Roadmap

- Probabilistic Parsing:
  - PCFG issues
  - Modeling improvements on PCFGs
    - Parent annotation
    - Lexicalization
    - Markovization
    - Reranking

- Efficiency improvements on PCFGs
  - Beam thresholding
  - Heuristic filtering
Issues with PCFGs

- Independence assumptions:
  - Rule expansion is context-independent
    - Allows us to multiply probabilities

- Is this valid?

<table>
<thead>
<tr>
<th></th>
<th>Pronoun</th>
<th>Non-pronoun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>91%</td>
<td>9%</td>
</tr>
<tr>
<td>Object</td>
<td>34%</td>
<td>66%</td>
</tr>
</tbody>
</table>

- In Treebank: roughly equi-probable
- How can we handle this?
  - Condition on Subj/Obj with parent annotation
Issues with PCFGs

- Insufficient lexical conditioning
  - Present in pre-terminal rules

- Are there cases where other rules should be conditioned on words?

Different verbs & prepositions have different attachment preferences
Parser Issues

- PCFGs make many (unwarranted) independence assumptions
  - Structural Dependency
    - NP → Pronoun: much more likely in subject position
  - Lexical Dependency
    - Verb subcategorization
    - Coordination ambiguity
Improving PCFGs: Structural Dependencies

• How can we capture Subject/Object asymmetry?
  • E.g., $NP_{subj} \rightarrow \text{Pron}$ vs $NP_{obj} \rightarrow \text{Pron}$

• Parent annotation:
  • Annotate each node with parent in parse tree
    • E.g., $NP^S$ vs $NP^VP$
    • Also annotate pre-terminals:
      • $RB^ADVP$ vs $RB^VP$
      • $IN^SBAR$ vs $IN^PP$

• Can also split rules on other conditions
Parent Annotation
Parent Annotation

• Advantages:
  • Captures structural dependency in grammars

• Disadvantages:
  • Increases number of rules in grammar
  • Decreases amount of training per rule

• Strategies to search for optimal # of rules
Improving PCFGs: Lexical Dependencies

- Lexicalized rules:
  - Best known parsers: Collins, Charniak parsers
  - Each non-terminal annotated with its lexical head
    - E.g. verb with verb phrase, noun with noun phrase
  - Each rule must identify RHS element as head
    - Heads propagate up tree
  - Conceptually like adding 1 rule per head value

- VP(dumped) → VBD(dumped)NP(sacks)PP(into)
- VP(dumped) → VBD(dumped)NP(cats)PP(into)
Lexicalized PCFGs

- Also, add head tag to non-terminals
  - Head tag: Part-of-speech tag of head word
    - VP(dumped) → VBD(dumped)NP(sacks)PP(into)
    - VP(dumped,VBD) → VBD(dumped,VBD)NP(sacks,NNS)PP(into,IN)

- Two types of rules:
  - Lexical rules: pre-terminal → word
    - Deterministic, probability 1
  - Internal rules: all other expansions
    - Must estimate probabilities
Lexicalized Parse Tree

Internal Rules
- Top → S(prefer,V)
- S(prefer,V) → NP(I,Pron) VP(prefer,V)
- NP(I,Pron) → Pron(I,Pron)
- VP(prefer,V) → V(prefer,V) NP(flight,NN)
- NP(flight,NN) → Det(a,Det) Nom(flight,NN)
- PP(on,IN) → IN(on,IN) NP(NWA,NNP)

Lexical Rules
- Pron(I,Pron) → I
- V(prefer,V) → prefer
- Det(a,Det) → a
- NN(flight,NN) → flight
- IN(on,IN) → on
- NNP(NWA,NNP) → NWA
PLCFGs

- Issue: Too many rules
  - No way to find corpus with enough examples

- (Partial) Solution: Independence assumed
  - Condition rule on
    - Category of LHS, head
  - Condition head on
    - Category of LHS and parent’s head

\[
P(T, S) = \prod_{n \in T} p(r(n) \mid n, h(n)) \ast p(h(n) \mid n, h(m(n)))
\]
Disambiguation Example
Disambiguation Example

\[
P(\text{VP} \rightarrow \text{VBDNPPP} | \text{VP}, \text{dumped}) = \frac{C(\text{VP}(\text{dumped}) \rightarrow \text{VBDNPP})}{\sum_\beta C(\text{VP}(\text{dumped}) \rightarrow \beta)} = \frac{6}{9} = 0.67
\]

\[
p(\text{VP} \rightarrow \text{VBDNP} | \text{VP}, \text{dumped}) = \frac{C(\text{VP}(\text{dumped}) \rightarrow \text{VBDNP})}{\sum_\beta C(\text{VP}(\text{dumped}) \rightarrow \beta)} = \frac{0}{9} = 0
\]

