TüSBL: A Similarity-Based Chunk Parser for Robust Syntactic Processing

Sandra Kübler
Seminar für Sprachwissenschaft
University of Tübingen
Wilhelmstr. 113
D-72074 Tübingen, Germany
kuebler@sfs.nphil.uni-tuebingen.de

Erhard W. Hinrichs
Seminar für Sprachwissenschaft
University of Tübingen
Wilhelmstr. 113
D-72074 Tübingen, Germany
eh@sfs.nphil.uni-tuebingen.de

ABSTRACT

Chunk parsing has focused on the recognition of partial constituent structures at the level of individual chunks. Little attention has been paid to the question of how such partial analyses can be combined into larger structures for complete utterances.

The TüSBL parser extends current chunk parsing techniques by a tree-construction component that extends partial chunk parses to complete tree structures including recursive phrase structure as well as function-argument structure. TüSBL's tree construction algorithm relies on techniques from memory-based learning that allow similarity-based classification of a given input structure relative to a pre-stored set of tree instances from a fully annotated treebank.

A quantitative evaluation of TüSBL has been conducted using a semi-automatically constructed treebank of German that consists of appr. 67,000 fully annotated sentences. The basic PARSEVAL measures were used although they were developed for parsers that have as their main goal a complete analysis that spans the entire input. This runs counter to the basic philosophy underlying TüSBL, which has as its main goal robustness of partially analyzed structures.

Keywords

robust parsing, chunk parsing, similarity-based learning

1. INTRODUCTION

Current research on natural language parsing tends to gravitate toward one of two extremes: robust, partial parsing with the goal of broad data coverage versus more traditional parsers that aim at complete analysis for a narrowly defined set of data. Chunk parsing [1, 2] offers a particularly promising and by now widely used example of the former kind. The main insight that underlies the chunk parsing strategy is to isolate the (finite-state) analysis of non-recursive, syntactic structure, i.e. chunks, from larger, recursive structures. This results in a highly-efficient parsing architecture that is realized as a cascade of finite-state transducers and that pursues a longest-match, right-most pattern-matching strategy at each level of analysis.

Despite the popularity of the chunk parsing approach, there seem to be two apparent gaps in current research:

1. Chunk parsing research has focused on the recognition of partial constituent structures at the level of individual chunks. By comparison, little or no attention has been paid to the question of how such partial analyses can be combined into larger structures for complete utterances.

2. Relatively little has been reported on quantitative evaluations of chunk parsers that measure the correctness of the output structures obtained by a chunk parser.

The main goal of the present paper is help close those two research gaps.

2. THE TÜSBL ARCHITECTURE

In order to ensure a robust and efficient architecture, TüSBL, a similarity-based chunk parser, is organized in a three-level architecture, with the output of each level serving as input for the next higher level. The first level is part-of-speech (POS) tagging of the input string with the help of the bigram tagger LIKELY [10].[1] The parts of speech serve as pre-terminal elements for the next step, i.e. the chunk analysis. Chunk parsing is carried out by an adapted version of Abney's [2] scol parser, which is realized as a cascade of finite-state transducers. The chunks, which extend if possible to the simplex clause level, are then remodeled into complete trees in the tree construction level.

The tree construction is similar to the DOP approach [3, 4] in that it uses complete tree structures instead of rules. Contrary to Bod, we do not make use of probabilities and do not allow tree cuts, instead we only use the complete trees and minimal tree modifications. Thus the number of possible combinations of partial trees is strictly controlled. The resulting parser is highly efficient (3770 English sentences took 106.5 seconds to parse on an Ultra Sparc 10).

3. CHUNK PARSING AND TREE CONSTRUCTION

The division of labor between the chunking and tree construction modules can best be illustrated by an example.

[1]The inventory of POS tags is based on the Stuttgart-Tübingen Tagset (STTS) [11].

.

