Problems with PCFGs?

- If we do no annotation, these trees differ only in one rule:
  - VP → VP PP
  - NP → 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
Problems with PCFGs

- What’s different between basic PCFG scores here?
- What (lexical) correlations need to be scored?

Problems with PCFGs

- Another example of PCFG indifference
  - Left structure far more common
  - How to model this?
  - Really structural: “chicken with potatoes with gravy”
  - Lexical parsers model this effect, but not by virtue of being lexical
Lexicalized Trees

- Add “headwords” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually use head rules, e.g.:
    - NP:
      - Take leftmost NP
      - Take rightmost N*
      - Take rightmost JJ
      - Take right child
    - VP:
      - Take leftmost VB*
      - Take leftmost VP
      - Take left child

Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like

\[
\text{VP(saw)} \rightarrow \text{VBD(saw)} \text{ NP-C(her) NP(today)}
\]

- Never going to get these atomically off of a treebank

- Solution: break up derivation into smaller steps
Lexical Derivation Steps

- Simple derivation of a local tree [simplified Charniak 97]

```
VP[saw]
  VBD[saw] NP[her] NP[today] PP[on]
  (VP->VBD...PP *)[saw]
  (VP->VBD...NP *)[saw]
  (VP->VBD...NP *)[saw]
  (VP->VBD...NP *)[saw]
  VBD[saw]
```

Still have to smooth with mono- and non-lexical backoffs

\[
P(\text{STOP}|VBD[saw], \text{VP}, \text{PP})
\]

\[
P(\text{PP}[on]|VBD[saw], \text{VP}, \text{NP})
\]

\[
P(\text{NP}[today]|VBD[saw], \text{VP}, \text{NP})
\]

\[
P(\text{NP}[her]|VBD[saw], \text{VP}, \text{START})
\]

\[
P(VBD[saw]|VP[saw])
\]

Lexical Derivation Steps

- Another derivation of a local tree [Collins 99]

Choose a head tag and word

Choose a complement bag

Generate children (incl. adjuncts)

Recursively derive children
Naïve Lexicalized Parsing

- Can, in principle, use CKY on lexicalized PCFGs
  - $O(Rn^3)$ time and $O(Sn^2)$ memory
  - But $R = rV^2$ and $S = sV$
  - Result is completely impractical (why?)
- Memory: 10K rules * 50K words * (40 words)$^2$ * 8 bytes $\approx$ 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: $O(rn^5)$ time, $O(sn^3)$
  - Memory: 10K rules * (40 words)$^3$ * 8 bytes $\approx$ 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')
        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]
- 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
Dependency Parsing

- Pure dependency parsing is only cubic [Eisner 99]

Some work on non-projective dependencies
- Common in, e.g. Czech parsing
- Can do with MST algorithms [McDonald and Pereira 05]

Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the $O(n^5)$ 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 PCFG

- The Charniak parser prunes using a two-pass approach [Charniak 97+]
  - First, parse with the base grammar
  - For each $X: [i,j]$ calculate $P(X|i,j,s)$
    - This isn’t trivial, and there are clever speed ups
  - Second, do the full $O(n^5)$ CKY
    - Skip any $X: [i,j]$ which had low (say, < 0.0001) posterior
    - Avoids almost all work in the second phase!
    - Currently the fastest lexicalized parser

- Charniak et al 06: can use more passes
- Petrov et al 07: can use many more passes

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]
Projection-Based A*

\[ \pi_{\text{SYNTACTIC}} \]

Factory payrolls fell in Sept.

\[ \pi_{\text{SEMANTIC}} \]

A* Speedup

- **Total time dominated by calculation of A* tables in each projection…** \( O(n^3) \)
Results

- Some results
  - Collins 99 – 88.6 F1 (generative lexical)
  - Charniak and Johnson 05 – 89.7 / 91.3 F1 (generative lexical / reranked)
  - Petrov et al 06 – 90.7 F1 (generative unlexical)
  - McClosky et al 06 – 92.1 F1 (gen + rerank + self-train)

- 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 $\varphi(T)$
  - 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
Parse Reranking

- 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

Shift-Reduce Parsers

- Another way to derive a tree:

  ![Tree Diagram]

- 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
Derivational Representations

- Generative derivational models:
  \[ P(D) = \prod_{d_i \in D} P(d_i | d_0 \ldots d_{i-1}) \]

- How is a PCFG a generative derivational model?

- Distinction between parses and parse derivations.
  \[ P(T) = \sum_{P(D) \rightarrow T} P(D) \]

- How could there be multiple derivations?

Tree-adjoining grammars

- Start with local trees
- Can insert structure with \textit{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)
TAG: Adjunction

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?)

```
John ⊨ NP
shares ⊨ NP
buys ⊨ (S\NP)/NP
sleeps ⊨ S\NP
well ⊨ (S\NP)\(S\NP)
```

```
S
  NP    S\NP
  John (S\NP)/NP  NP
       buys  shares
```