Statistical NLP

Spring 2009

University of California
CALNLP
Berkeley

Lecture 15: Parsing II

Dan Klein – UC Berkeley

Classical NLP: Parsing

- Write symbolic or logical rules:

Grammar (CFG)		Lexicon
ROOT → S	NP → NP PP	NN → interest
S → NP VP	VP → VBP NP	NNS → raises
NP → DT NN	VP → VBP NP PP	VBP → interest
NP → NN NNS	PP → IN NP	VBZ → raises
		…

- Use deduction systems to prove parses from words
  - Minimal grammar on "Fed raises" sentence: 36 parses
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses
- This scaled very badly, didn't yield broad-coverage tools

Probabilistic Context-Free Grammars

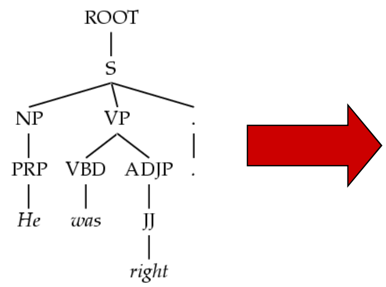
- A context-free grammar is a tuple $<N, T, S, R>$
  - *N* : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - *T* : the set of terminals (the words)
  - *S* : the start symbol
    - Often written as ROOT or TOP
    - *Not* usually the sentence non-terminal S
  - *R* : the set of rules
    - Of the form $X \rightarrow Y_1\ Y_2 \ldots Y_k$, with $X, Y_i \in N$
    - Examples: $S \rightarrow NP\ VP$, $VP \rightarrow VP\ CC\ VP$
    - Also called rewrites, productions, or local trees
- A PCFG adds:
  - A top-down production probability per rule $P(Y_1\ Y_2 \ldots Y_k \mid X)$

Treebank Sentences

```
( (S (NP-SBJ The move)
     (VP followed
         (NP (NP a round)
             (PP of
                 (NP (NP similar increases)
                     (PP by
                         (NP other lenders))
                     (PP against
                         (NP Arizona real estate loans)))))
         ,
         (S-ADV (NP-SBJ *)
                (VP reflecting
                    (NP (NP a continuing decline)
                        (PP-LOC in
                                (NP that market))))))
     .))
```

Treebank Grammars

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn't work well):

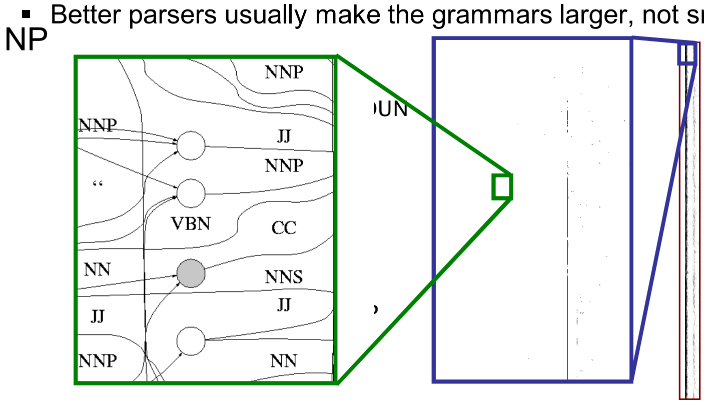


ROOT → S	1
S → NP VP .	1
NP → PRP	1
VP → VBD ADJP	1
…..	

- Better results by enriching the grammar (e.g., lexicalization).
- Can also get reasonable parsers without lexicalization.

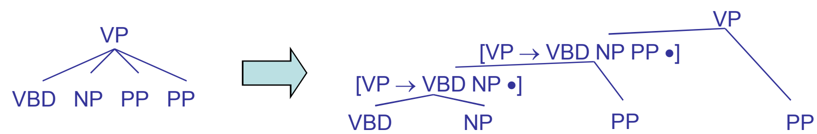
Treebank Grammar Scale

- Treebank grammars can be enormous
  - As FSAs, the raw grammar has ~10K states, excluding the lexicon
  - Better parsers usually make the grammars larger, not smaller



Chomsky Normal Form

- Chomsky normal form:
  - All rules of the form $X \rightarrow Y\ Z$ or $X \rightarrow w$
  - In principle, this is no limitation on the space of (P)CFGs
    - N-ary rules introduce new non-terminals



    - Unaries / empties are "promoted"
  - In practice it's kind of a pain:
    - Reconstructing n-aries is easy
    - Reconstructing unaries is trickier
    - The straightforward transformations don't preserve tree scores
  - Makes parsing algorithms simpler!

