**Classical NLP: Parsing**

- **Write symbolic or logical rules:**
  - Grammar (CFG)
  - Lexicon
  - ROOT $\rightarrow$ S
  - NP $\rightarrow$ NP PP
  - NN $\rightarrow$ interest
  - S $\rightarrow$ NP VP
  - VP $\rightarrow$ VBP NP
  - NNS $\rightarrow$ raises
  - NP $\rightarrow$ DT NN
  - VP $\rightarrow$ VBP NP PP
  - VBZ $\rightarrow$ raises
  - NP $\rightarrow$ NN NNS
  - PP $\rightarrow$ IN NP
  - VBP $\rightarrow$ interest
  - VBZ $\rightarrow$ 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.
  - $T$: the set of terminals (the words)
  - $S$: the start symbol
    - Not usually the sentence non-terminal S
    - Often written as ROOT or TOP
  - $R$: 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, $VP \rightarrow$ VBP CC VP
    - Also called rewrites, productions, or local trees

- A PCFG adds:
  - A top-down production probability per rule $P(Y_1 Y_2 \ldots Y_k | X)$

---

**Treebank Sentences**

```
( (S (NP SB) 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 SB)
  (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):

```
<table>
<thead>
<tr>
<th>Production</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT $\rightarrow$ S</td>
<td>1</td>
</tr>
<tr>
<td>S $\rightarrow$ NP VP</td>
<td>1</td>
</tr>
<tr>
<td>NP $\rightarrow$ PRP</td>
<td>1</td>
</tr>
<tr>
<td>VP $\rightarrow$ VBD ADJP</td>
<td>1</td>
</tr>
</tbody>
</table>
| ...
```

- 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

- $\text{bestScore}(X, i, j, s)$
  - if $j = i+1$
    - return $\text{tagScore}(X, s[i])$
  - else
    - return $\max \{ \text{score}(X \rightarrow Y Z) \ast \text{bestScore}(Y, i, k) \ast \text{bestScore}(Z, k, j) \}$

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

A Memoized Parser

- One small change:

```python
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 \rightarrow Y Z) \ast 
      bestScore(Y, i, k) \ast 
      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

```python
bestScore(s)
for (k : [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 \rightarrow Y Z : rule)
      for (k : [i+1, j-1])
        score[X][i][j] = max \{ score[X][i][j], 
          score[X \rightarrow Y Z] \ast 
          score[Y][i][k] \ast 
          score[Z][k][j] \}
```

Unary Rules

- Unary rules?

```python
bestScore(X, i, j, s)
if (j = i+1)
  return tagScore(X, s[i])
else
  return max \{ score(X \rightarrow Y Z) \ast 
    bestScore(Y, i, k) \ast 
    bestScore(Z, k, j) \}
  \max score(X \rightarrow Y) \ast 
  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

```plaintext
bestScore\(X, i, j, s\)
- return \(\max_{k} \text{score}(X \rightarrow YZ) \times \max_{l} \text{bestScore}(Y, i, l) \times \text{bestScore}(Z, k, j)\)
```

#### 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: \(|\text{rules}|n^3\)
  - Something like 5 sec for an unoptimized parse of a 20-word sentences

### Memory

- **How much memory does this require?**
  - Have to store the score cache
  - Cache size: \(|\text{symbols}|n^3\) doubles
- **For the plain treebank grammar:**
  - \(X \sim 20k, n = 40\), double \(\sim 8 \text{ bytes} = \sim 266 \text{ MB}\)
  - Big, but workable.

- **Pruning: Beams**
  - \(\text{score}(X[i][i])\) can get too large (why?)
  - Can keep beams (truncated maps \(\text{score}(X[i][i])\)) 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: 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

- Example: NP CC
  - 1 Alignment

### Rule State Reachability

- 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).

**Unary Projection**

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

**Item Successors**

- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)
    - Y[i,j] with X → Y forms X[i,j]
  - Hypergraph successors: combine with items already in our chart
    - Y[i,j] and Z[i,k] with X → Y Z form X[i,k]

**An Example**

**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 optimality (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.

Treebank PCFGs [Charniak 96]

- Use PCFGs 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
  …. 
```

**Model** | **F1**
--- | ---
Baseline | 72.0

Conditional Independence?

- Not every NP expansion can fill every NP slot
- A grammar with symbols like “NP” won’t be context-free
- Statistically, conditional independence too strong

Non-Independence

- Independence assumptions are often too strong.

```
<table>
<thead>
<tr>
<th></th>
<th>All NPs</th>
<th>NPs under S</th>
<th>NPs under VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP PP</td>
<td>11%</td>
<td>9%</td>
<td>6%</td>
</tr>
<tr>
<td>DT NN</td>
<td>9%</td>
<td>9%</td>
<td>21%</td>
</tr>
<tr>
<td>PRP</td>
<td>6%</td>
<td>7%</td>
<td>4%</td>
</tr>
</tbody>
</table>
```

- 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!

Grammar Refinement

- Example: PP attachment

```
They raised a point of order
```
Grammar Refinement

- Structure Annotation [Johnson '98, Klein&Manning '03]
- Lexicalization [Collins '99, Charniak '00]
- Latent Variables [Matsuzaki et al. 05, Petrov et al. '06]

Accuracy – F1: harmonic mean of per-node labeled precision and recall.
Here: also size – number of symbols in grammar.

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

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

Vertical Markovization

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

Vertical and Horizontal

- Examples:
  - Model: F1, Size
  - Base: v=hm=2v
    - 77.8, 7.9K
Unary Splits

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

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>

Tag Splits

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

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>

Other Tag Splits

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

<table>
<thead>
<tr>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>

A Fully Annotated (Unlex) Tree

Some Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
<th>CB</th>
<th>0 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Unlexicalized</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
<td>0.90</td>
<td>67.1</td>
</tr>
</tbody>
</table>

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