

# Improving the Efficiency of Parsing Discontinuous Constituents

Mike Daniels

6 December 2002

## Talk Structure

- Background and Motivation
- Definition of Task
- Solution and Outlook

## Discontinuous Constituents

- Basic idea: the categories dominated by a nonterminal need not dominate adjacent strings of terminals.
- Linked to concepts like tangled trees, linearization, wrapping, precedence, order domains, (relatively) free word order, etc.
- Two examples of alleged discontinuous constituents in English:
  1. He **picked** his daughter **up** from school yesterday.
  2. He is **an easy** man **to please**.

## Motivation

- There has been a growing acceptance of discontinuous constituents in recent years.
- In particular, one can point to the ongoing development of HPSG linearization grammar (e.g. Reape 1993, Kathol 2000, Müller 1999).
- Similarly, one sees increasing use of discontinuous constituents throughout syntactic frameworks and treebanks (NEGRA (Skut et al. 1997), VERBMOBIL (Hinrichs et al. 2000)).
- One corpus study (Krujiff 2002) found that 1% of all English constituents were discontinuous; for German, Dutch, and Czech, the percentages were 11%, 18%, and 58%.

## Motivation (Cont.)

- As it becomes more natural for linguists to use discontinuous constituents to analyze natural language, it becomes more important for parsers to be able to handle them efficiently.
- This efficient handling is currently lacking. Most current approaches for parsing HPSG linearization grammars use some variant of the generate-and-test paradigm: all constraints on word order are removed from the parser and relegated to a constraint solver.
- The goal of this research was therefore to develop a parser that could work directly with grammars using discontinuous constituents and efficiently parse according to those grammars.

So what does it actually mean to have a grammar that uses discontinuous constituents?

## Grammar Format

Based on the Linear Specification Language (Götz and Penn 1997; Suhre 1999).

We have **lexical entries** of the form  $A \rightarrow t$  and **grammar rules** of the form  $A \rightarrow \alpha; L$ , where:

- $\alpha$  is a **list of non-terminals**.
- $L$  is a set of **linearization constraints**:
  - $A < B$  (**precedence**): The terminals dominated by  $A$  all occur to the left of the terminals dominated by  $B$ .
  - $A \ll B$  (**immediate precedence**): The rightmost terminal dominated by  $B$  occurs immediately to the left of the leftmost terminal dominated by  $B$ .
  - $[A]$  (**isolation**):  $A$  dominates an uninterrupted sequence of terminals in the input.

- In a normal context-free grammar, the order of the right-hand side categories is used to encode linear order.
- But even though our format is no longer strongly tied to linear order, the right-hand side of a grammar rule is still stated as a list.
- The reason: the ordering is used to guide parsing; it need not reflect surface order.

- For example, the rule

$s \rightarrow \text{verb}_1 \text{ nom}_2 \text{ acc}_3; \{2 < 1, 3 < 1\}$

requires that the verb be parsed first, even though it must follow the other two constituents.

- In part, this format can be seen as relaxing the right-hand side of each rule from representing a total order on the categories listed to merely a partial order.