A Recursive Parser

```
bestScore(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max score(X->YZ) *
                  bestScore(Y,i,k) *
                  bestScore(Z,k,j)
```

- Will this parser work?
- Why or why not?
- Memory requirements?

A Memoized Parser

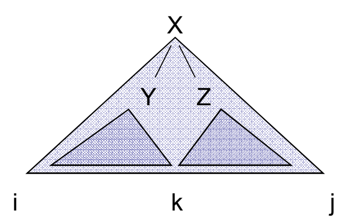
- One small change:

```
bestScore(X,i,j,s)
   if (scores[X][i][j] == null)
       if (j = i+1)
           score = tagScore(X,s[i])
       else
           score = max score(X->YZ) *
                       bestScore(Y,i,k) *
                       bestScore(Z,k,j)
       scores[X][i][j] = score
   return scores[X][i][j]
```

A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

```
bestScore(s)
   for (i : [0,n-1])
     for (X : tags[s[i]])
       score[X][i][i+1] =
           tagScore(X,s[i])
   for (diff : [2,n])
     for (i : [0,n-diff])
       j = i + diff
       for (X->YZ : rule)
         for (k : [i+1, j-1])
           score[X][i][j] = max score[X][i][j],
                                score(X->YZ) *
                                score[Y][i][k] *
                                score[Z][k][j]
```



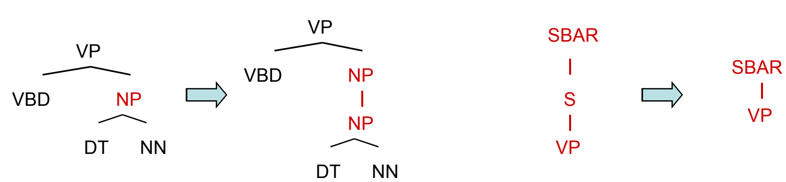
Unary Rules

- Unary rules?

```
bestScore(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max max score(X->YZ) *
                      bestScore(Y,i,k) *
                      bestScore(Z,k,j)
                  max score(X->Y) *
                      bestScore(Y,i,j)
```

CNF + Unary Closure

- We need unaries to be non-cyclic
  - Can address by pre-calculating the *unary closure*
  - Rather than having zero or more unaries, always have exactly one



  - Alternate unary and binary layers
  - Reconstruct unary chains afterwards

Alternating Layers

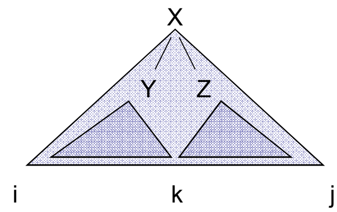
```
bestScoreB(X,i,j,s)
      return max max score(X->YZ) *
                     bestScoreU(Y,i,k) *
                     bestScoreU(Z,k,j)



bestScoreU(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max max score(X->Y) *
                      bestScoreB(Y,i,j)
```

Memory

- How much memory does this require?
  - Have to store the score cache
  - Cache size: |symbols|*n^2 doubles
  - For the plain treebank grammar:
    - X ~ 20K, n = 40, double ~ 8 bytes = ~ 256MB
    - Big, but workable.
- Pruning: Beams
  - score[X][i][j] can get too large (when?)
  - Can keep beams (truncated maps score[i][j]) which only store the best few scores for the span [i,j]
- Pruning: Coarse-to-Fine
  - Use a smaller grammar to rule out most X[i,j]
  - Much more on this later…

