

Statistical NLP

Spring 2008



Lecture 15: Parsing II

Dan Klein – UC Berkeley

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]
```

Memory: Theory

- How much memory does this require?
 - Have to store the score cache
 - Cache size: $|\text{symbols}| * n^2$ doubles
 - For the plain treebank grammar:
 - $X \sim 20K$, $n = 40$, double ~ 8 bytes = $\sim 256MB$
 - Big, but workable.
- What about sparsity?

Time: Theory

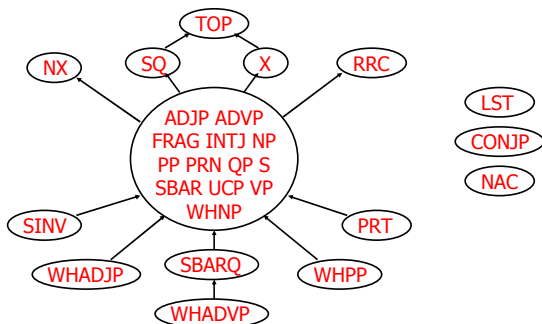
- How much time will it take to parse?
 - Have to fill each cache element (at worst)
 - Each time the cache fails, we have to:
 - Iterate over each rule $X \rightarrow YZ$ and split point k
 - Do constant work for the recursive calls
 - Total time: $|\text{rules}| * n^3$
 - Cubic time
 - Something like 5 sec for an unoptimized parse of a 20-word sentences

Unary Rules

- Unary rules?

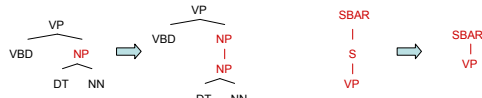
```
bestScore(X, i, j, s)
  if (j = i+1)
    return tagScore(X, s[i])
  else
    return max max score(X->YZ) *
              bestScore(Y, i, k) *
              bestScore(Z, k, j)
              max score(X->Y) *
              bestScore(Y, i, j)
```

Same-Span Reachability



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)

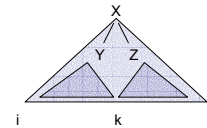
bestScoreU(X, i, j, s)
    if (j = i+1)
        return tagScore(X, s[i])
    else
        return max max score(X->Y) *
                    bestScoreB(Y, i, j)
    
```

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]
    