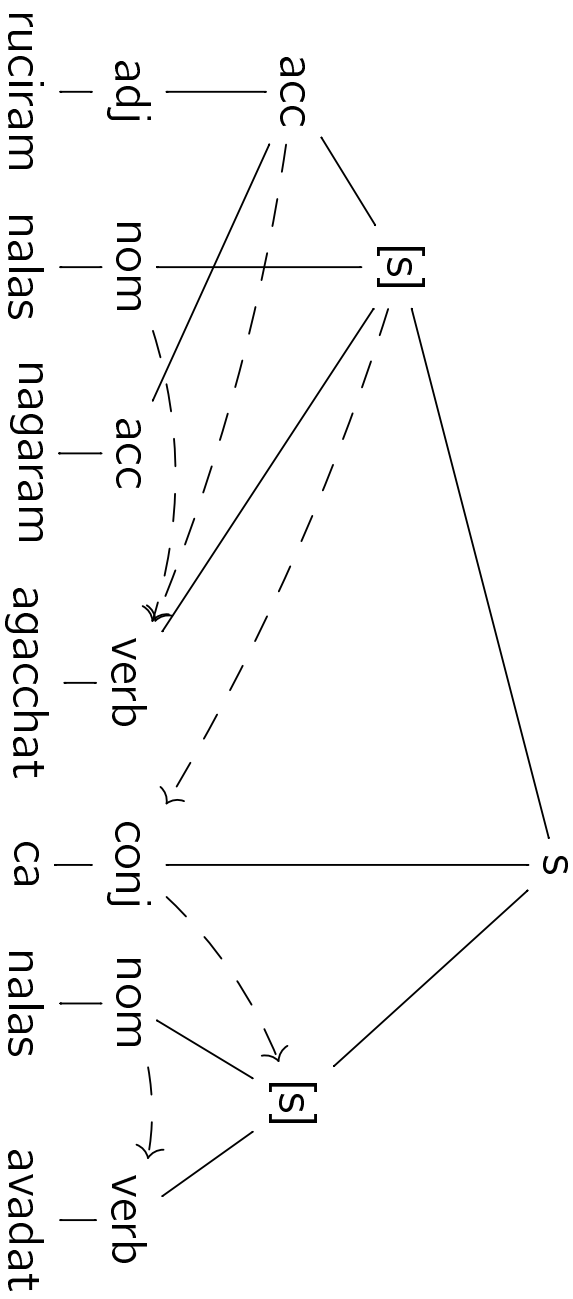
## Sample Grammar

1	$s \rightarrow \text{verb}_1 \text{ nom}_2 \text{ acc}_3; \{2 < 1, 3 < 1\}$	
2	$s \rightarrow \text{verb}_1 \text{ nom}_2; \{2 < 1\}$	
3	$s \rightarrow \text{conj}_1 s_2 s_3; \{2 \ll 1, 1 \ll 3, [2], [3]\}$	
4	$\text{acc} \rightarrow \text{adj}_1 \text{acc}_2; \{ \}$	
5	$\text{nalas} \rightarrow \text{nom}$	'Nala' (a proper name)
6	$\text{nagaram} \rightarrow \text{acc}$	'city'
7	$\text{agacchat} \rightarrow \text{verb}$	'went'
8	$\text{ca} \rightarrow \text{conj}$	'and'
9	$\text{avadat} \rightarrow \text{verb}$	'spoke'
10	$\text{ruciram} \rightarrow \text{adj}$	'shining'

## Example

(1) रुचिरं नलो नगरमगच्छन्नलो ऽवदत्

ruciram nalas nagaram agacchat ca nalas avadat  
 shining nala city went and nala spoke  
 'Nala went to the shining city and Nala spoke.'



(Solid lines represent dominance, while dashed arrows represent the precedence relations enforced by the grammar.)

## Parser Overview

- Representational issues
- The parsing algorithm itself

## Parsing Framework

- Based on Earley's Algorithm: a top-down, goal-directed approach.
- Also a bottom-up approach: driven by lexical information.
- Modified to start with lexical seeding and to use an active chart.

## Edge Coverage: Earley

- An active chart contains edges encoding in-process as well as completed parses of a (sub)string.
- Traditionally, edge coverage is modelled by an interval.
- For instance,  $\langle [0,2], S, [VP] \rangle$  might represent the edge for *The dog* in  ${}_0The_1dog_2sleeps_3$ .

## Edge Coverage: GIDL P

- GIDL P grammars allow for discontinuity, and so a single interval is no longer sufficient: a constituent may dominate any subset of the input.
- Following Johnson (1985) and Reape (1991), we represent edge coverage with bitvectors.
- Each active bit represents the inclusion of the word at that position; the left edge of the input corresponds to the left edge of the vector.

## Encoding Bitvectors

- We want to represent the coverage of the discontinuous accusative noun phrase *ruciram nagaram* in the example given earlier:

**ruciram**    **nalas**    **nagaram**    **agacchat**    **ca**    **nalas**    **avadat**

yes    no    yes    no    no    no    no

- The following are equivalent in terms of descriptive power:
  - Interval lists (Johnson 1985):  $[[0, 1], [2, 3]]$
  - Bit lists (Reape 1991):  $[1, 0, 1, 0, 0, 0, 0]$
  - Binary numbers:  $0000101_2 (= 5_{10})$(Notice that the least significant bit of the number corresponds to the left-most word in the input.)

