Classical NLP: Parsing

- Write symbolic or logical rules:

<table>
<thead>
<tr>
<th>Grammar (CFG)</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT → S</td>
<td>NP → NP PP</td>
</tr>
<tr>
<td>S → NP VP</td>
<td>VP → VBZ NP</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>VP → VBZ NP</td>
</tr>
<tr>
<td>NP → NN NN</td>
<td>PP → IN NP</td>
</tr>
</tbody>
</table>

- 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)
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - 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 → Y₁ Y₂ ... Yₖ with X, Yi ∈ N
    - Also called rewrites, productions, or local trees

- A PCFG adds:
  - A top-down production probability per rule P(Y₁, Y₂ ... Yₖ | X)

Treebank Sentences

```plaintext
( 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):

```
NP  S  1
   VP  1
  PRP  PRP  1
     VP  VBZ 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-ary is easy.
  - Reconstructing unaries is trickier.
  - The straightforward transformations don't preserve tree scores
  - Makes parsing algorithms simpler!

A Memoized Parser

- One small change:

  ```
  bestScore(X, i, j, s)
  if (scores[i][j][s] == 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[i][j][s] = score
  return scores[i][j][s]
  ```

A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

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

Unary Rules

- Unary rules?

  ```
  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)
  ```

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

  ```
  VP
  NP        VP
  VBD NP    NP
  DT NN     IP
  VP        IP
  ```

- Alternate unary and binary layers
- Reconstruct unary chains afterwards
Alternating Layers

bestScore(X,i,j,s) =
  return max max score(X→YZ) *
    bestScore(Y,i,k) *
    bestScore(U,k,j)

bestScore(X,i,j,s)
  if (j = i+1)
    return tagScore(X,s[i])
  else
    return max max score(X→Y) *
      bestScore(Y,i,j)

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 •

Example: NP CC NP •

- 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)

![Diagram showing agenda items and edges](image)

**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

![Diagram showing unary projection](image)

**Item Successors**

- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)
  - Hypergraph successors: combine with items already in our chart

![Diagram showing item successors](image)

**An Example**

![Diagram showing an example parse tree](image)

**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)
- OK: doing 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.

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

---

**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, NPS
  - Active / incomplete symbols: NP → NP CC

Horizontal Markovization

- Horizontal Markovization
- Order 1
- Order \( \infty \)
- Symbols
- Order 1
- Order 2
- Vertical Markovization
- Vertical Markov order: rewrites depend on past \( k \) ancestor nodes.
  - (cf. parent annotation)
- Symbols
- Vertical Order
- Horizontal Order

Vertical and Horizontal

- Examples:
  - Raw treebank: \( v = 1, h = \infty \)
  - Johnson 96: \( v = 2, h = \infty \)
  - 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

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
- Base 77.8 7.5K
- UNARY 78.3 8.0K
**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>

**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><strong>84.7</strong></td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td><strong>86.0</strong></td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Unlexicalized</td>
<td>86.9</td>
<td>85.7</td>
<td><strong>86.3</strong></td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td><strong>87.4</strong></td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td><strong>88.6</strong></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.