the annotation process (automatic regocnition of phrase labels and grammatical functions) has reduced the average annotation time from about 10 to 1.5 – 2 minutes per sentence, i.e. 600 – 800 tokens per minute, which is comparable to the figures published by the creators of the Penn Treebank in (Marcus, Santorini, and Marcinkiewicz, 1994).

The test version of the annotation tool using the statistical chunking technique described in section 3.3 permits even larger annotation increments and we expect a further increase in annotation speed. The user just has to select the words constituting an

[2] The chunk tagger has not yet been fully integrated into the annotation tool.

NP or PP. The program assigns a sequence of structural tags to them; these tags are then converted to a tree structure and all labels are inserted.

3.5 Reliability

To make automatic annotation more reliable, the program assigning labels performs an additional reliability check. We do not only calculate the best assignment, but also the second-best alternative and its probability. If the probability of the alternative comes very close to that of the best sequence of labels, we regard the choice as unreliable, and the annotator is asked for confirmation.

Currently, we employ three reliability levels, expressed by quotients of probabilities p_{best}/p_{second}. If this quotient is close to one (i.e., smaller than some threshold θ_1), the decision counts as unreliable, and annotation is left to the annotator. If the quotient is very large (i.e., greater than some threshold $\theta_2 > \theta_1$), the decision is regarded as reliable and the respective annotation is made by the program.

If the quotient falls between θ_1 and θ_2, the decision is tagged as "almost reliable". The annotation is inserted by the program, but has to be confirmed by the annotator.

This method enables the detection of a number of errors that are likely to be missed if the annotator is not asked for confirmation.

The results of using these reliability levels are reported in the experiments section below.

3.6 Experiments

This section reports on the accuracy achieved by the methods described in the previous sections.

At present, our corpus contains approx. 6300 sentences (115,000 tokens) of German newspaper text (Frankfurter Rundschau). Results of tagging grammatical functions and phrase categories have improved slightly compared to those reported for a smaller corpus of approx. 1200 sentences (Brants, Skut, and Krenn, 1997). Accuracy figures for tagging the hierarchical structure are published for the first time.

For each experiment, the corpus was divided into two disjoint parts: 90% training data and 10% test data. This procedure was repeated ten times, and the results were averaged.

The thresholds θ_1 and θ_2 determining the reliability levels were set to $\theta_1 = 5$ and $\theta_2 = 100$.

3.6.1 Grammatical Functions

We employ the technique described in section 3.1 to assign grammatical functions to a structure defined by an annotator. Grammatical functions are represented by edge labels. Additionally, we exploit the recall/accuracy tradeoff as described in section 3.5. The tagset of grammatical functions consists of 45 tags.

Table 1: Levels of reliability and the percentage of cases in which the tagger assigned a correct grammatical function (or would have assigned if a decision had been forced).

grammatical function	cases	correct
reliable	88%	97.0%
marked	8%	85.0%
unreliable	4%	59.5%
overall	100%	94.6%

Tagging results are shown in table 1. Overall accuracy is 94.6%. 88% of all predictions are classified as reliable, which is the most important class for the actual annotation task. Accuracy in this class is 97.0%. It depends on the category of the phrase, e.g. accuracy for reliable cases reaches 99% for NPs and PPs.

3.6.2 Phrasal Categories

Now the task is to assign phrasal categories to a structure specified by the annotator, i.e., only the hierarchical structure is given. We employ the technique of competing Markov models as described in section 3.2 to assign phrase categories to the structure. Additionally, we compute alternatives to assign one of the three reliability levels to each decision as described in section 3.5. The tagset for phrasal categories consists of 25 tags.

Table 2: Levels of reliability and the percentage of cases in which the tagger assigned a correct phrase category (or would have assigned it if a decision had been forced).

phrase category	cases	correct
reliable	76%	99.0%
marked	19%	91.5%
unreliable	5%	56.7%
overall	100%	95.4%

As can be seen from table 2, the results of assigning phrasal categories are even better than those of assigning grammatical functions. Overall accuracy is 95.4%. Tags that are regarded as reliable (76% of all cases) have an accuracy of 99.0%, which results

Table 3: Chunk tagger accuracy with respect to hierarchical structure.

structural tags	cases	correct
reliable	86%	95.8%
marked	11%	93.2%
unreliable	3%	67.0%
overall	100%	94.4%

in very reliable annotations.

3.6.3 Chunk Tagger

The chunk tagger described in section 3.3 assigns tags encoding structural information to a sequence of words and tags. The accuracy figures presented here refer to the correct assignments of these tags (see table 3).

The assignment of structural tags allows us to construct a tree; the labels are afterwards assigned in a bottom-up fashion by the function/category label tagger described in earlier sections.