Time: Theory

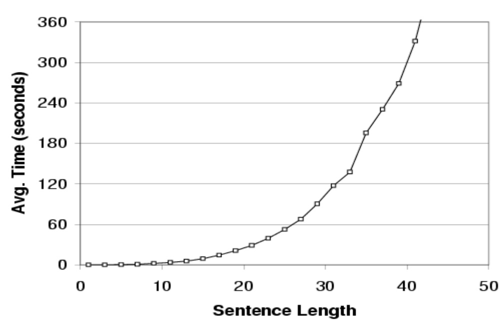
- How much time will it take to parse?
  - For each diff (<= n)
    - For each i (<= n)
      - For each rule X → Y Z
        - For each split point k
          Do constant work



  - Total time: |rules|*n^3
  - Something like 5 sec for an unoptimized parse of a 20-word sentences

Time: Practice

- Parsing with the vanilla treebank grammar:



~ 20K Rules

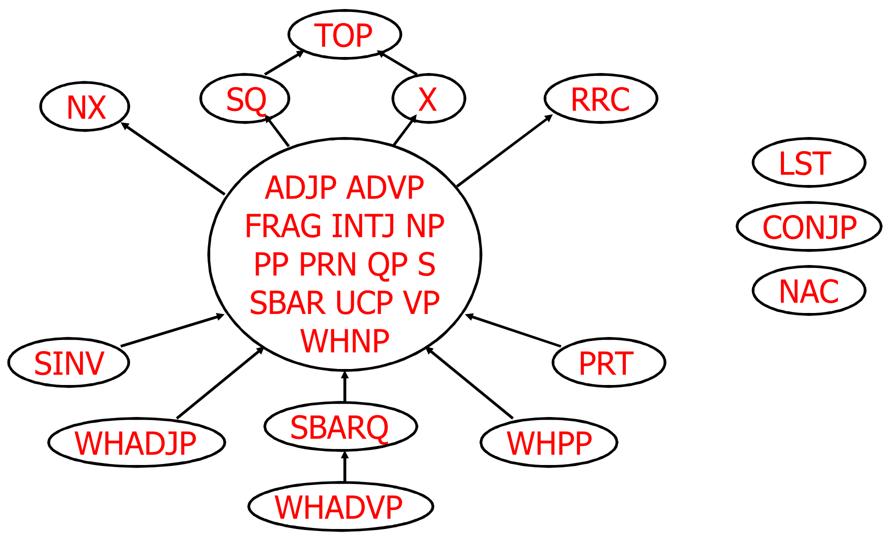
(not an optimized parser!)

Observed exponent:

3.6

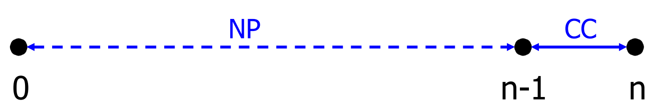
- Why's it worse in practice?
  - Longer sentences "unlock" more of the grammar
  - All kinds of systems issues don't scale

Same-Span Reachability



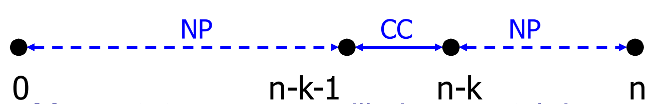
Rule State Reachability

Example: NP CC •



1 Alignment

Example: NP CC NP •

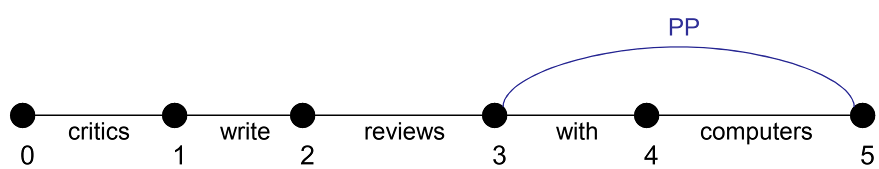


n Alignments

- Many states are more likely to match larger spans!

Agenda-Based Parsing

- Agenda-based parsing is like graph search (but over a hypergraph)
- Concepts:
  - Numbering: we number fenceposts between words
  - “Edges” or items: spans with labels, e.g. PP[3,5], represent the sets of trees over those words rooted at that label (cf. search states)
  - A chart: records edges we’ve expanded (cf. closed set)
  - An agenda: a queue which holds edges (cf. a fringe or open set)



Word Items

- Building an item for the first time is called discovery. Items go into the agenda on discovery.
- To initialize, we discover all word items (with score 1.0).

