A Recursive Parser

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

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

- One small change:

```java
def 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]
```

Memory: Theory

- 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.

- What about sparsity?
Time: Theory

- How much time will it take to parse?
  - Have to fill each cache element (at worst)
  - Each time the cache fails, we have to:
    - Iterate over each rule \( X \rightarrow Y Z \) and split point \( k \)
    - Do constant work for the recursive calls
  - Total time: \(|\text{rules}| \cdot n^3\)
  - Cubic time
  - Something like 5 sec for an unoptimized parse of a 20-word sentences

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\rightarrow Y Z) *
              bestScore(Y, i, k) *
              bestScore(Z, k, j)
              max score(X\rightarrow Y) *
              bestScore(Y, i, j)
```
Same-Span Reachability

- ADJP ADVP
- FRAG INTJ NP
- PP PRN QP S
- SBAR UCP VP WHNP

- TOP
- NX
- SQ
- X
- RRC
- LST
- CONJP
- NAC
- SINV
- WHADJP
- SBARQ
- WHPP
- WHADVP

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

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

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

A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

\[
\text{bestScore}(s) \\
\quad \text{for } (i : [0,n-1]) \\
\quad \quad \text{for } (X : \text{tags}[s[i]]) \\
\quad \quad \quad \text{score}[X][i][i+1] = \\
\quad \quad \quad \text{tagScore}(X,s[i]) \\
\quad \text{for } (\text{diff} : [2,n]) \\
\quad \quad \text{for } (i : [0,n-\text{diff}]) \\
\quad \quad \quad j = i + \text{diff} \\
\quad \quad \quad \text{for } (X\rightarrow YZ : \text{rule}) \\
\quad \quad \quad \quad \text{for } (k : [i+1, j-1]) \\
\quad \quad \quad \quad \quad \text{score}[X][i][j] = \max \text{ score}[X][i][j], \\
\quad \quad \quad \quad \quad \text{score}(X\rightarrow YZ) * \\
\quad \quad \quad \quad \quad \text{score}[Y][i][k] * \\
\quad \quad \quad \quad \quad \text{score}[Z][k][j]
\]
Efficient CKY

- Lots of tricks to make CKY efficient
  - Most of them are little engineering details:
    - E.g., first choose k, then enumerate through the Y:\[i,k\] which are non-zero, then loop through rules by left child.
    - Optimal layout of the dynamic program depends on grammar, input, even system details.
  - Another kind is more critical:
    - Many X:\[i,j\] can be suppressed on the basis of the input string
    - We'll see this next class as figures-of-merit or A* heuristics

Memory: Practice

- Memory:
  - Still requires memory to hold the score table

- Pruning:
  - score[X][i][j] can get too large (when?)
  - can instead keep beams scores[i][j] which only record scores for the top K symbols found to date for the span [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|*n³

Runtime: 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
Rule State Reachability

Example: NP CC •

```
0          n-1          n
```

1 Alignment

Example: NP CC NP •

```
0          n-k-1        n-k        n
```

n Alignments

- Many states are more likely to match larger spans!

(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.
A Simple Chart Parser

- Chart parsers are sparse dynamic programs
- Ingredients:
  - Nodes: positions between words
  - Edges: spans of words with labels, represent the set of trees over those words rooted at x
  - A chart: records which edges we’ve built
  - An agenda: a holding pen for edges (a queue)
- We’re going to figure out:
  - What edges can we build?
  - All the ways we built them.

 critics write reviews with computers

0 1 2 3 4 5

AGENDA

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

CHART [EMPTY]
Unary Projection

- When we pop an word edge off the agenda, we check the lexicon to see what tag edges we can build from it

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