Overall accuracy is 94.4% and reaches 95.8% in the reliable cases.

A different measure of the chunker's correctness is the percentage of *complete phrases* recognized correctly. In order to determine this percentage, we extracted all chunks of the maximal depth recognizable by the chunker. In a cross evaluation, 87.3% of these chunks were recognized correctly as far as the hierarchical structure is concerned.

4 Comparing Trees

Annotations produced by different annotators are compared automatically and differences are marked. The output of the comparison is given to the annotators. First, each of the annotators goes through the differences on his own and corrects obvious errors. Then remaining differences are resolved in a discussion of the annotators.

Additionally, the program calculates the probabilities of the two different annotations. This is intended to be a first step towards resolving conflicting annotations automatically. Both parts, tree matching and the calculation of probabilities for complete trees are described in the following sections.

4.1 Tree Matching

The problem addressed here is the comparison of two syntactic structures that share identical terminal nodes (the words of the annotated sentence).

```
proc compare(A, B)
  for each non-terminal node X in A:
    search node Y in B
      such that yield(X) = yield(Y)
    if Y exists:
      emit different labels if any
    if Y does not exist:
      emit X and its yield
  end
end
```

Figure 4: Basic asymmetric algorithm to compare annotation A with annotation B of the same sentence

(Calder, 1997) presents a method of comparing the structure of context free trees found in different annotations. This section presents an extension of this algorithm that compares predicate-argument structures possibly containing crossing branches (cf. figure 2). Node and edge labels, representing phrasal categories and grammatical functions, are also taken into account.

Phrasal (non-terminal) nodes are compared on the basis of their yields: the yield of a nonterminal node X in an annotation A is the ordered set of terminals that are (directly or indirectly) dominated by X. The yield need not be contiguous since predicate-argument structures allow discontinuous constituents.

If both annotations contain nonterminal nodes that cover the same terminal nodes, the labels of the nonterminal nodes and their edges are compared.

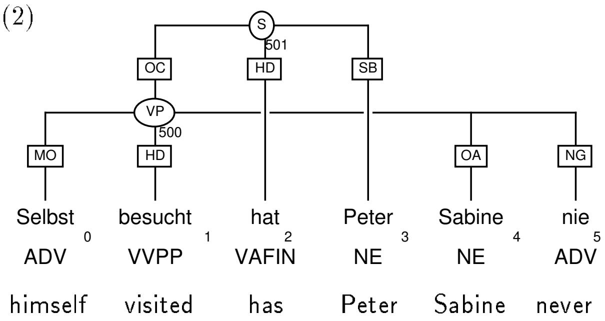
This results in a combined measure of structural and labeling differences, which is very useful in cleaning the corpus and keeping track of the development of the treebank.

We use the basic algorithm shown in figure 4 to determine the differences in two annotations A and B. The basic form is asymmetric. Therefore, a complete comparison consists of two runs, one for each direction, and the outputs of both runs are combined.

Figures 5 and 6 show examples of the output of the algorithm. These outputs can be directly used to mark the corresponding nodes and edges.

The yield is sufficient to uniquely determine corresponding nodes since the annotations used here do not contain unary branching nodes. If unary branching occurs, both the parent and the child have the same terminal yield and further mechanism to determine corresponding nodes are needed. (Calder, 1997) points out possible solutions to this problem.

(2)

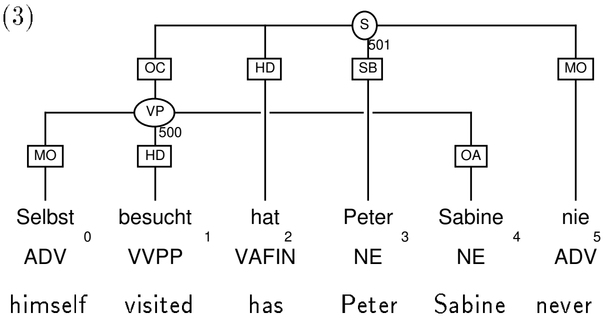


'Peter never visited Sabine himself'

```
sentence 1 errors 1
(1) structure: 500 VP [OC] 0 1 4
               (Selbst besucht Sabine)
(2) structure: 500 VP [OC] 0 1 4 5
               (Selbst besucht Sabine nie)
```

Figure 5: Erroneous annotation (2) of the example sentence in figure 2 (*nie* should be attached to S instead of VP), together with the output of the tree comparison algorithm. All nodes are numbered to enable identification. Additionally, this output can be used to highlight the corresponding nodes and edges.

