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 → VBP NP</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>VP → VBP NP PP</td>
</tr>
<tr>
<td>NP → NN NNS</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)
  - \(S\): the start symbol
    - Often written as ROOT or TOP
    - Not usually the sentence non-terminal S
  - \(R\): the set of rules
    - Of the form \(X \rightarrow Y_1 \ldots Y_k\), with \(X, Y_i \in N\)
    - Examples: \(S \rightarrow NP \ VB\), \(VP \rightarrow VP \ CC\ VP\)
    - Also called rewrites, productions, or local trees

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

---

Treebank Sentences

```
(S (NP-SBJ 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-SBJ *)
  (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):

```
ROOT
  | S
  |   NP VP
  |       | PRP VBD ADJP
  |       | He was JJ
  |       | right

ROOT → S 1
S → NP VP . 1
NP → PRP 1
VP → VBD 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-aries is easy
  - Reconstructing unaries is trickier
  - The straightforward transformations don’t preserve tree scores
- Makes parsing algorithms simpler!

A Recursive Parser

```python
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:

```java
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]
```

A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

```java
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]
```
Unary Rules

- Unary rules?

```plaintext
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)
```

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

\[
\text{bestScoreB}(X, i, j, s) \\
\quad \text{return max} \quad \max \text{ score}(X \rightarrow YZ) \quad \ast \\
\quad \quad \text{bestScoreU}(Y, i, k) \quad \ast \\
\quad \quad \text{bestScoreU}(Z, k, j)
\]

\[
\text{bestScoreU}(X, i, j, s) \\
\quad \text{if} \quad (j = i+1) \\
\quad \quad \text{return} \quad \text{tagScore}(X, s[i]) \\
\quad \text{else} \\
\quad \quad \text{return max} \quad \max \text{ score}(X \rightarrow Y) \quad \ast \\
\quad \quad \quad \quad \text{bestScoreB}(Y, i, j)
\]

Memory

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

- Pruning: Beams
  - score[X][i][j] can get too large (when?)
  - Can keep beams (truncated maps score[i][j]) 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: 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)

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).

AGENDA

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

CHART [EMPTY]
Unary Projection

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

<table>
<thead>
<tr>
<th>critics[0,1]</th>
<th>write[1,2]</th>
<th>reviews[2,3]</th>
<th>with[3,4]</th>
<th>computers[4,5]</th>
</tr>
</thead>
</table>

Item Successors

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

- Enqueue / promote resulting items (if not in chart already)
- Record backtraces as appropriate
- Stick the popped edge in the chart (closed set)

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

Graph successors: unary projections (NNS → critics, NP → NNS)
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 \( \varepsilon \)\([i,i]\)
  - Add rules like \( NP \rightarrow \varepsilon \) 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.
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.

In the PTB, this construction is for possessives.
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

- Order 1
- Order ∞
Vertical Markovization

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

![Order 1](Order1.png)
![Order 2](Order2.png)

### Vertical and Horizontal

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

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base: $v=h=2v$</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
</tbody>
</table>
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.