Constituency Parsing

CMSC 723 / LING 723 / INST 725

MARINE CARPUAT
marine@cs.umd.edu
Today’s Agenda

• Grammar-based parsing with CFGs
  – CKY algorithm
• Dealing with ambiguity
  – Probabilistic CFGs
• Strategies for improvement
  – Rule rewriting / Lexicalization

Note: we’re back in sync with textbook
[Sections 13.1, 13.4.1, 14.1-14.6]
# Sample Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP\ VP$</td>
<td>Det → that</td>
</tr>
<tr>
<td>$S \rightarrow Aux\ NP\ VP$</td>
<td>Noun → book</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>Verb → book</td>
</tr>
<tr>
<td>$NP \rightarrow Pronoun$</td>
<td>Pronoun → I</td>
</tr>
<tr>
<td>$NP \rightarrow Proper-Noun$</td>
<td>Proper-Noun → Houston</td>
</tr>
<tr>
<td>$NP \rightarrow Det\ Nominal$</td>
<td>Aux → does</td>
</tr>
<tr>
<td>Nominal → Noun</td>
<td>Preposition → from</td>
</tr>
</tbody>
</table>
GRAMMAR-BASED PARSING: CKY
Grammar-based Parsing

• Problem setup
  – Input: string and a CFG
  – Output: parse tree assigning proper structure to input string

• “Proper structure”
  – Tree that covers all and only words in the input
  – Tree is rooted at an S
  – Derivations obey rules of the grammar
  – Usually, more than one parse tree...
Parsing Algorithms

• Parsing is (surprise) a search problem
• Two basic (= bad) algorithms:
  – Top-down search
  – Bottom-up search
• A “real” algorithms:
  – CKY parsing
Top-Down Search

• Observation: trees must be rooted with an S node

• Parsing strategy:
  – Start at top with an S node
  – Apply rules to build out trees
  – Work down toward leaves
Top-Down Search
Top-Down Search

```plaintext
S

S
NP VP

S
Aux NP VP

S
VP
```
Top-Down Search
Bottom-Up Search

• Observation: trees must cover all input words

• Parsing strategy:
  – Start at the bottom with input words
  – Build structure based on grammar
  – Work up towards the root S
Bottom-Up Search

Book that flight
Bottom-Up Search

Book that flight

Noun Det Noun                   Verb Det Noun
|        |        |        |        |
Book that flight                 Book that flight
Bottom-Up Search

Book that flight

Noun Det Noun  Verb Det Noun
  |     |     |     |
Book that flight Book that flight

Nominal Nominal Nominal
  |     |     |     |
Noun Det Noun Verb Det Noun
  |     |     |     |
Book that flight Book that flight
Bottom-Up Search

Book that flight

Noun Det Noun
  Book that flight

Verb Det Noun
  Book that flight

Nominal

Noun Det Noun
  Book that flight

Nominal

Verb Det Noun
  Book that flight

Nominal

Noun

Book that flight

Nominal

NP

Nominal

Noun Det Noun
  Book that flight

Nominal

VP

Verb Det Noun
  Book that flight

Nominal

Verb Det Noun
  Book that flight

Nominal

NP

Nominal

Noun

Book that flight
Bottom-Up Search

Book that flight

Noun Det Noun
  Book that flight

Verb Det Noun
  Book that flight

Nominal
  Noun Det Noun
    Book that flight

Nominal
  Verb Det Noun
    Book that flight

Nominal
  Nominal
    Noun Det Noun
      Book that flight

Nominal
  Nominal
    Verb Det Noun
      Book that flight

Nominal
  NP
    Nominal
      Noun Det Noun
        Book that flight

Nominal
  VP
    Nominal
      Verb Det Noun
        Book that flight

Nominal
  NP
    Nominal
      Verb Det Noun
        Book that flight

Nominal
  NP
    Nominal
      Book that flight
Top-Down vs. Bottom-Up

• Top-down search
  – Only searches valid trees
  – But, considers trees that are not consistent with any of the words

• Bottom-up search
  – Only builds trees consistent with the input
  – But, considers trees that don’t lead anywhere
Parsing as Search

• Search involves controlling choices in the search space:
  – Which node to focus on in building structure
  – Which grammar rule to apply

• General strategy: backtracking
  – Make a choice, if it works out then fine
  – If not, back up and make a different choice
Backtracking isn’t enough!