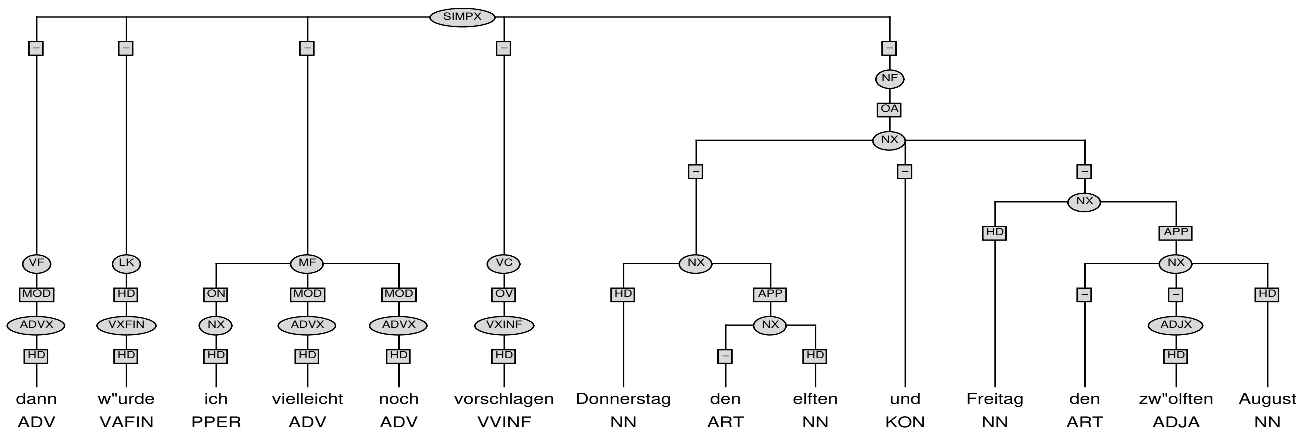

Figure 2: Sample tree construction output

Input:
dann w"urde ich vielleicht noch vorschlagen Donnerstag den elften und Freitag den zw"olften August
(then I would suggest maybe Thursday eleventh and Friday twelfth of August)

Chunk parser output:

```
[simpx [advx   [adv    dann]]
       [vxfin  [vafin  w"urde]]
       [nx2    [pper   ich]]
       [advx   [adv    vielleicht]]
       [advx   [advmd  noch]]
       [vvinf  vorschlagen]]

[nx3   [day    Donnerstag]
       [art    den]
       [adja   elften]]

[kon   und]

[nx3   [day     Freitag]
       [art     den]
       [adja    zw"olften]
       [month   August]]
```

Figure 1: Chunk parser output

For complex sentences such as the German input *dann w"urde ich vielleicht noch vorschlagen Donnerstag den elften und Freitag den zw"olften August* (then I would suggest maybe Thursday eleventh and Friday twelfth of August), the chunker produces a structure in which some constituents remain unattached or partially annotated in keeping with the chunk-parsing strategy to factor out recursion and to resolve only unambigous attachments, as shown in Fig. 1.

In the case at hand, the subconstituents of the extraposed coordinated noun phrase are not attached to the simplex clause that ends with the non-finite verb that is typically in clause-final position in declarative main clauses of German. Moreover, each conjunct of the coordinated noun phrase forms a completely flat structure. TüSBL's tree construction module enriches the chunk output as shown in Fig. 2[2]. Here the internally recursive NP conjuncts have been coordinated and integrated correctly into the clause as a whole. In addition, function labels such as *mod* (for: modifier), *hd* (for: head), *on* (for: subject), *oa* (for: direct object), and *ov* (for: verbal object) have been added that encode the function-argument structure of the sentence.

4. SIMILARITY-BASED TREE CONSTRUCTION

The tree construction algorithm is based on the machine learning paradigm of *memory-based learning* [12].[3] Memory-based learning assumes that the classification of a given input should be based on the similarity to previously seen instances of the same type that have been stored in memory. This paradigm is an instance of *lazy learning* in the sense that these previously encountered instances are stored "as is" and are crucially not abstracted over, as is typically the case in rule-based systems or other learning approaches. Past applications of memory-based learning to NLP tasks consist of classification problems in which the set of classes to be learnt is simple in the sense that the class items do not have any internal structure and the number of distinct items is small.

The use of a memory-based approach for parsing implies that parsing needs to be redefined as a classification task. There are two fundamentally different, possible approaches: the one is to split parsing up into different subtasks, that is, one needs separate classifiers for each functional category and for each level in a recursive structure. Since the classifiers for the functional categories as well as the individual decisions of the classifiers are independent, multiple or no candidates for a specific grammatical function or constituents with several possible functions may be found so that an additional classifier is needed for selecting the most appropriate assignment (cf. [6]).

The second approach, which we have chosen, is to regard the complete parse trees as classes so that the task is defined as the selection of the most similar tree from the instance base. Since in

[2]All trees in this contribution follow the data format for trees defined by the NEGRA project of the Sonderforschungsbereich 378 at the University of the Saarland, Saarbrücken. They were printed by the NEGRA annotation tool [5].