```



Efficient CKY

- Lots of tricks to make CKY efficient
 - Most of them are little engineering details:
 - E.g., first choose k, then enumerate through the Y:[i,k] which are non-zero, then loop through rules by left child.
 - Optimal layout of the dynamic program depends on grammar, input, even system details.
 - Another kind is more critical:
 - Many X:[i,j] can be suppressed on the basis of the input string
 - We'll see this next class as figures-of-merit or A* heuristics

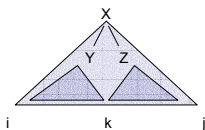
Memory: Practice

- Memory:
 - Still requires memory to hold the score table
- Pruning:
 - score[X][i][j] can get too large (when?)
 - can instead keep beams scores[i][j] which only record scores for the top K symbols found to date for the span [i,j]

Time: Theory

- How much time will it take to parse?

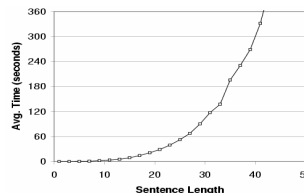
- For each diff ($\leq n$)
 - For each i ($\leq n$)
 - For each rule $X \rightarrow YZ$
 - For each split point k
 - Do constant work



- Total time: $|\text{rules}| * n^3$

Runtime: Practice

- Parsing with the vanilla treebank grammar:



~ 20K Rules
(not an optimized parser!)
Observed exponent:
3.6

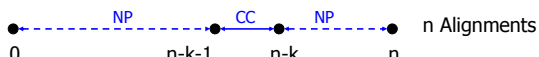
- Why's it worse in practice?
 - Longer sentences "unlock" more of the grammar
 - All kinds of systems issues don't scale

Rule State Reachability

Example: NP CC •



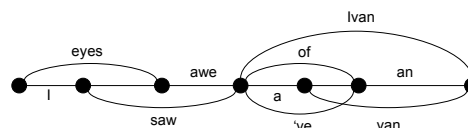
Example: NP CC NP •



- Many states are more likely to match larger spans!

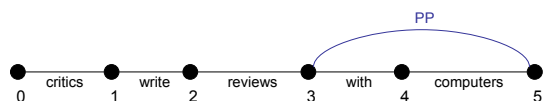
(Speech) Lattices

- 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.



A Simple Chart Parser

- Chart parsers are sparse dynamic programs
- Ingredients:
 - Nodes: positions between words
 - Edges: spans of words with labels, represent the set of trees over those words rooted at x
 - A chart: records which edges we've built
 - An agenda: a holding pen for edges (a queue)
- We're going to figure out:
 - What edges can we build?
 - All the ways we built them.



Word Edges

- An edge found for the first time is called discovered. Edges go into the agenda on discovery.
- To initialize, we discover all word edges.

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



Unary Projection

- When we pop an word edge off the agenda, we check the lexicon to see what tag edges we can build from it

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]



The "Fundamental Rule"

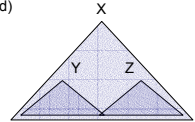
- When we pop edges off of the agenda:
 - Check for unary projections ($NNS \rightarrow critics$, $NP \rightarrow NNS$)

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

- Combine with edges already in our chart (this is sometimes called the fundamental rule)

$Y[i,j]$ and $Z[j,k]$ with $X \rightarrow YZ$ form $X[i,k]$

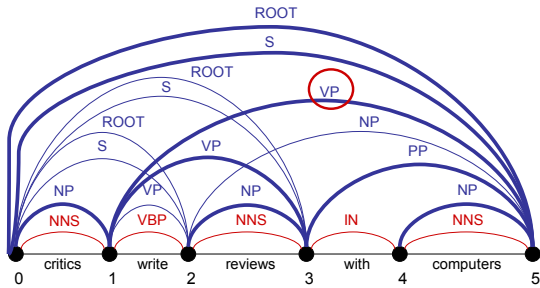
- Enqueue resulting edges (if newly discovered)
- Record backtraces (called traversals)
- Stick the popped edge in the chart



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