2 key issues remain

• Ambiguity

• Shared sub-problems
Ambiguity

S
  NP
    Pronoun
      I
    Verb
      shot
    Det
      an
    Nominal
      Noun
        in my pajamas
elephant
  VP
  NP
    Nominal
      PP
        in my pajamas

S
  NP
    Pronoun
    Verb
      shot
    Det
      an
    Nominal
      Noun
        elephant
Shared Sub-Problems

• Observation: ambiguous parses still share sub-trees
• We don’t want to redo work that’s already been done
• Unfortunately, naïve backtracking leads to duplicate work
Efficient Parsing with the CKY Algorithm

• Dynamic programming to the rescue!
• Intuition: store partial results in tables
  – Thus avoid repeated work on shared sub-problems
  – Thus efficiently store ambiguous structures with shared sub-parts

• We’ll cover one example
  – CKY: roughly, bottom-up
CKY Parsing: CNF

• CKY parsing requires that the grammar consist of \( \varepsilon \)-free, binary rules = Chomsky Normal Form
  – All rules of the form:

\[
A \rightarrow B \ C \\
D \rightarrow w
\]

  – What does the tree look like?
CKY Parsing with Arbitrary CFGs

- What if my grammar has rules like VP → NP PP PP
  - Problem: can’t apply CKY!
  - Solution: rewrite grammar into CNF
    - Introduce new intermediate non-terminals into the grammar
      - Introduce new intermediate non-terminals into the grammar

\[
\begin{align*}
A &\rightarrow B \ C \ D \\
A &\rightarrow X \ D \\
X &\rightarrow B \ C
\end{align*}
\]

(Where X is a symbol that doesn’t occur anywhere else in the grammar)
## Sample Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$Det \rightarrow that \mid this \mid a$</td>
</tr>
<tr>
<td>$S \rightarrow Aux \ NP \ VP$</td>
<td>$Noun \rightarrow book \mid flight \mid meal \mid money$</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$Verb \rightarrow book \mid include \mid prefer$</td>
</tr>
<tr>
<td>$NP \rightarrow Pronoun$</td>
<td>$Pronoun \rightarrow I \mid she \mid me$</td>
</tr>
<tr>
<td>$NP \rightarrow Proper-Noun$</td>
<td>$Proper-Noun \rightarrow Houston \mid NWA$</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ Nominal$</td>
<td>$Aux \rightarrow does$</td>
</tr>
<tr>
<td>Nominal $\rightarrow$ Noun</td>
<td>$Preposition \rightarrow from \mid to \mid on \mid near \mid through$</td>
</tr>
<tr>
<td>Nominal $\rightarrow$ Nominal Noun</td>
<td></td>
</tr>
<tr>
<td>Nominal $\rightarrow$ Nominal PP</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow$ Verb</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow$ Verb NP</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow$ Verb NP PP</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow$ Verb PP</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow$ VP PP</td>
<td></td>
</tr>
<tr>
<td>$PP \rightarrow$ Preposition NP</td>
<td></td>
</tr>
</tbody>
</table>
## CNF Conversion