[3]Memory-based learning has recently been applied to a variety of NLP classification tasks, including part-of-speech tagging, noun phrase chunking, grapheme-phoneme conversion, word sense disambiguation, and pp attachment (see [9], [14], [15] for details).

```
construct_tree(chunk_list, treebank):
    while (chunk_list is not empty) do
        remove first chunk from chunk_list
        process_chunk(chunk, treebank)
```

Figure 3: Pseudo-code for tree construction, main routine.

```
process_chunk(chunk, treebank):
    words := string_yield(chunk)
    tree := complete_match(words, treebank)
    if (tree is not empty)                                  direct hit,
    then output(tree)                                       i.e. complete chunk found in treebank
    else
        tree := partial_match(words, treebank)
        if (tree is not empty)
        then
            if (tree = postfix of chunk)
            then
                tree1 := attach_next_chunk(tree, treebank)
                if (tree is not empty)
                then tree := tree1
            if ((chunk - tree) is not empty)                if attach_next_chunk succeeded
            then tree := extend_tree(chunk - tree, tree, treebank)   chunk might consist of both chunks
            output(tree)
            if ((chunk - tree) is not empty)                chunk might consist of both chunks (s.a.)
            then process_chunk(chunk - tree, treebank)      i.e. process remaining chunk
        else                                                back off to POS sequence
            pos := pos_yield(chunk)
            tree := complete_match(pos, treebank)
            if (tree is not empty)
            then output(tree)
            else                                            back off to subchunks
                while (chunk is not empty) do
                    remove first subchunk c1 from chunk
                    process_chunk(c1, treebank)
```

Figure 4: Pseudo-code for tree construction, subroutine process_chunk.

this case, the internal structure of the item to be classified (i.e. the input sentence) and of the class item (i.e. the most similar tree in the instance base) need to be considered, the classification task is much more complex, and the standard memory-based approach needs to be adapted to the requirements of the parsing task.

The features TüSBL uses for classification are the sequence of words in the input sentence, their respective POS tags and (to a lesser degree) the labels in the chunk parse. Rather than choosing a bag-of-words approach, since word order is important for choosing the most similar tree, the algorithm needed to be modified in order to rely more on sequential information.

Another modification was necessitated by the need to generalize from the limited number of trees in the instance base. The classification is simple only in those cases where a direct hit is found, i.e. where a complete match of the input with a stored instance exists. In all other cases, the most similar tree from the instance base needs to be modified to match the chunked input.

If these strategies for matching complete trees fail, TüSBL attempts to match smaller subchunks in order to preserve the quality of the annotations rather than attempt to pursue only complete parses.

The algorithm used for tree construction is presented in a slightly simplified form in Figs. 3-6. For readability's sake, we assume here that chunks and complete trees share the same data structure so that subroutines like *string_yield* can operate on both of them indiscriminately.

The main routine *construct_tree* in Fig. 3 separates the list of input chunks and passes each one to the subroutine *process_chunk* in Fig. 4 where the chunk is then turned into one or more (partial) trees. *process_chunk* first checks if a complete match with an instance from the instance base is possible.[4] If this is not the case, a partial match on the lexical level is attempted. If a partial tree is found, *attach_next_chunk* in Fig. 5 and *extend_tree* in Fig. 6 are used to extend the tree by either attaching one more chunk or by resorting to a comparison of the missing parts of the chunk with tree extensions on the POS level. *attach_next_chunk* is necessary to ensure that the best possible tree is found even in the rare case that the original segmentation into chunks contains mistakes. If no partial tree is found, the tree construction backs off to finding a complete match in the POS level or to starting the subroutine for processing a chunk recursively with all the subchunks of the present chunk.