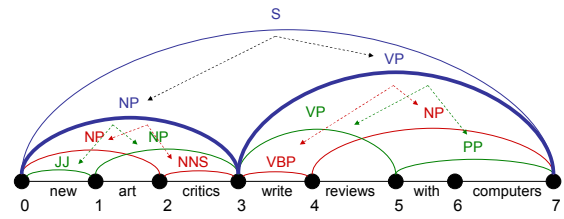
An Example

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]



Exploiting Substructure

- Each edge records all the ways it was built (locally)
 - Can recursively extract trees
 - A chart may represent too many parses to enumerate (how many?)



Order Independence

- A nice property:
 - It doesn't matter what policy we use to order the agenda (FIFO, LIFO, random).
- Why? Invariant: before popping an edge:
 - Any edge $X[i,j]$ that can be directly built from chart edges and a single grammar rule is either in the chart or in the agenda.
 - Convince yourselves this invariant holds!
- This will not be true once we get weighted parsers.

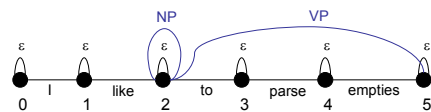
Empty Elements

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

I want John to parse this sentence

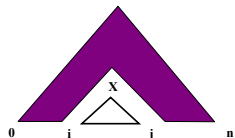
I want [] to parse this sentence

- These are easy to add to our chart parser!
 - For each position i , add the "word" edge $\epsilon[i,i]$
 - Add rules like $NP \rightarrow \epsilon$ to the grammar
 - That's it!



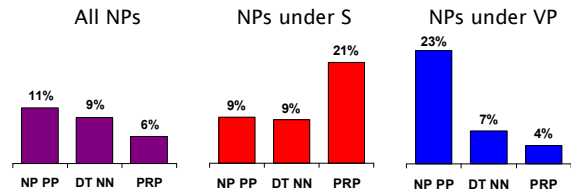
UCS / A*

- 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 optimality [Klein and Manning 03]



Non-Independence I

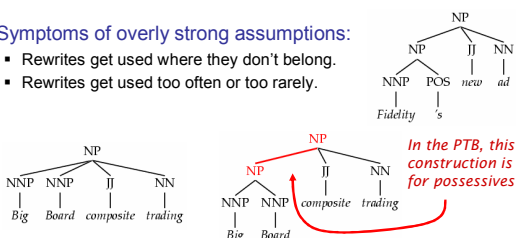
- Independence assumptions are often too strong.



- 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

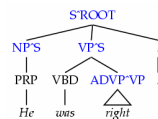
- Who cares?
 - NB, HMMs, all make false assumptions!
 - For **generation**, consequences would be obvious.
 - For **parsing**, does it impact accuracy?
- Symptoms of overly strong assumptions:
 - Rewrites get used where they don't belong.
 - Rewrites get used too often or too rarely.



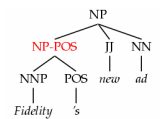
Breaking Up the Symbols

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

Parent annotation [Johnson 98]



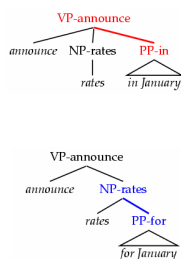
Marking possessive NPs



- 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

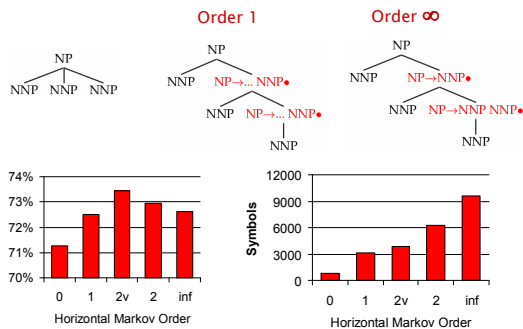
- Corpus: Penn Treebank, WSJ



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

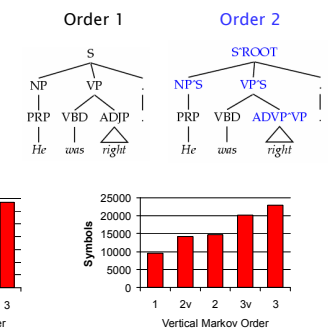
- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Here: also size – number of symbols in grammar.
 - Passive / complete symbols: NP, NP^S
 - Active / incomplete symbols: NP → NP CC •

Horizontal Markovization

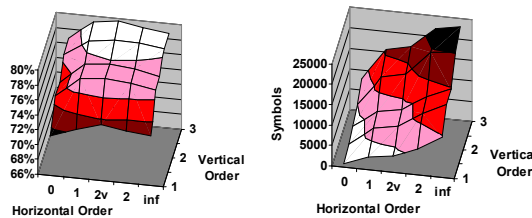


Vertical Markovization

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



Vertical and Horizontal

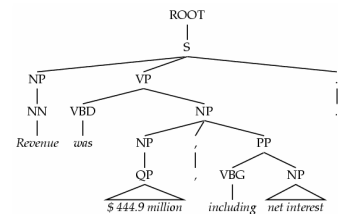


- Examples:
 - Raw treebank: $v=1, h=\infty$
 - Johnson 98: $v=2, h=\infty$
 - Collins 99: $v=2, h=2$
 - Best F1: $v=3, h=2v$

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

Unary Splits

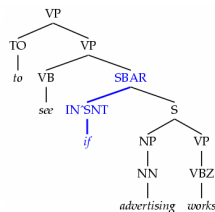
- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.
- Solution: Mark unary rewrite sites with -U



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.