<table>
<thead>
<tr>
<th>Original Grammar</th>
<th>CNF Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow NP \ VP )</td>
<td>( S \rightarrow NP \ VP )</td>
</tr>
<tr>
<td>( S \rightarrow Aux \ NP \ VP )</td>
<td>( S \rightarrow X1 \ VP )</td>
</tr>
<tr>
<td>( X1 \rightarrow Aux \ NP )</td>
<td></td>
</tr>
<tr>
<td>( S \rightarrow VP )</td>
<td>( S \rightarrow book</td>
</tr>
<tr>
<td>( S \rightarrow book</td>
<td>include</td>
</tr>
<tr>
<td>( S \rightarrow Verb \ NP )</td>
<td>( S \rightarrow Verb \ NP )</td>
</tr>
<tr>
<td>( S \rightarrow X2 PP )</td>
<td>( S \rightarrow X2 PP )</td>
</tr>
<tr>
<td>( S \rightarrow Verb \ PP )</td>
<td>( S \rightarrow Verb \ PP )</td>
</tr>
<tr>
<td>( S \rightarrow VP \ PP )</td>
<td>( S \rightarrow VP \ PP )</td>
</tr>
<tr>
<td>( NP \rightarrow Pronoun )</td>
<td>( NP \rightarrow I</td>
</tr>
<tr>
<td>( NP \rightarrow Proper-Noun )</td>
<td>( NP \rightarrow TWA</td>
</tr>
<tr>
<td>( NP \rightarrow Det Nominal )</td>
<td>( NP \rightarrow Det Nominal )</td>
</tr>
<tr>
<td>( Nominal \rightarrow Noun )</td>
<td>( Nominal \rightarrow book</td>
</tr>
<tr>
<td>( Nominal \rightarrow Nominal Noun )</td>
<td>( Nominal \rightarrow Nominal Noun )</td>
</tr>
<tr>
<td>( Nominal \rightarrow Nominal PP )</td>
<td>( Nominal \rightarrow Nominal PP )</td>
</tr>
<tr>
<td>( VP \rightarrow Verb )</td>
<td>( VP \rightarrow book</td>
</tr>
<tr>
<td>( VP \rightarrow Verb \ NP )</td>
<td>( VP \rightarrow Verb \ NP )</td>
</tr>
<tr>
<td>( VP \rightarrow Verb \ NP \ PP )</td>
<td>( VP \rightarrow X2 PP )</td>
</tr>
<tr>
<td>( X2 \rightarrow Verb \ NP )</td>
<td>( VP \rightarrow Verb \ NP )</td>
</tr>
<tr>
<td>( VP \rightarrow Verb \ PP )</td>
<td>( VP \rightarrow VP \ PP )</td>
</tr>
<tr>
<td>( VP \rightarrow VP \ PP )</td>
<td>( PP \rightarrow Preposition NP )</td>
</tr>
<tr>
<td>( PP \rightarrow Preposition NP )</td>
<td></td>
</tr>
</tbody>
</table>
CKY Parsing: Intuition

- Consider the rule $D \to w$
  - Terminal (word) forms a constituent
  - Trivial to apply
- Consider the rule $A \to B \ C$
  - If there is an $A$ somewhere in the input then there must be a $B$ followed by a $C$ in the input
  - First, precisely define span $[i, j]$
  - If $A$ spans from $i$ to $j$ in the input then there must be some $k$ such that $i < k < j$
  - Easy to apply: we just need to try different values for $k$
CKY Parsing: Table

- Any constituent can conceivably span \([i, j]\) for all \(0 \leq i < j \leq N\), where \(N = \text{length of input string}\)
  - We need an \(N \times N\) table to keep track of all spans...
  - But we only need half of the table

- Semantics of table: cell \([i, j]\) contains \(A\) iff \(A\) spans \(i\) to \(j\) in the input string
  - Of course, must be allowed by the grammar!