(3)



'Peter never visited Sabine himself'

```
sentence 1 errors 1
(1) edge: 5 (ADV) [NG] nie
(3) edge: 5 (ADV) [MO] nie
```

Figure 6: Erroneous annotation (3) of the example sentence in figure 2 (*nie* should have grammatical function NG instead of MO), together with the output of the tree comparison algorithm.

4.2 Probabilities

The probabilities of each sub-structure of depth one are calculated separately according to the model described in sections 3.1 and 3.2. Subsequently, the product of these probabilities is used as a scoring function for the complete structure. This method is based on the assumption that productions at different levels in a structure are independent, which is inherent to context free rules.

Using the formulas from sections 3.1 and 3.2, the probability $P(A)$ of an annotation A is evaluated as

$$\begin{aligned} P(A) &= \prod_{i=1}^{n_{nt}} P(Q_i) \\ &= \prod_{i=1}^{n_{nt}} P_{Qi}(T_i, G_i) \\ &= \prod_{i=1}^{n_{nt}} \prod_{j=1}^{k_i} P_{Qi}(g_{i,j} | g_{i,j-2}, g_{i,j-1}) \\ &\quad \cdot P_Q(t_{i,j} | g_{i,j}) \end{aligned}$$

A	annotation (structure) for a sentence
n_{nt}	number of nonterminal nodes in A
n_t	number of terminal nodes in A
n	number of nodes $= n_{nt} + n_t$
Q_i	ith phrase in A
T_i	sequence of tags in Q_i
G_i	sequence of gramm. func. in Q_i
k_i	number of elements in Q_i
$t_{i,j}$	tag of jth child in Q_i
$g_{i,j}$	grammatical function of jth child in Q_i

Probabilities computed in this way cannot be used directly to compare two annotations since they favor annotations with fewer nodes. Each new nonterminal node introduces a new element in the product and makes it smaller.

Therefore, we normalize the probabilities w.r.t. the number of nodes in the annotation, which yields the perplexity $PP(A)$ of an annotation A:

$$PP(A) = \sqrt[n]{\frac{1}{P(A)}} \qquad (6)$$

4.3 Application to a Corpus

The procedures of tree matching and probability calculation were applied to our corpus, which currently consists of approx. 6300 sentences (115,000 tokens) of German newspaper text, each sentence annotated at least twice.

We measured the agreement of independent annotations after first annotation but before correction

Table 4: Comparison of independent semi-automatic annotations ⟨1⟩ after first, independent annotation and ⟨2⟩ after comparison but before the final discussion (current stage).

		⟨1⟩	⟨2⟩
	word level:		
(1)	ident. parent node	92.3%	98.7%
(2)	ident. gram. func.	93.8%	99.1%
	node level:		
(3)	identical nodes	87.6%	98.1%
(4)	identical nodes/labels	84.2%	97.4%
(5)	ident. node/gram. func.	76.6%	96.3%
	sentence level:		
(6)	identical structure	48.6%	90.8%
(7)	identical annotation	34.6%	87.9%

⟨1⟩, and after correction but before the final discussion ⟨2⟩, which is the current stage of the corpus. The results are shown in table 4.

As for measuring differences, we can count them at word, node and sentence level.

At the word level, we are interested in (1) the number of correctly assigned parent categories (does a word belong to a PP, NP, etc.?), and (2) the number of correctly assigned grammatical functions (is a word a head, modifier, subject, etc.?).

At the node level (non-terminals, phrases) we measure (3) the number of identical nodes, i.e., if there is a node in one annotation, we check whether it corresponds to a node in the other annotation having the same yield. Additionally, we count (4) the number of identical nodes having the same phrasal category, and (5) the number of identical nodes having the same phrasal category and the same grammatical function within its parent phrase.

At the sentence level, we measure (6) the number of annotated sentences having the same structure, and, which is the strictest measure, (7) the number of sentences having the same structure and the same labels (i.e., exactly the same annotation).

At the node level, we find 87.6% agreement in independent annotations. A large amount of the differences come from misinterpretation of the annotation guidelines by the annotators and are eliminated after comparison, which results in 98.1% agreement. This kind of comparison is the one most frequently used in the statistical parsing community for comparing parser output.

The sentence level is the strictest measure, and the agreement is low (34.6% identical annotations after first annotation, 87.9% after comparison). But at this level, one error (e.g. a wrong label) renders the whole annotation to be wrong and the sentence counts as an error.

Table 5: Using model perplexities to compare different annotations: Accuracy of using the hypothesis that a correct annotation has a lower perplexity than a wrong annotation.

recall	precision
30%	95.3%
45%	92.2%
60%	88.6%
85%	81.4%
100%	65.8%

If we make the assumption that a correct annotation always has a lower perplexity than a wrong annotation for the same sentence, the system would make a correct decision for 65.8% of the sentences (see table 5, last row).