AGENDA

critics[0,1], write[1,2], reviews[2,3], with[3,4], computers[4,5]

CHART [EMPTY]

Unary Projection

- When we pop a word item, the lexicon tells us the tag item successors (and scores) which go on the agenda

critics[0,1]	write[1,2]	reviews[2,3]	with[3,4]	computers[4,5]
NNS[0,1]	VBP[1,2]	NNS[2,3]	IN[3,4]	NNS[4,5]

0 critics 1 write 2 reviews 3 with 4 computers 5

critics write reviews with computers

Item Successors

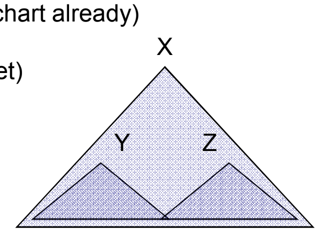
- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)

$$Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j]$$

  - Hypergraph successors: combine with items already in our chart

$$Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y\ Z \text{ form } X[i,k]$$

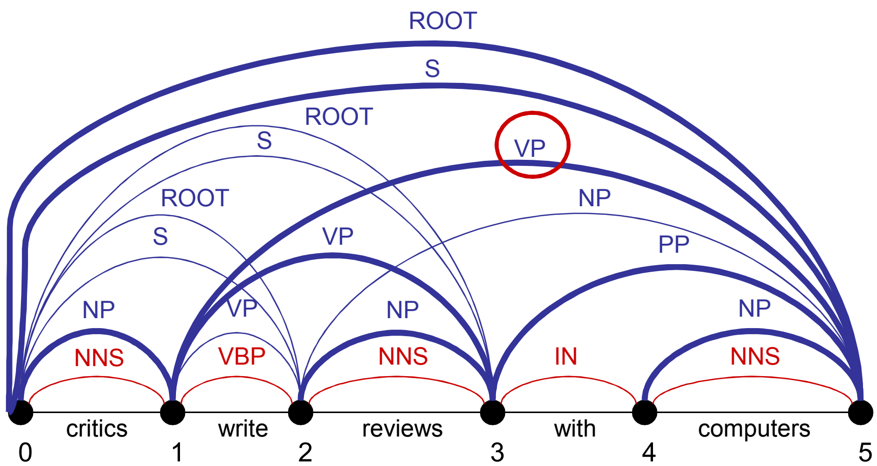
  - Enqueue / promote resulting items (if not in chart already)
  - Record backtraces as appropriate
  - Stick the popped edge in the chart (closed set)



- Queries a chart must support:
  - Is edge X:[i,j] in the chart? (What score?)
  - What edges with label Y end at position j?
  - What edges with label Z start at position i?

An Example

NNS[0,1] VBP[1,2] NNS[2,3] IN[3,4] NNS[3,4] NP[0,1] VP[1,2] NP[2,3] NP[4,5] S[0,2] VP[1,3] PP[3,5] ROOT[0,2] S[0,3] VP[1,5] NP[2,5] ROOT[0,3] S[0,5] ROOT[0,5]



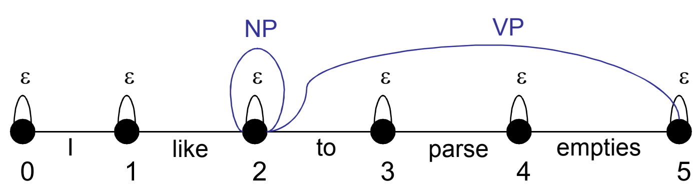
Empty Elements

- Sometimes we want to posit nodes in a parse tree that don't contain any pronounced words:

  I want you to parse this sentence

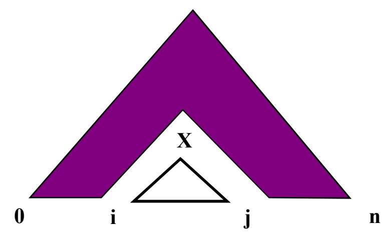
  I want [ ] to parse this sentence

- These are easy to add to a chart parser!
  - For each position i, add the "word" edge ε:[i,i]
  - Add rules like NP → ε to the grammar
  - That's it!