<table>
<thead>
<tr>
<th>FROM:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</td>
<td>0–6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1–2</td>
<td>1–3</td>
<td>1–4</td>
<td>1–5</td>
<td>1–6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2–3</td>
<td>2–4</td>
<td>2–5</td>
<td>2–6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4–5</td>
<td>4–6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5–6</td>
</tr>
</tbody>
</table>
CKY Parsing: Table-Filling

- In order for $A$ to span $[i, j]$
  - $A \rightarrow BC$ is a rule in the grammar, and
  - There must be a $B$ in $[i, k]$ and a $C$ in $[k, j]$ for some $i<k<j$

- Operationally
  - To apply rule $A \rightarrow BC$, look for a $B$ in $[i, k]$ and a $C$ in $[k, j]$
  - In the table: look left in the row and down in the column

<table>
<thead>
<tr>
<th>FROM:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-1</td>
<td>0-2</td>
<td>0-3</td>
<td>0-4</td>
<td>0-5</td>
<td>0-6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1-2</td>
<td>1-3</td>
<td>1-4</td>
<td>1-5</td>
<td>1-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2-3</td>
<td>2-4</td>
<td>2-5</td>
<td>2-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3-4</td>
<td>3-5</td>
<td>3-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4-5</td>
<td>4-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>5-6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CKY Parsing: Rule Application

note: mistake in book (Fig. 13.11, p 441), should be [0,n]
CKY Parsing: Canonical Ordering

• Standard CKY algorithm:
  – Fill the table a column at a time, from left to right, bottom to top
  – Whenever we’re filling a cell, the parts needed are already in the table (to the left and below)

• Nice property: processes input left to right, word at a time
CKY Parsing: Ordering Illustrated

<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through</th>
<th>Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>S, VP, Verb Nominal, Noun [0,1]</td>
<td>S, VP, X2 [0,2]</td>
<td>S, VP, X2 [0,3]</td>
<td>S, VP, X2 [0,4]</td>
<td>S, VP, X2 [0,5]</td>
</tr>
<tr>
<td>Det [1,2]</td>
<td>NP [1,3]</td>
<td>NP [1,4]</td>
<td>Nominal [1,5]</td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing the ordering process in CKY Parsing.
function CKY-PARSE(words, grammar) returns table

for j ← from 1 to LENGTH(words) do
    table[j - 1, j] ← \{A | A \rightarrow \text{words}[j] \in grammar\}
for i ← from j - 2 downto 0 do
    for k ← i + 1 to j - 1 do
        table[i, j] ← table[i, j] ∪
        \{A | A \rightarrow BC \in grammar,\n        B \in \text{table}[i, k],\n        C \in \text{table}[k, j]\}
CKY Parsing: Recognize or Parse

• Is this really a parser?

• Recognizer to parser: add backpointers!
### CKY: Example

<table>
<thead>
<tr>
<th></th>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through</th>
<th>Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>S, VP, Verb, Nominal, Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,2</td>
<td>Det</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,3</td>
<td>NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,4</td>
<td>Nominal, Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>1,2</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>1,4</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>1,5</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>2,4</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>2,5</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>3,4</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>3,5</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>4,5</td>
<td></td>
<td></td>
<td></td>
<td>NP, Proper-Noun</td>
<td></td>
</tr>
</tbody>
</table>

Filling column 5
Recall our CNF grammar:

\[
\begin{align*}
S & \rightarrow \text{NP VP} \\
S & \rightarrow \text{X1 VP} \\
X1 & \rightarrow \text{Aux NP} \\
S & \rightarrow \text{book} \mid \text{include} \mid \text{prefer} \\
S & \rightarrow \text{Verb NP} \\
S & \rightarrow \text{X2 PP} \\
S & \rightarrow \text{Verb PP} \\
S & \rightarrow \text{VP PP} \\
\text{NP} & \rightarrow \text{I} \mid \text{she} \mid \text{me} \\
\text{NP} & \rightarrow \text{TWA} \mid \text{Houston} \\
\text{NP} & \rightarrow \text{Det Nominal} \\
\text{Nominal} & \rightarrow \text{book} \mid \text{flight} \mid \text{meal} \mid \text{money} \\
\text{Nominal} & \rightarrow \text{Nominal Noun} \\
\text{Nominal} & \rightarrow \text{Nominal PP} \\
\text{VP} & \rightarrow \text{book} \mid \text{include} \mid \text{prefer} \\
\text{VP} & \rightarrow \text{Verb NP} \\
\text{VP} & \rightarrow \text{X2 PP} \\
\text{X2} & \rightarrow \text{Verb NP} \\
\text{VP} & \rightarrow \text{Verb PP} \\
\text{VP} & \rightarrow \text{VP PP} \\
\text{PP} & \rightarrow \text{Preposition NP}
\end{align*}
\]
CKY: Example