The application of memory-based techniques is implemented in the two subroutines *complete_match* and *partial_match*. The presentation of the two cases as two separate subroutines is for expository purposes only. In the actual implementation, the search is carried out only once. The two subroutines exist because of

[4] *string_yield* returns the sequence of words included in the input structure, *pos_yield* the sequence of POS tags.

```
attach_next_chunk(tree, treebank):
    take first chunk chunk2 from chunk_list
    words2 := string_yield(tree, chunk2)
    tree2 := complete_match(words2, treebank)
    if (tree2 is not empty)
    then
        remove chunk2 from chunk_list
        return tree2
    else return empty
```

attempts to attach the next chunk to the tree

Figure 5: Pseudo-code for tree construction, subroutine attach_next_chunk.

```
extend_tree(rest_chunk, tree, treebank):
    words := string_yield(tree)
    rest_pos := pos_yield(rest_chunk)
    tree2 := partial_match(words + rest_pos, treebank)
    if ((tree2 is not empty) and (subtree(tree, tree2)))
    then return tree2
    else return empty
```

extends the tree on basis of POS comparison

Figure 6: Pseudo-code for tree construction, subroutine extend_tree.

the postprocessing of the chosen tree which is necessary for partial matches and which also deviates from standard memory-based applications. Postprocessing mainly consists of shortening the tree from the instance base so that it covers only those parts of the chunk that could be matched. However, if the match is done on the lexical level, a correction of tagging errors is possible if there is enough evidence in the instance base. TüSBL currently uses an *overlap metric*, the most basic metric for instances with symbolic features, as its similarity metric. This overlap metric is based on either lexical or POS features. Instead of applying a more sophisticated metric like the weighted overlap metric, TüSBL uses a backing-off approach that heavily favors similarity of the input with pre-stored instances on the basis of substring identity. Splitting up the classification and adaptation process into different stages allows TüSBL to prefer analyses with a higher likelihood of being correct. This strategy enables corrections of tagging and segmentation errors that may occur in the chunked input.

4.1 Example

Input:
dann w”urde ich sagen ist das vereinbart

(then I would say this is arranged)
Chunk parser output:

```
[simpx     [advx     [adv dann]]
           [vxfin    [vafin w"urde]]
           [nx2      [pper ich]]
           [vvinf    sagen]]

[simpx     [vafin    ist]
           [nx2      [pds das]]
           [vvpp     vereinbart]]
```

Figure 7: Chunk parser output

For the input sentence *dann w”urde ich sagen ist das vereinbart* (then I would say this is arranged), the chunked output is shown in Fig. 7. The chunk parser correctly splits the input into two clauses *dann w”urde ich sagen* and *ist das vereinbart*. A look-up in the instance base finds a direct hit for the first clause. Therefore, the correct tree can be output directly. For the second clause, only a partial match on the level of words can be found. The system finds the tree for the subsequence of words *ist das*, as shown in Fig. 8. By backing off to a comparison on the POS level, it finds a tree for the sentence *hatten die gesagt* (they had said) with the same POS sequence and the same structure for the first two words. Thus the original tree that covers only two words is extended via the newly found tree. TüSBL's output for the complete sentence is shown in Fig. 9.

Table 1: Quantitative evaluation

	minimum	maximum	average
precision	76.82%	77.87%	77.23%
recall	66.90%	67.65%	67.28%
crossing accuracy	93.44%	93.95%	93.70%

5. QUANTITATIVE EVALUATION

A quantitative evaluation of TüSBL has been conducted using a semi-automatically constructed treebank of German that consists of appr. 67,000 fully annotated sentences or sentence fragments.[5] The evaluation consisted of a ten-fold cross-validation test, where the training data provide an instance base of already seen cases for TüSBL's tree construction module.

The evaluation focused on three PARSEVAL measures: labeled precision, labeled recall and crossing accuracy, with the results shown in Table 1.

While these results do not reach the performance reported for other parsers (cf. [7], [8]), it is important to note that the task carried out here is more difficult in a number of respects:

1. The set of labels does not only include phrasal categories, but also functional labels marking grammatical relations such as subject, direct object, indirect object and modifier. Thus, the evaluation carried out here is not subject to the justified criticism levelled against the gold standards that are typically

[5] See [13] for further details.