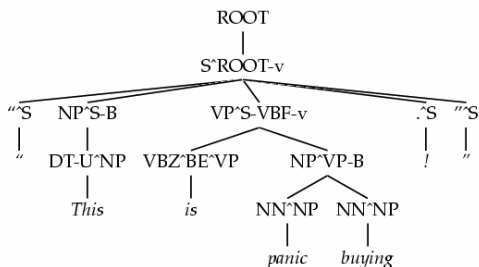
Annotation	F1	Size
Previous	78.3	8.0K
SPLIT-IN	80.3	8.1K

Other Tag Splits

- UNARY-DT: mark demonstratives as DT^A ("the X" vs. "those")
- UNARY-RB: mark phrasal adverbs as RB^A ("quickly" vs. "very")
- TAG-PA: mark tags with non-canonical parents ("not" is an RB^AVP)
- SPLIT-AUX: mark auxiliary verbs with -AUX [cf. Charniak 97]
- SPLIT-CC: separate "but" and "&" from other conjunctions
- SPLIT-%: "%" gets its own tag.

F1	Size
80.4	8.1K
80.5	8.1K
81.2	8.5K
81.6	9.0K
81.7	9.1K
81.8	9.3K

A Fully Annotated (Unlex) Tree

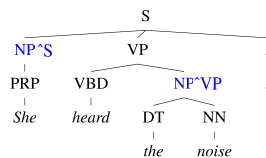


Some Test Set Results

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

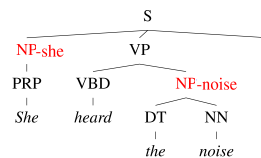
- Beats "first generation" lexicalized parsers.
- Lots of room to improve – more complex models next.

The Game of Designing a Grammar



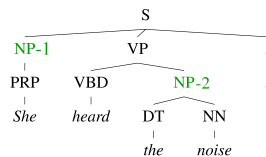
- Annotation refines base treebank symbols to improve statistical fit of the grammar
 - Parent annotation [Johnson '98]

The Game of Designing a Grammar



- Annotation refines base treebank symbols to improve statistical fit of the grammar
 - Parent annotation [Johnson '98]
 - Head lexicalization [Collins '99, Charniak '00]

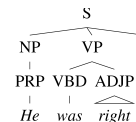
The Game of Designing a Grammar



- Annotation refines base treebank symbols to improve statistical fit of the grammar
 - Parent annotation [Johnson '98]
 - Head lexicalization [Collins '99, Charniak '00]
 - Automatic clustering?

Manual Annotation

- Manually split categories
 - NP: subject vs object
 - DT: determiners vs demonstratives
 - IN: sentential vs prepositional



- Advantages:

- Fairly compact grammar
- Linguistic motivations

- Disadvantages:

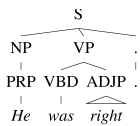
- Performance leveled out
- Manually annotated

Model	F1
Naïve Treebank Grammar	72.6
Klein & Manning '03	86.3

Automatic Annotation Induction

Advantages:

- Automatically learned: Label *all* nodes with latent variables. Same number k of subcategories for all categories.



Disadvantages:

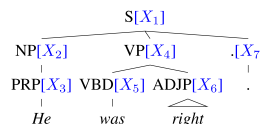
- Grammar gets too large
- Most categories are oversplit while others are undersplit.

Model	F1
Klein & Manning '03	86.3
Matsuzaki et al. '05	86.7

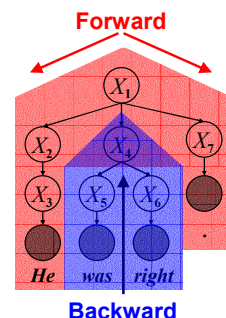
Learning Latent Annotations

EM algorithm:

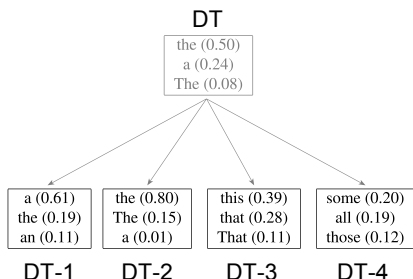
- Brackets are known
- Base categories are known
- Only induce subcategories



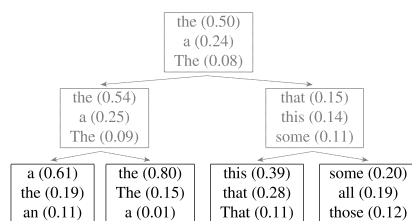
Just like Forward-Backward for HMMs.



Refinement of the DT tag

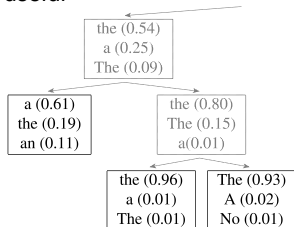


Hierarchical refinement



Adaptive Splitting

- Want to split complex categories more
- Idea: split everything, roll back splits which were least useful



Adaptive Splitting

- Evaluate loss in likelihood from removing each split =

$\frac{\text{Data likelihood with split reversed}}{\text{Data likelihood with split}}$

- No loss in accuracy when 50% of the splits are reversed.

