

On the Automatic Construction of Grammars from the
NEGRA Trebank

Mike Daniels

LING 795K

5 December 2002

Introduction

- Basic idea: Every treebank contains an implicit grammar. For every mother A that dominates $B \dots C$, we take the rule $A \rightarrow B \dots C$.

- This grammar is both large and redundant. We can therefore construct a similar grammar with the same coverage yet fewer rules and use this grammar to parse the sentences in the treebank.

- Initial goal: to develop a set of methods that yield the smallest grammar possible without sacrificing coverage.
- On the other hand, early indications are that parsing efficiency is determined more by the “branching factor” of each grammar symbol, rather than the overall number of rules.
- Revised goal: to develop a set of methods that simultaneously minimize the number of rules in the grammar as well as the branching factor of the grammar.

NEGRA

- A fairly large treebank for German.
- All nodes (except the root of a tree) are given both a part-of-speech tag (noun, verb, etc.) as well as a grammatical function label (subject, object, modifier, etc.). For our purposes, the tag-label pair was taken as an atomic symbol.
- For this project, a subset of 10708 sentences was chosen: those that both:
 - were rooted in the category S.
 - were devoid of coordination.

Outline

1. Annotate the corpus with order information.
2. Untangle the annotated corpus.
3. Extract rules and remove duplicates.
4. Lift optionals from rules and remove duplicates.
5. Cluster the rules.
6. Extract LP information from clusters.
7. Turn the text rules into a prolog grammar.
8. Extract lexicon and remove duplicates.
9. Turn the lexicon into a prolog lexicon.
10. Generate a set of test sentences.
11. Turn the test sentences into prolog queries.

- Technology
- XSL interpreter (XML manipulation)
- Perl (text processing)
- sort

Order Annotation

- All tree nodes in the NEGRA corpus are assigned an ID number. Leaves start at 1 and increase left-to-right, while interior nodes start at 500 and increase left-to-right, bottom-to-top. Thus no node refers to something that follows it.
- The daughters of an interior node are listed in order of ID number.
- Thus the information about the linear order of the daughters of an interior node is obscured and must be recovered.
- We first mark each terminal with its left edge (i.e. position), and then each nonterminal with its left edge (the minimum of all daughters' left edges).

Untangling

- While the part-of-speech tags are stored on the nodes themselves, the labels are stored on the mother nodes. Since we want to use the tag and label together as a category, they must be brought together.
- This is also the point where the set of trees is turned into a set of rules. For each rule, its right-hand side is sorted according to the left edge information added in the last step.

Lifting Optional Elements

- Much of the redundancy in the initial grammar arises from optional elements.
- That is, if there are five categories that could modify a given head, there are likely to be rules in the grammar in which each of the five occurs once with the head, each of the five occurs multiply with the head, or multiple categories appear multiply with the head.
- We can therefore reduce the number of rules in the grammar by lifting the optional categories into a separate rule.
- Currently, we go from 12367 rules to 5205 rules in this step.

Lifting Example

- This grammar:

– $A:OB \rightarrow B:OB \ C:OB \ D:OB$
 $A:OB \rightarrow E:OP \ F:OB \ G:OB$
 $A:OB \rightarrow H:OP \ I:OB \ J:OP$

becomes this grammar after lifting:

– $A:OB \rightarrow B:OB \ C:OB \ D:OB$
 $A:OB \rightarrow F:OB \ G:OB$
 $A:OB \rightarrow A:OB \ E:OP$
 $A:OB \rightarrow I:OB$
 $A:OB \rightarrow A:OB \ H:OP$
 $A:OB \rightarrow A:OB \ J:OP$

Order Generalization

- So far, each of the rules from the grammar has a strictly-ordered right-hand side. Since this is to be a grammar for a linearization parser, we can conflate rules that have the same constituents in multiple orders.

- For example, the rules:

– $A \rightarrow B D C$

$A \rightarrow B C D$

- would become

– $A \rightarrow B C D; 0 < 1, 0 < 2$

- This step takes us from 5205 rules to 4667 rules.

G12N Algorithm

- We first cluster the rules: that is, we store each rule in a hash table indexed by the sorted right-hand side and then dump each bucket in the hash table.

- For example,

$- A \rightarrow B C$
 $A \rightarrow B D$
 $A \rightarrow B C D$
 $A \rightarrow B D C$
 $A \rightarrow C B$
 $B \rightarrow B E$
 $B \rightarrow B E C D$

becomes

$- A \rightarrow B C$

$A \rightarrow C B$

$A \rightarrow B D$

$A \rightarrow B C D$

$A \rightarrow B D C$

$B \rightarrow B E$

$B \rightarrow B E C D$

G12N Algorithm 2

- Now, for each cluster, we generate the set of order constraints for each rule in that cluster, intersect the sets so generated, and output the result.
- Special handling is required when categories are duplicated within the rule.

- From the last example:

– $A \rightarrow B \ C$

$A \rightarrow B \ D; 0 < 1$

$A \rightarrow B \ C \ D; 0 < 1, 0 < 2$

$B \rightarrow B \ E; 0 < 1$

$B \rightarrow B \ C \ D \ E; 3 < 1, 3 < 2, 0 < 1, 0 < 2, 0 < 3, 1 < 2$
 $A \rightarrow B \ B \ B \ C; 0 < 1, 1 < 2, 0 < 2, 0 < 3$

Current status

- The final grammar of 4667 rules is still too large to be comfortably parsed by my parser.
- Indications seem to be that this is the result of the branching factor of the grammar. For example, 171 rules expand the NN:OA symbol.
- Future work will concentrate on reducing this branching factor. Two potential avenues:
 - Introducing more categories in the lifting process.
 - Giving the parser a better search strategy: find lexical categories before phrasal categories.