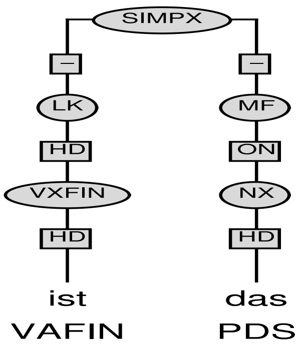

Figure 8: A partial tree found be the system

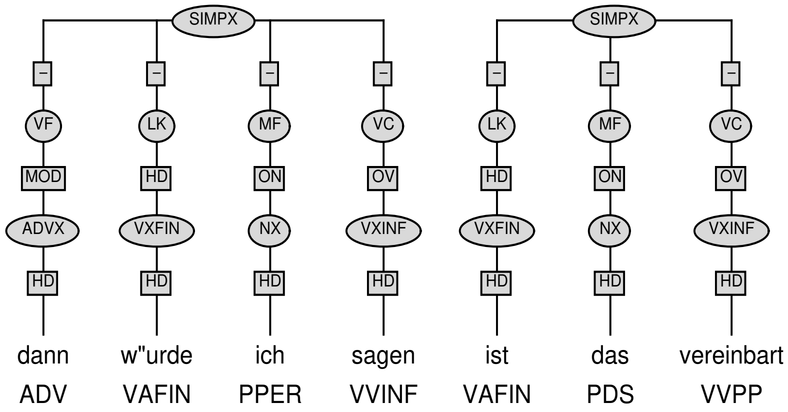

Figure 9: TüSBL's output for the complete sentence

in conjunction with the PARSEVAL measures, namely that the gold standards used typically do not include annotations of syntactic-semantic dependencies between bracketed constituents.

2. The German treebank consists of transliterated spontaneous speech data. The fragmentary and partially ill-formed nature of such spoken data makes them harder to analyze than written data such as the Penn treebank typically used as gold standard.

It should also be kept in mind that the basic PARSEVAL measures were developed for parsers that have as their main goal a complete analysis that spans the entire input. This runs counter to the basic philosophy underlying an amended chunk parser such as TüSBL, which has as its main goal robustness of partially analyzed structures: Precision and recall measure the percentage of brackets, i.e. constituents with the same yield or bracketing scope, which are identical in the parse tree and the gold standard. If TüSBL finds only a partial grouping on one level, both measures consider this grouping wrong, as a consequence of the different bracket scopes. In most cases, the error 'percolates' up to the highest level. Fig. 10 gives an example of a partially matched tree structure for the sentence "bei mir ginge es im Februar ab Mittwoch den vierten" (for me it would work in February after Wednesday the fourth). The only missing branch is the branch connecting the second noun phrase (NX) above "Mittwoch" to the NX "den vierten". This results in precision and recall values of 10 out of 15 because of the altered bracketing scopes of the noun phrase, the two prepositional phrases (PX), the field level (MF) and the sentence level (SIMPX).

In order to capture this specific aspect of the parser, a second evaluation was performed that focused on the quality of the structures produced by the parser. This evaluation consisted of manually judging the TüSBL output and scoring the accuracy of the recognized constituents. The scoring was performed by the human annotator who constructed the treebank and was thus in a privileged position to judge constituent accuracy with respect to the treebank annotation standards. This manual evaluation resulted in a score of 92.4% constituent accuracy; that is: of all constituents that were recognized by the parser, 92.4% were judged correct by the human annotator. This seems to indicate that approximately 20% of the precision errors are due to partial constituents whose yield is shorter than in the corresponding gold standard. Such discrepancies typically arise when TüSBL outputs only partial trees. This occurs when no complete tree structures can be constructed that span the entire input.

6. CONCLUSION AND FUTURE RESEARCH

In this paper we have described how the TüSBL parser extends current chunk parsing techniques by a tree-construction component that completes partial chunk parses to tree structures including function-argument structure.

As noted in section 4, TüSBL currently uses an *overlap metric*, i. e. the most basic metric for instances with symbolic features, as its