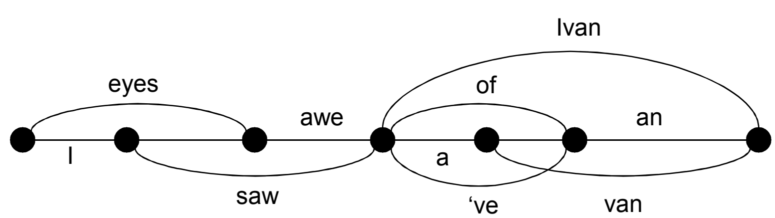
UCS / A*

- With weighted edges, order matters
  - Must expand optimal parse from bottom up (subparses first)
  - CKY does this by processing smaller spans before larger ones
  - UCS pops items off the agenda in order of decreasing Viterbi score
  - A* search also well defined
- You can also speed up the search without sacrificing optimality
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]



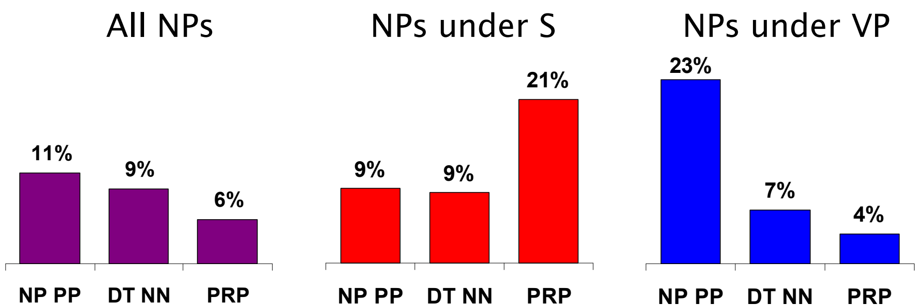
(Speech) Lattices

- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don’t know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.



Non-Independence I

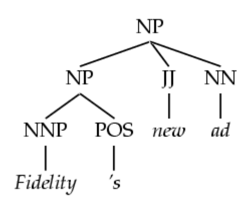
- Independence assumptions are often too strong.

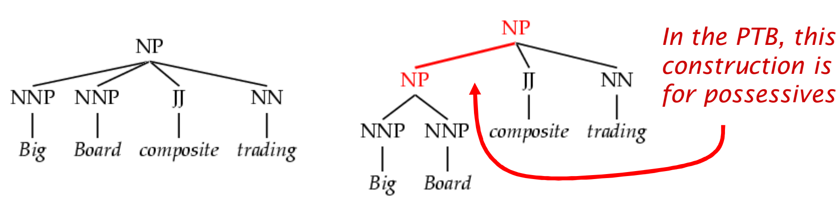


- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also: the subject and object expansions are correlated!

Non-Independence II

- Who cares?
  - NB, HMMs, all make false assumptions!
  - For generation, consequences would be obvious.
  - For parsing, does it impact accuracy?
- Symptoms of overly strong assumptions:
  - Rewrites get used where they don't belong.
  - Rewrites get used too often or too rarely.

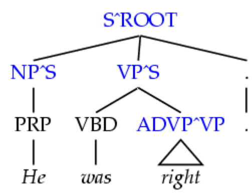




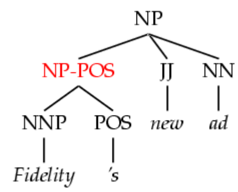
Breaking Up the Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols:

Parent annotation [Johnson 98]



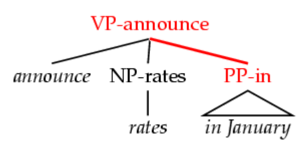
Marking possessive NPs

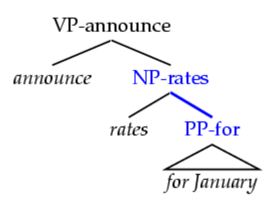


- What are the most useful "features" to encode?

