Problems with PCFGs?

- If we do no annotation, these trees differ only in one rule:
  - $\text{VP} \rightarrow \text{VP PP}$
  - $\text{NP} \rightarrow \text{NP PP}$
- Parse will go one way or the other, regardless of words
- We addressed this in one way with unlexicalized grammars (how?)
- Lexicalization allows us to be sensitive to specific words

Lexicalized Trees

- Add "headwords" to each phrasal node
- Syntactic vs. semantic heads
- Headship not in (most) treebanks
- Usually use head rules, e.g.:
  - $\text{NP}$: Take leftmost NP, Take rightmost JJ, Take right child
  - $\text{VP}$: Take leftmost VB, Take left child

Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like $\text{NP(saw)} \rightarrow \text{NP(saw)} \text{NP} \rightarrow \text{Other} \text{NP(today)}$
- Never going to get these atomically off of a treebank
- Solution: break up derivation into smaller steps
Derivational Representations

- Generative derivational models:
  \[ P(D) = \prod_{d \in D} P(d_1 d_2 \ldots d_{n-1}) \]
- How is a PCFG a generative derivational model?
- Distinction between parses and parse derivations.
  \[ P(T) = \sum_{D \in D(T)} P(D) \]
- How could there be multiple derivations?

Lexical Derivation Steps

- Simple derivation of a local tree [simplified Charniak 97]
  \[ P(\text{VP} \rightarrow \text{VBD} \cdot \text{VP}) \rightarrow \text{VP} \]
  \[ P(\text{VP} \rightarrow \text{VBD} \cdot \text{NP} \cdot \text{PP}) \rightarrow \text{VP} \]
  \[ P(\text{NP} \rightarrow \text{NP} \cdot \text{PP}) \rightarrow \text{NP} \]
  \[ P(\text{PP} \rightarrow \text{NP} \cdot \text{PP}) \rightarrow \text{PP} \]

Lexical Derivation Steps

- Another derivation of a local tree [Collins 99]
  1. Choose a head tag and word
  2. Choose a complement bag
  3. Generate children (incl. adjuncts)
  4. Recursively derive children

Naïve Lexicalized Parsing

- Can, in principle, use CKY on lexicalized PCFGs
  - \( O(Rn^3) \) time and \( O(Sn^2) \) memory
  - But \( R = n^2 \) and \( S = n^2 \)
  - Result is completely impractical (why?)
  - Memory: 10K rules * 50K words * (40 words)^2 * 8 bytes = 6TB
- Can modify CKY to exploit lexical sparsity
  - Lexicalized symbols are a base grammar symbol and a pointer into the input sentence, not any arbitrary word
  - Result: \( O(n^5) \) time, \( O(n^3) \)
  - Memory: 10K rules * (40 words)^3 * 8 bytes = 5GB

Lexicalized CKY

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

Quartic Parsing

- Turns out, you can do better [Eisner 99]
  \[ P(\text{VBD} \cdot \text{NP} \cdot \text{PP}) \rightarrow \text{VP} \]
  \[ P(\text{NP} \rightarrow \text{NP} \cdot \text{PP}) \rightarrow \text{NP} \]
  \[ P(\text{PP} \rightarrow \text{NP} \cdot \text{PP}) \rightarrow \text{PP} \]
- Gives an \( O(n^4) \) algorithm
- Still prohibitive in practice if not pruned
Dependency Parsing

- Lexicalized parsers can be seen as producing dependency trees
- Each local binary tree corresponds to an attachment in the dependency graph

Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the O(n^3) CKY
  - Remember only a few hypotheses for each span \(<i,j>\)
  - If we keep \(K\) hypotheses at each span, then we do at most \(O(nK^2)\) work per span (why?)
  - Keeps things more or less cubic
  - Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)

Pruning with A*

- You can also speed up the search without sacrificing optimality
- For agenda based parsers:
  - 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]
### A* Speedup

- Total time dominated by calculation of A* tables in each projection... $O(n^3)$

### Some Results

- **Lexicalized parsers**
  - Collins 99 – 88.6 F1
  - Charniak 00 – 90.1 F1

- **However**
  - Bilexical counts rarely make a difference (why?)
  - Gildea 01 – Removing bilexical counts costs < 0.5 F1

- **Bilexical vs. monolexical vs. smart smoothing**

### Parse Reranking

- **Assume the number of parses is very small**
- We can represent each parse $T$ as an arbitrary feature vector $\phi(T)$
  - Typically, all local rules are features
  - Also non-local features, like how right-branching the overall tree is
  - [Charniak and Johnson 05] gives a rich set of features

- Since the number of parses is no longer huge
  - Can enumerate all parses efficiently
  - Can use simple machine learning methods to score trees
  - E.g. maxent reranking: learn a binary classifier over trees where:
    - The top candidates are positive
    - All others are negative
    - Rank trees by $P(+|T)$

- The best parsing numbers are from reranking systems
  - Collins 05: 90.3
  - Charniak and Johnson 05: 91.0 (!)

### Shift-Reduce Parsers

- **Another way to derive a tree:**

- **Parsing**
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]

### Data-oriented parsing:

- **Rewrite large (possibly lexicalized) subtrees in a single step**

- Formally, a tree-insertion grammar
- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable parse is NP-complete
TIG: Insertion

- Start with local trees
- Can insert structure with adjunction operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- ... as well as other weird stuff that CFGs don't capture well (e.g., cross-serial dependencies)

Tree-adjointing grammars

TAG: Adjunction

- N
  - Adj
  - N
  - N
  - Adj

TAG: Long Distance

CCG Parsing

- Combinatory Categorial Grammar
  - Fully (mono-) lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus (more later)
  - Can have spurious ambiguities (why?)

Digression: Is NL a CFG?

- Cross-serial dependencies in Dutch

... dit: Wie Jan de kinderen dag helpt boven mensen... telt elken...