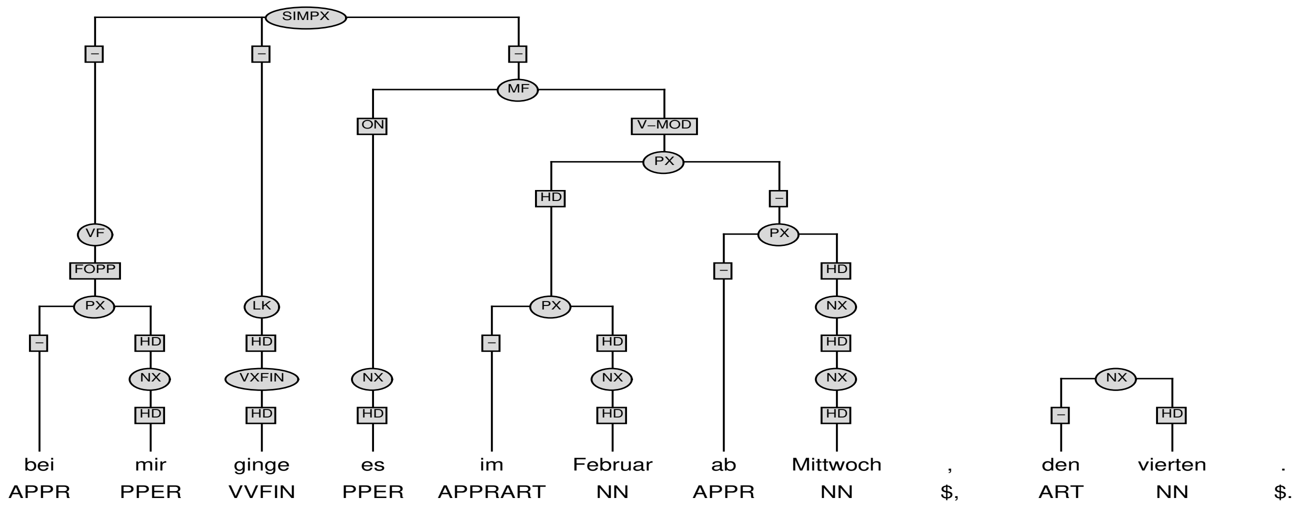

Figure 10: A partially grouped tree output of the TÜSBL system

similarity metric. We anticipate that the results reported in Fig. 1 can be further improved by experimenting with more sophisticated similarity metrics. However, we will have to leave this matter to future research.[6]

7. ACKNOWLEDGMENTS

The research reported here was funded both by the German Federal Ministry of Education, Science, Research, and Technology (BMBF) in the framework of the VERBMOBIL Project under Grant 01 IV 101 N 0 and by the Deutsche Forschungsgemeinschaft (DFG) in the framework of the Sonderforschungsbereich 441.

8. REFERENCES

[1] S. Abney. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenney, editors, *Principle-Based Parsing*. Kluwer Academic Publishers, 1991.

[2] S. Abney. Partial parsing via finite-state cascades. In J. Carroll, editor, *Workshop on Robust Parsing (ESSLLI '96)*, 1996.

[3] R. Bod. *Beyond Grammar: An Experience-Based Theory of Language*. CSLI Publications, Stanford, California, 1998.

[4] R. Bod. Parsing with the shortest derivation. In *Proceedings of COLING 2000*, 2000.

[5] T. Brants and W. Skut. Automation of treebank annotation. In *Proceedings of NeMLaP-3/CoNLL98, Sydney, Australia*, 1998.

[6] S. Buchholz, J. Veenstra, and W. Daelemans. Cascaded grammatical relation assignment. In *Proceedings of EMNLP/VLC-99, University of Maryland, USA, June 21-22, 1999*, pages 239 – 246, 1999.

[7] E. Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artifical Intelligence, Menlo Park*, 1997.

[8] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[9] W. Daelemans, J. Zavrel, and A. van den Bosch. Forgetting exceptions is harmful in language learning. *Machine Learning: Special Issue on Natural Language Learning*, 34, 1999.

[10] H. Feldweg. Stochastische Wortartendisambiguierung für das Deutsche: Untersuchungen mit dem robusten System LIKELY. Technical report, Universität Tübingen, 1993. SfS-Report-08-93.

[11] A. Schiller, S. Teufel, and C. Thielen. Guidelines für das Tagging deutscher Textkorpora mit STTS. Technical report, Universität Stuttgart and Universität Tübingen, 1995. (URL: http://www.sfs.nphil.uni-tuebingen.de/Elwis/stts/stts.html).

[12] C. Stanfill and D. Waltz. Towards memory-based reasoning. *Communications of the ACM*, 29(12), 1986.

[13] R. Stegmann, H. Schulz, and E. W. Hinrichs. Stylebook for the German Treebank in VERBMOBIL. Technical Report 239, Verbmobil, 2000.

[14] J. Veenstra, A. van den Bosch, S. Buchholz, W. Daelemans, and J. Zavrel. Memory-based word sense disambiguation. *Computers and the Humanities, Special Issue on Senseval, Word Sense Disambiguations*, 34, 2000.

[15] J. Zavrel, W. Daelemans, and J. Veenstra. Resolving PP attachment ambiguities with memory-based learning. In M. Ellison, editor, *Proceedings of the Workshop on Computational Natural Language Learning (CoNLL'97), Madrid*, 1997.

[6][9] reports that the *gain ratio* similarity metric has yielded excellent results for the NLP applications considered by these investigators.