Lexicalization

- Lexical heads important for certain classes of ambiguities (e.g., PP attachment):
- Lexicalizing grammar creates a much larger grammar. (cf. next week)
  - Sophisticated smoothing needed
  - Smarter parsing algorithms
  - More data needed
- How necessary is lexicalization?
  - Bilexical vs. monolexical selection
  - Closed vs. open class lexicalization





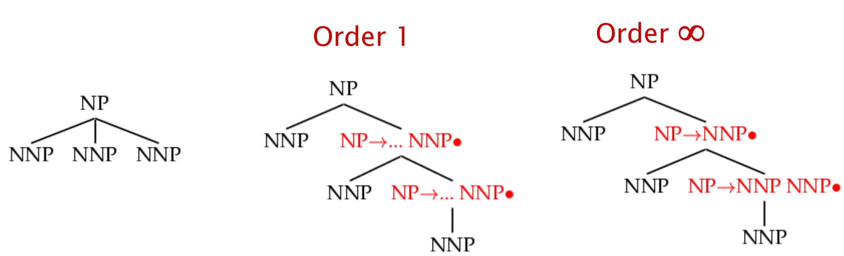
Typical Experimental Setup

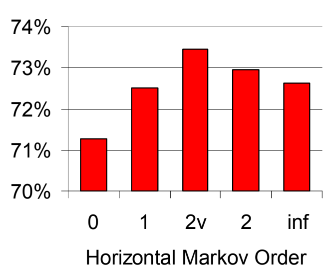
- Corpus: Penn Treebank, WSJ

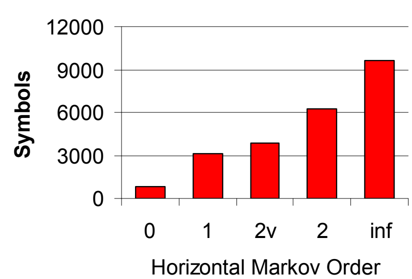
Training:	sections	02-21
Development:	section	22 (here, first 20 files)
Test:	section	23

- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Here: also size – number of symbols in grammar.
  - Passive / complete symbols: NP, NP^S
  - Active / incomplete symbols: NP → NP CC •

Horizontal Markovization



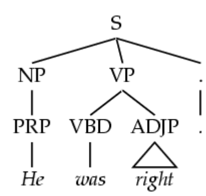




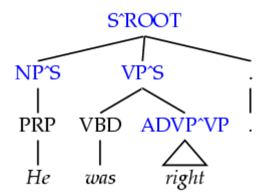
Vertical Markovization

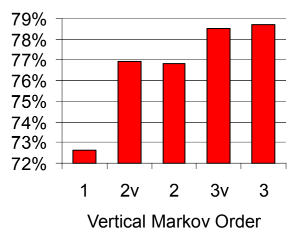
- Vertical Markov order: rewrites depend on past k ancestor nodes.
  (cf. parent annotation)