```
critics   write   reviews   with   computers
```

The “Fundamental Rule”

- When we pop edges off of the agenda:
  - Check for unary projections (NNS → critics, NP → NNS)

  \[ Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j] \]

  - Combine with edges already in our chart (this is sometimes called the fundamental rule)

  \[ Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y \ Z \text{ form } X[i,k] \]

  - Enqueue resulting edges (if newly discovered)
  - Record backtraces (called traversals)
  - Stick the popped edge in the chart

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


Exploiting Substructure

- Each edge records all the ways it was built (locally)
  - Can recursively extract trees
  - A chart may represent too many parses to enumerate (how many?)
Order Independence

- A nice property:
  - It doesn't matter what policy we use to order the agenda (FIFO, LIFO, random).

- Why? Invariant: before popping an edge:
  - Any edge $X[i,j]$ that can be directly built from chart edges and a single grammar rule is either in the chart or in the agenda.
  - Convince yourselves this invariant holds!

- This will not be true once we get weighted parsers.

Empty Elements

- Sometimes we want to posit nodes in a parse tree that don’t contain any pronounced words:

  I want John to parse this sentence
  I want [ ] to parse this sentence

  These are easy to add to our chart parser!
  - For each position $i$, add the “word” edge $\varepsilon:[i,i]$
  - Add rules like $\text{NP} \to \varepsilon$ 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]

Non-Independence I

- 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></td>
<td>NP PP</td>
<td>DT NN</td>
<td>PRP</td>
</tr>
<tr>
<td>11%</td>
<td>9%</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>9%</td>
<td>9%</td>
<td>21%</td>
<td></td>
</tr>
<tr>
<td>9%</td>
<td>9%</td>
<td>23%</td>
<td>7%</td>
</tr>
<tr>
<td>9%</td>
<td>9%</td>
<td>21%</td>
<td>7%</td>
</tr>
<tr>
<td>21%</td>
<td></td>
<td></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, 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)
**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.
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson '98]

- Head lexicalization [Collins '99, Charniak '00]
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson '98]
  - Head lexicalization [Collins '99, Charniak '00]
  - Automatic clustering?

Manual Annotation

- Manually split categories
  - NP: subject vs object
  - DT: determiners vs demonstratives
  - IN: sentential vs prepositional

Advantages:
- Fairly compact grammar
- Linguistic motivations

Disadvantages:
- Performance leveled out
- Manually annotated

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Treebank Grammar</td>
<td>72.6</td>
</tr>
<tr>
<td>Klein &amp; Manning '03</td>
<td>86.3</td>
</tr>
</tbody>
</table>
Automatic Annotation Induction

- **Advantages:**
  - Automatically learned:
    - Label all nodes with latent variables.
    - Same number $k$ of subcategories for all categories.
  - Disadvantages:
    - Grammar gets too large
    - Most categories are oversplit while others are undersplit.

<table>
<thead>
<tr>
<th>Model</th>
<th>$F1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klein &amp; Manning ’03</td>
<td>86.3</td>
</tr>
<tr>
<td>Matsuzaki et al. ’05</td>
<td>86.7</td>
</tr>
</tbody>
</table>

Learning Latent Annotations

EM algorithm:
- Brackets are known
- Base categories are known
- Only induce subcategories

Just like Forward-Backward for HMMs.

Backward
Refinement of the DT tag

Hierarchical refinement
Adaptive Splitting

- Want to split complex categories more
- Idea: split everything, roll back splits which were least useful

Adaptive Splitting

- Evaluate loss in likelihood from removing each split =
  \[
  \frac{\text{Data likelihood with split reversed}}{\text{Data likelihood with split}}
  \]
- No loss in accuracy when 50% of the splits are reversed.
Adaptive Splitting Results

- 50% Merging
- Hierarchical Training
- Flat Training

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>88.4</td>
</tr>
<tr>
<td>With 50% Merging</td>
<td>89.5</td>
</tr>
</tbody>
</table>

Number of Phrasal Subcategories

- NP, VP, ADJP, SBARQ, X, ROOT, LST, etc.
Number of Lexical Subcategories

Final Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>F1 ≤ 40 words</th>
<th>F1 all words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klein &amp; Manning ’03</td>
<td>86.3</td>
<td>85.7</td>
</tr>
<tr>
<td>Matsuzaki et al. ’05</td>
<td>86.7</td>
<td>86.1</td>
</tr>
<tr>
<td>Collins ’99</td>
<td>88.6</td>
<td>88.2</td>
</tr>
<tr>
<td>Charniak &amp; Johnson ’05</td>
<td>90.1</td>
<td>89.6</td>
</tr>
<tr>
<td>Petrov et. al. 06</td>
<td>90.2</td>
<td>89.7</td>
</tr>
</tbody>
</table>
Learned Splits

- **Proper Nouns (NNP):**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP-12</td>
<td>John</td>
<td>Robert</td>
<td>James</td>
</tr>
<tr>
<td>NNP-2</td>
<td>J.</td>
<td>E.</td>
<td>L.</td>
</tr>
<tr>
<td>NNP-1</td>
<td>Bush</td>
<td>Noriega</td>
<td>Peters</td>
</tr>
<tr>
<td>NNP-15</td>
<td>New</td>
<td>San</td>
<td>Wall</td>
</tr>
<tr>
<td>NNP-3</td>
<td>York</td>
<td>Francisco</td>
<td>Street</td>
</tr>
</tbody>
</table>

- **Personal pronouns (PRP):**

<table>
<thead>
<tr>
<th>PRP-0</th>
<th>it</th>
<th>He</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP-1</td>
<td>it</td>
<td>he</td>
<td>they</td>
</tr>
<tr>
<td>PRP-2</td>
<td>it</td>
<td>them</td>
<td>him</td>
</tr>
</tbody>
</table>

- **Relative adverbs (RBR):**

<table>
<thead>
<tr>
<th>RBR-0</th>
<th>further</th>
<th>lower</th>
<th>higher</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBR-1</td>
<td>more</td>
<td>less</td>
<td>More</td>
</tr>
<tr>
<td>RBR-2</td>
<td>earlier</td>
<td>Earlier</td>
<td>later</td>
</tr>
</tbody>
</table>

- **Cardinal Numbers (CD):**

<table>
<thead>
<tr>
<th>CD-7</th>
<th>one</th>
<th>two</th>
<th>Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-4</td>
<td>1989</td>
<td>1990</td>
<td>1988</td>
</tr>
<tr>
<td>CD-11</td>
<td>million</td>
<td>billion</td>
<td>trillion</td>
</tr>
<tr>
<td>CD-0</td>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>CD-3</td>
<td>1</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>CD-9</td>
<td>78</td>
<td>58</td>
<td>34</td>
</tr>
</tbody>
</table>