<table>
<thead>
<tr>
<th></th>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through</th>
<th>Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,1]</td>
<td>S, VP, Verb, Nominal, Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[0,2]</td>
<td>Det</td>
<td>NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1,2]</td>
<td>Nominal, Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2,3]</td>
<td>Prep</td>
<td>PP</td>
<td>NP, Proper-Noun</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3,4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4,5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recall our CNF grammar:

\[
\begin{align*}
S & \rightarrow NP \ VP \\
S & \rightarrow X1 \ VP \\
X1 & \rightarrow Aux \ NP \\
S & \rightarrow book \mid include \mid prefer \\
S & \rightarrow Verb \ NP \\
S & \rightarrow X2 \ PP \\
S & \rightarrow Verb \ PP \\
S & \rightarrow VP \ PP \\
NP & \rightarrow I \mid she \mid me \\
NP & \rightarrow TWA \mid Houston \\
NP & \rightarrow Det \ Nominal \\
Nominal & \rightarrow book \mid flight \mid meal \mid money \\
Nominal & \rightarrow Nominal \ Noun \\
Nominal & \rightarrow Nominal \ PP \\
VP & \rightarrow book \mid include \mid prefer \\
VP & \rightarrow Verb \ NP \\
VP & \rightarrow X2 \ PP \\
X2 & \rightarrow Verb \ NP \\
VP & \rightarrow Verb \ PP \\
VP & \rightarrow VP \ PP \\
PP & \rightarrow Preposition \ NP
\end{align*}
\]
Back to Ambiguity

• Did we solve it?

• No: CKY returns multiple parse trees...
  – Plus: compact encoding with shared sub-trees
  – Plus: work deriving shared sub-trees is reused
  – Minus: algorithm doesn’t tell us which parse is correct
PROBABILISTIC CONTEXT-FREE GRAMMARS
Simple Probability Model

• A derivation (tree) consists of the bag of grammar rules that are in the tree
  – The probability of a tree is the product of the probabilities of the rules in the derivation.

\[ P(T, S) = \prod_{\text{node} \in T} P(\text{rule}(n)) \]
Rule Probabilities

• What’s the probability of a rule?

• Start at the top...
  – A tree should have an S at the top. So given that we know we need an S, we can ask about the probability of each particular S rule in the grammar: \( P(\text{particular rule} \mid S) \)

• In general we need \( P(\alpha \rightarrow \beta \mid \alpha) \) for each rule in the grammar
Training the Model

• We can get the estimates we need from a treebank

\[ P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)} \]

For example, to get the probability for a particular VP rule:
1. count all the times the rule is used
2. divide by the number of VPs overall.
How can we get the best (most probable) parse for a given input?

1. Enumerate all the trees for a sentence

2. Assign a probability to each using the model

3. Return the argmax
Consider...

– *Book the dinner flight*
Examples

- These trees consist of the following rules.