For approx. 70% of all sentences, at least one of the initial annotations was completely correct. This means that the two initial annotations and the automatic comparison yield a corpus with approx. $65.8\% \times 70\% = 46\%$ completely correct annotations (complete structure and all tags).

One can further increase precision at the cost of recall by requiring the difference in perplexity to exceed some minimum distance. This precision/recall tradeoff is also shown in table 5.

5 Conclusion

The techniques and automatic tools described in this paper are designed to support annotation proper, online/offline consistency checking and the comparison of independent annotations of the same sentences. Most of the techniques employ stochastic processing methods, which guarantee high accuracy and robustness.

The bootstrapping approach adopted in our project makes the degree of automation a function of available training data. Easier processing tasks are automated first. Experience gained and data annotated at a lower level allow to increase the level of automation step by step. The current size of our corpus (approx. 6300 sentences) enables reliable automatic assignment of category and function labels as well as simple structures.

Future work will be concerned with developing automatic annotation methods handling complex structures, which should ultimately lead to the development of a parser for predicate-argument trees containing crossing branches.

6 Acknowledgements

This work is part of the DFG Sonderforschungsbereich 378 *Resource-Adaptive Cognitive Processes*, Project C3 *Concurrent Grammar Processing*.

We wish to thank the universities of Stuttgart and Tübingen for kindly providing us with a hand-corrected part-of-speech tagged corpus. We also wish to thank Oliver Plaehn, who did a great job in implementing the annotation tool, and Peter Schäfer, who built the tree comparison tool. Special thanks go to the five annotators continually increasing the size and the quality of our corpus. And finally, we thank Sabine Kramp for proof-reading this paper.

References

Black, Ezra, Stephen Eubank, Hideki Kashioka, David Magerman, Roger Garside, and Geoffrey Leech. 1996. Beyond skeleton parsing: Producing a comprehensive large-scale general-English treebank with full grammatical analysis. In *Proc. of COLING-96*, pages 107–113, Kopenhagen, Denmark.

Brants, Thorsten, Wojciech Skut, and Brigitte Krenn. 1997. Tagging grammatical functions. In *Proceedings of EMNLP-97*, Providence, RI, USA.

Brown, P. F., V. J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Calder, Jo. 1997. On aligning trees. In *Proc. of EMNLP-97*, Providence, RI, USA.

Church, Kenneth Ward. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. Second Conference on Applied Natural Language Processing*, pages 136–143, Austin, Texas, USA.

Cutting, Doug, Julian Kupiec, Jan Pedersen, and Penelope Sibun. 1992. A practical part-of-speech tagger. In *Proceedings of the 3rd Conference on Applied Natural Language Processing (ACL)*, pages 133–140.

Feldweg, Helmut. 1995. Implementation and evaluation of a german hmm for pos disambiguation. In *Proceedings of EACL-SIGDAT-95 Workshop*, Dublin, Ireland.

Marcus, Mitchell, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: the Penn Treebank. In Susan Armstrong, editor, *Using Large Corpora*. MIT Press.

Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of EMNLP-97*, Providence, RI, USA.

Skut, Wojciech, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. 1997a. Annotating unrestricted German text. In *Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*, Heidelberg, Germany.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997b. An annotation scheme for free word order languages. In *Proceedings of ANLP-97*, Washington, DC.

Thielen, Christine and Anne Schiller. 1995. Ein kleines und erweitertes Tagset fürs Deutsche. In *Tagungsberichte des Arbeitstreffens Lexikon + Text 17./18. Februar 1994, Schloß Hohentübingen. Lexicographica Series Maior*, Tübingen. Niemeyer.

Appendix A: Tagsets

This section contains descriptions of tags used in this paper. These are *not* complete lists.

A.1 Part-of-Speech Tags

We use the Stuttgart-Tübingen-Tagset. The complete set is described in (Thielen and Schiller, 1995).

ADJA	attributive adjective
ADV	adverb
APPR	preposition
ART	article
NE	proper noun
NN	common noun
PROAV	pronominal adverb
VAFIN	finite auxiliary
VAINF	infinite auxiliary
VMFIN	finite modal verb
VVPP	past participle of main verb

A.2 Phrasal Categories

AP	adjective phrase
MPN	multi-word proper noun
NP	noun phrase
PP	prepositional phrase
S	sentence
VP	verb phrase

A.3 Grammatical Functions

AC	adpositional case marker
HD	head
MO	modifier
NG	negation
NK	noun kernel
OA	accusative object
OC	clausal object
PNC	proper noun component
SB	subject