\[
p(\text{int o} | \text{PP}, \text{dumped}) = \frac{C(\text{X}(\text{dumped}) \rightarrow \ldots \text{PP}(\text{int o}) \ldots)}{\sum_\beta C(\text{X}(\text{dumped}) \rightarrow \ldots \text{PP} \ldots)} = \frac{2}{9} = 0.22
\]

\[
p(\text{int o} | \text{PP}, \text{sacks}) = \frac{C(\text{X}(\text{sacks}) \rightarrow \ldots \text{PP}(\text{int o}) \ldots)}{\sum_\beta C(\text{X}(\text{sacks}) \rightarrow \ldots \text{PP} \ldots)} = \frac{0}{0}
\]
CNF Factorization & Markovization

• CNF factorization:
  • Converts n-ary branching to binary branching
  • Can maintain information about original structure
    • Neighborhood history and parent

• Issue:
  • Potentially explosive
    • If keep all context: 72 → 10K non-terminals!!!

• How much context should we keep?
  • What Markov order?
Different Markov Orders
Markovization & Costs
(Mohri & Roark 2006)

<table>
<thead>
<tr>
<th>PCFG</th>
<th>Time (s)</th>
<th>Words/s</th>
<th></th>
<th>V</th>
<th></th>
<th></th>
<th>P</th>
<th></th>
<th>LR</th>
<th>LP</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right-factored</td>
<td>4848</td>
<td>6.7</td>
<td>10105</td>
<td>23220</td>
<td>69.2</td>
<td>73.8</td>
<td>71.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right-factored, Markov order-2</td>
<td>1302</td>
<td>24.9</td>
<td>2492</td>
<td>11659</td>
<td>68.8</td>
<td>73.8</td>
<td>71.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right-factored, Markov order-1</td>
<td>445</td>
<td>72.7</td>
<td>564</td>
<td>6354</td>
<td>68.0</td>
<td>73.0</td>
<td>70.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right-factored, Markov order-0</td>
<td>206</td>
<td>157.1</td>
<td>99</td>
<td>3803</td>
<td>61.2</td>
<td>65.5</td>
<td>63.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parent-annotated, Right-factored, Markov order-2</td>
<td>7510</td>
<td>4.3</td>
<td>5876</td>
<td>22444</td>
<td>76.2</td>
<td>78.3</td>
<td>77.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Improving PCFGs: Tradeoffs

- **Tensions:**
  - Increase accuracy:
    - Increase specificity
    - E.g. Lexicalizing, Parent annotation, Markovization, etc
  - Increases grammar
    - Increases processing times
    - Increases training data requirements

- How can we balance?
Efficiency

- PCKY is $|G|n^3$
  - Grammar can be huge
  - Grammar can be extremely ambiguous
    - 100s of analyses not unusual, esp. for long sentences

- However, only care about best parses
  - Others can be pretty bad

- Can we use this to improve efficiency?
Beam Thresholding

- Inspired by beam search algorithm

- Assume low probability partial parses unlikely to yield high probability overall
  - Keep only top k most probably partial parses
    - Retain only k choices per cell
      - For large grammars, could be 50 or 100
      - For small grammars, 5 or 10
Heuristic Filtering

- Intuition: Some rules/partial parses are unlikely to end up in best parse. Don’t store those in table.

- Exclusions:
  - Low frequency: exclude singleton productions
  - Low probability: exclude constituents $x$ s.t. $\text{p}(x) < 10^{-200}$
  - Low relative probability:
    - Exclude $x$ if there exists $y$ s.t. $\text{p}(y) > 100 \times \text{p}(x)$
Reranking

- Issue: Locality
  - PCFG probabilities associated with rewrite rules
  - Context-free grammars
  - Approaches create new rules incorporating context:
    - Parent annotation, Markovization, lexicalization
- Other problems:
  - Increase rules, sparseness
- Need approach that incorporates broader, global info
Discriminative Parse Reranking

- General approach:
  - Parse using (L)PCFG
  - Obtain top-N parses
  - Re-rank top-N parses using better features

- Discriminative reranking
  - Use arbitrary features in reranker (MaxEnt)
    - E.g. right-branching-ness, speaker identity, conjunctive parallelism, fragment frequency, etc
Reranking Effectiveness

- How can reranking improve?
  - N-best includes the correct parse

- Estimate maximum improvement
  - **Oracle** parse selection
    - Selects correct parse from N-best
      - If it appears

  - Base accuracy: 0.897
  - Oracle accuracy on 50-best: 0.968

- Discriminative reranking: 0.917