<table>
<thead>
<tr>
<th>Rules</th>
<th>P</th>
<th>Rules</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → VP</td>
<td>.05</td>
<td>S → VP</td>
<td>.05</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td>.20</td>
<td>VP → Verb NP NP</td>
<td>.10</td>
</tr>
<tr>
<td>NP → Det Nominal</td>
<td>.20</td>
<td>NP → Det Nominal</td>
<td>.20</td>
</tr>
<tr>
<td>Nominal → Nominal Noun</td>
<td>.20</td>
<td>Nominal → Noun</td>
<td>.75</td>
</tr>
<tr>
<td>Nominal → Noun</td>
<td>.75</td>
<td>Nominal → Noun</td>
<td>.75</td>
</tr>
<tr>
<td>Verb → book</td>
<td>.30</td>
<td>Verb → book</td>
<td>.30</td>
</tr>
<tr>
<td>Det → the</td>
<td>.60</td>
<td>Det → the</td>
<td>.60</td>
</tr>
<tr>
<td>Noun → dinner</td>
<td>.10</td>
<td>Noun → dinner</td>
<td>.10</td>
</tr>
<tr>
<td>Noun → flights</td>
<td>.40</td>
<td>Noun → flights</td>
<td>.40</td>
</tr>
</tbody>
</table>

\[
P(T_{left}) = 0.05 \times 0.20 \times 0.20 \times 0.20 \times 0.75 \times 0.30 \times 0.60 \times 0.10 \times 0.40 = 2.2 \times 10^{-6}
\]

\[
P(T_{right}) = 0.05 \times 0.10 \times 0.20 \times 0.15 \times 0.75 \times 0.75 \times 0.30 \times 0.60 \times 0.10 \times 0.40 = 6.1 \times 10^{-7}
\]
Dynamic Programming

• Of course, as with normal parsing we don’t really want to do it that way...
• Instead, we need to exploit dynamic programming
  – For the parsing (as with CKY)
  – And for computing the probabilities and returning the best parse (as with Viterbi and HMMs)
Probabilistic CKY

• Store probabilities of constituents in the table as they are derived:
  – \( \text{table}[i,j,A] = \text{probability of constituent } A \text{ that spans positions } i \text{ through } j \text{ in input} \)

• If \( A \) is derived from the rule \( A \rightarrow B \ C \):
  – \( \text{table}[i,j,A] = P(A \rightarrow B \ C \mid A) \times \text{table}[i,k,B] \times \text{table}[k,j,C] \)
  – Where
    • \( P(A \rightarrow B \ C \mid A) \) is the rule probability
    • \( \text{table}[i,k,B] \) and \( \text{table}[k,j,C] \) are already in the table given the way that CKY operates

• We only store the MAX probability over all the \( A \) rules.
function PROBABILISTIC-CKY(words, grammar) returns most probable parse and its probability

for \( j \leftarrow \text{from} \ 1 \ \text{to} \ \text{LENGTH}(\text{words}) \) do
  for all \( \{ A \mid A \rightarrow \text{words}[j] \in \text{grammar} \} \)
    \( \text{table}[j-1, j, A] \leftarrow P(A \rightarrow \text{words}[j]) \)
  for \( i \leftarrow \text{from} \ j-2 \ \text{downto} \ 0 \) do
    for \( k \leftarrow i+1 \ \text{to} \ j-1 \) do
      for all \( \{ A \mid A \rightarrow BC \in \text{grammar}, \)
        and \( \text{table}[i, k, B] > 0 \ \text{and} \ \text{table}[k, j, C] > 0 \} \)
        if \( (\text{table}[i, j, A] < P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]) \) then
          \( \text{table}[i, j, A] \leftarrow P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C] \)
          \( \text{back}[i, j, A] \leftarrow \{ k, B, C \} \)
      return BUILD_TREE(back[1, \text{LENGTH}(\text{words}), S]), table[1, \text{LENGTH}(\text{words}), S]
Problems with PCFGs

• The probability model we’re using is just based on the bag of rules in the derivation...
  1. Doesn’t take the actual words into account in any useful way.
  2. Doesn’t take into account *where* in the derivation a rule is used
  3. *Doesn’t work terribly well*
IMPROVING OUR PARSER
Problem example: PP Attachment
Problem example: PP Attachment

```
S
  NP
    NNS workers
  VP
    VBD dumped
    NP
      NNS
        sacks
  PP
    P
      into
    NP
      DT a
      NN bin
```
Improved Approaches

There are two approaches to overcoming these shortcomings

1. Rewrite the grammar to better capture the dependencies among rules

2. Integrate lexical dependencies into the model
Solution 1: Rule Rewriting

• Goal:
  – capture local tree information
  – so that the rules capture the regularities we want

• Approach:
  – split and merge the non-terminals in the grammar
Example: Splitting NPs (1/2)

• Our CFG rules for NPs don’t condition on where in a tree the rule is applied.