## Computing With Bitvectors

- Parsing requires a number of bitwise operations: overlap, lbound, suffix, and so on. (See Appendix A in the handout for a full list.)
- The nature of current hardware allows bitwise operations to be computed in parallel in a single processor instruction.
- Thus it takes just as long to, say, compute the bitwise-AND of two 20-bit numbers as it does for two 30-bit numbers.
- In general, for a processor that has a word size of  $n$ , arithmetic computation time is proportional to the base- $n$  logarithm of the inputs. Operations on any list-based representation will take time proportional to the length of the list itself.

## Edge Constraints

- We also use bitvectors to encode **constraints** on coverage; bitvectors used this way are called bitmasks.
- We store a negative and positive bitmask on each edge, representing different aspects of precedence information.  
Assume we've found a noun as the third word of a sentence:
  - Negative: If we know that verbs follow nouns, we shouldn't look for verbs in the first or second position.
  - Positive: If we know that verbs immediately follow nouns, then any verb we find must cover at least the fourth position.

## Prediction

- Given an edge with a non-empty RHS, consider the first element of that RHS and see if any rules have that category as their LHS.

*Consider the edge*

$s : s_2 s_3; \{s_2 \lll 0010000, 0010000 \lll s_3, [s_2], [s_3]\}$ ,

Coverage = 0010000, NMask = 0010000

*which represents a point in the parse where the conjunction of rule 3 has been found as the fifth word of the sentence.*

*One qualifying rule in the grammar is number 3 again:*

$s \rightarrow c_1 s_2 s_3; \{2 \lll 1, 1 \lll 3\}$ .

- Compute the masks for each such rule by considering each order constraint in turn:
  - For precedence constraints, have we already found one of the appropriate categories? If so, apply a left- or right-mask, as appropriate.

*Here, we're predicting  $s_2$ , which is mentioned in the constraint  $s_2 \lll 0010000$ . Since the predicted element needs to precede a known location, we augment the negative mask with suffix(lbound(0010000)) = 1110000.*

- For immediate precedence, also augment the positive mask with the bit adjacent to the appropriate bound of the position.

*With the constraint  $s_2 \lll 0010000$ , we use the bit adjacent to the left bound of  $0010000$ : we augment the positive mask with  $0001000$ .*

- Make sure the negative mask has enough “holes” for the RHS of the current rule to fit into.

*The negative mask  $1110000$  does in fact have enough space for the three elements of the rule we’re predicting.*

- Enter the resulting edge into the chart.

*We end up with  $s : c s_2 s_3; \{s_2 \lll c, c \lll s_3\}$ ,*

*Coverage = 0000000, NMask = 1110000,*

*PMask = 0100000*

## Completion

- Given an active and a passive edge:

*Fast-forward to the point where we've recognized that the second, third, and fourth words together form a sentence:*

*Active:*

$s : s_2 s_3 ; \{s_2 \lll 0010000, 0010000 \lll s_3, [s_2], [s_3]\}$ ,

Coverage = 0010000, NMask = 0010000

*Passive:*

$s : ; \{ \}$ , Coverage = 0001110

- Check that the active edge's negative mask does not overlap with the passive edge's coverage.  
*0010000 does not overlap with 0001110.*

- If we're about to create a passive edge: check that the passive edge's coverage is bitwise-implied by the active edge's positive mask.

*Nothing to check here, since we'll still need to find  $s_3$ .*

- If the active edge contains an isolation constraint, make sure the terminals it would govern are contiguous.  
*0001110 is indeed contiguous.*

- Compute the new coverage and update the LP constraints.  
*Combining 0001110 and 0010000 yields 0011110. The LP constraints  $s_2 \ll 0010000$  and  $[s_2]$  are discarded, as they no longer refer to unfound elements.*
- Add the resulting edge to the chart.  
 $s : s_3; \{0010000 \ll s_3, [s_3]\},$   
Coverage = 0010000, NMask = 0010000

## Summary

- Grammar format generalizes ID/LP grammars to allow for discontinuous constituents.
- Parser generalizes Earley's algorithm to allow for discontinuous constituents; also allows the processing order of the RHS to be specified, thereby generalizing head-driven parsing.
- Coverage constraints encoded as bitvectors provide an efficient, compiled representation of linear precedence information.

## Advanced Issues

- Global precedence and isolation statements.
- Chart indexing strategies that avoid the enumeration of all bitvectors of length  $n$ .

## Future Work

- Move to complex categories (first compound terms, then feature structures).
- Large-scale grammar development and evaluation: automatic extraction from the NEGRA corpus, and recoding the grammar encoded in the Babel program (Müller 1996).