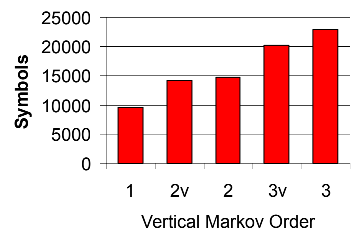
Order 1



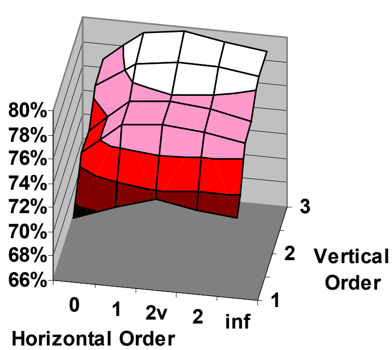
Order 2

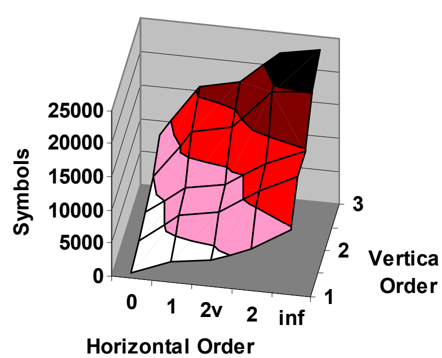






Vertical and Horizontal



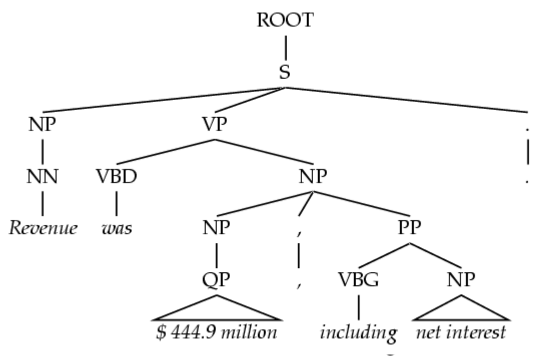


- Examples:
  - Raw treebank: v=1, h=∞
  - Johnson 98: v=2, h=∞
  - Collins 99: v=2, h=2
  - Best F1: v=3, h=2v

Model	F1	Size
Base: v=h=2v	77.8	7.5K

Unary Splits

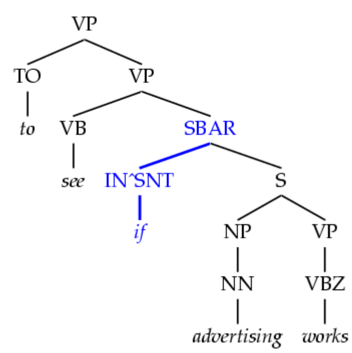
- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.
- Solution: Mark unary rewrite sites with -U



Annotation	F1	Size
Base	77.8	7.5K
UNARY	78.3	8.0K

Tag Splits

- Problem: Treebank tags are too coarse.
- Example: Sentential, PP, and other prepositions are all marked IN.
- Partial Solution:
  - Subdivide the IN tag.

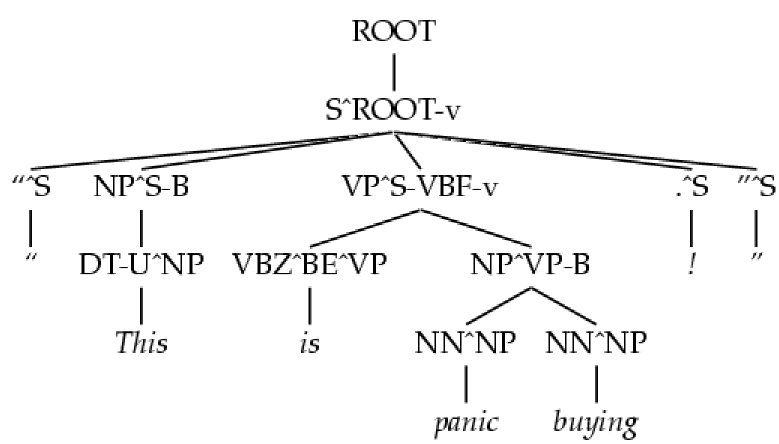


Annotation	F1	Size
Previous	78.3	8.0K
SPLIT-IN	80.3	8.1K

Other Tag Splits

	F1	Size
▪ UNARY-DT: mark demonstratives as DT^U ("the X" vs. "those")	80.4	8.1K
▪ UNARY-RB: mark phrasal adverbs as RB^U ("quickly" vs. "very")	80.5	8.1K
▪ TAG-PA: mark tags with non-canonical parents ("not" is an RB^VP)	81.2	8.5K
▪ SPLIT-AUX: mark auxiliary verbs with –AUX [cf. Charniak 97]	81.6	9.0K
▪ SPLIT-CC: separate "but" and "&" from other conjunctions	81.7	9.1K
▪ SPLIT-%: "%" gets its own tag.	81.8	9.3K

A Fully Annotated (Unlex) Tree



Some Test Set Results

Parser	LP	LR	**F1**	CB	0 CB
Magerman 95	84.9	84.6	**84.7**	1.26	56.6
Collins 96	86.3	85.8	**86.0**	1.14	59.9
Unlexicalized	86.9	85.7	**86.3**	1.10	60.3
Charniak 97	87.4	87.5	**87.4**	1.00	62.1
Collins 99	88.7	88.6	**88.6**	0.90	67.1

- Beats "first generation" lexicalized parsers.
- Lots of room to improve – more complex models next.