• But we know that not all the rules occur with equal frequency in all contexts.
  – Consider NPs that involve pronouns vs. those that don’t.

<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>
The rules are now

- NP^S -> PRP
- NP^VP -> DT
- VP^S -> NP^VP

Non-terminals NP^S and NP^VP capture the subject/object and pronoun/full NP cases.
Solution 2: Lexicalized Grammars

• Lexicalize the grammars with heads

• Compute the rule probabilities on these lexicalized rules

• Run Prob CKY as before
Lexicalized Grammars: Example

S(dumped)

NP(workers)
  NNS(workers)
    workers
  VBD(dumped)
    dumped

VP(dumped)
  NP(sacks)
    NNS(sacks)
      sacks
    P(into)
      into
  PP(into)
    DT(a)
      a
    NN(bin)
      bin

NP(bin)
How can we learn probabilities for lexicalized rules?

- We used to have
  - VP -> V NP PP
  - \( P(\text{rule}|\text{VP}) = \frac{\text{count of this rule divided by the number of VPs in a treebank}}{\text{the number of VPs in a treebank}} \)

- Now we have fully lexicalized rules...
  - VP(dumped) -> V(dumped) NP(sacks)PP(into)
  - \( P(\text{r}|\text{VP} \wedge \text{dumped} \text{ is the verb} \wedge \text{sacks} \text{ is the head of the NP} \wedge \text{into} \text{ is the head of the PP}) \)
We need to make independence assumptions

• Strategies: exploit independence and collect the statistics we can get
  • Many many ways to do this...

• Let’s consider one generative story: given a rule we’ll
  1. Generate the head
  2. Generate the stuff to the left of the head
  3. Generate the stuff to the right of the head
From the generative story to rule probabilities...

The rule probability for

$$P(\text{VP(dumped}, \text{VBD}) \rightarrow \text{VBD(dumped, VBD)} \ \text{NP(sacks, NNS)} \ \text{PP(into, P)})$$

Can be estimated as

$$P_H(\text{VBD}| \text{VP, dumped}) \times P_L(\text{STOP}| \text{VP, VBD, dumped}) \times P_R(\text{NP(sacks, NNS)}| \text{VP, VBD, dumped}) \times P_R(\text{PP(into, P)}| \text{VP, VBD, dumped}) \times P_R(\text{STOP}| \text{VP, VBD, dumped})$$
Framework

• That’s just one simple model
  – “Collins Model 1”

• You can imagine a gazzillion other assumptions that might lead to better models
  – make sure that you can get the counts you need
  – make sure they can get exploited efficiently during decoding
Wrapping up... (1/3)

• Grammar-based parsing with CFGs
  – CKY algorithm

• Dealing with ambiguity
  – Probabilistic CFGs

• Strategies for improving the model
  – Rule rewriting / Lexicalization
Wrapping Up... (2/3)

• 2 flavors of syntactic representations
  – Dependency Grammars
  – Constituency Grammars

• Parsing = producing a syntactic analysis given an input sentence
  – Grammar-based algorithms (e.g. CKY for CFGs)
  – Data-driven algorithms (e.g., transition-based and graph-based parsing for dependency)
Wrapping Up... (3/3)

• State-of-the-art
  – Many useful parsing tools
    http://www.maltparser.org/
    http://nlp.stanford.edu/software/lex-parser.shtml
    ...
  - Used for many tasks (e.g., information extraction, machine translation)

- Still some important open questions
  - Beyond English?
  - Informal language?