Dependency Parsing II

CMSC 470

Marine Carpuat
Graph-based Dependency Parsing

Slides credit: Joakim Nivre
Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - $G'$ is a tree (acyclic)

- A spanning tree of the following (multi-)digraphs
Dependency Parsing as Finding the Maximum Spanning Tree

- Views parsing as finding the best directed spanning tree
  - of multi-digraph that captures all possible dependencies in a sentence
  - needs a score that quantifies how good a tree is

- Assume we have an **arc factored** model
  i.e. weight of graph can be factored as sum or product of weights of its arcs

- Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us
  - Recursive algorithm
  - Naïve implementation: O(n^3)
Chu-Liu-Edmonds illustrated (for unlabeled dependency parsing)
Chu-Liu-Edmonds illustrated

- Find highest scoring incoming arc for each vertex

- If this is a tree, then we have found MST!!
Chu-Liu-Edmonds illustrated

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle
Chu-Liu-Edmonds illustrated

- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John → Mary is 3 and saw → Mary is 30
Chu-Liu-Edmonds illustrated

- **Incoming arc weights**
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29
This is a tree and the MST for the contracted graph!!

Go back up recursive call and reconstruct final graph
Chu-Liu-Edmonds algorithm

```
function MAXSPANNINGTREE(G=(V,E), root, score) returns spanning tree

    F ← []
    T' ← []
    score' ← []

    for each v ∈ V do
        bestInEdge ← argmax e=(u,v)∈ E score[e]
        F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
        score'[e] ← score[e] − score[bestInEdge]

    if T=(V,F) is a spanning tree then return it
    else
        C ← a cycle in F
        G' ← CONTRACT(G, C)
        T′ ← MAXSPANNINGTREE(G', root, score')
        T ← EXPAND(T', C)
    return T

function CONTRACT(G, C) returns contracted graph

function EXPAND(T, C) returns expanded graph
```

*Figure 5.13* The Chu-Liu-Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.
For dependency parsing, we will view arc weights as linear classifiers

\[ w_{ij}^k = e^{w \cdot f(i,j,k)} \]

- Arc weights are a linear combination of features of the arc, \( f \), and a corresponding weight vector \( w \)
- Raised to an exponent (simplifies some math ...)
- What arc features?
Example of classifier features

John saw Mary McGuire yesterday with his telescope

- Features from [McDonald et al. 2005]:
  - Identities of the words $w_i$ and $w_j$ and the label $l_k$

  head=saw & dependent=with
Typical classifier features

• Word forms, lemmas, and parts of speech of the headword and its dependent
• Corresponding features derived from the contexts before, after and between the words
• Word embeddings
• The dependency relation itself
• The direction of the relation (to the right or left)
• The distance from the head to the dependent
• ...

How to score a graph G using features?

By definition of arc weights as linear classifiers

\[
G = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}
\]

\[
= \arg \max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}
\]

\[
= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w \cdot f(i,j,k)
\]

\[
= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) = \arg \max_{G \in T(G_x)} w \cdot f(G)
\]
Learning parameters with the Structured Perceptron

Training data: $\mathcal{T} = \{(x_t, G_t)\}_{t=1}^{\mathcal{T}}$

1. $w^{(0)} = 0; \quad i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $G' = \arg \max_{G'} w^{(i)} \cdot f(G')$
5. if $G' \neq G_t$
6. $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7. $i = i + 1$
8. return $w^{(i)}$
This is the exact same perceptron algorithm as for multiclass classification, sequence labeling

\[ \hat{y} = \arg\max_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x, \hat{y}) \]

**Algorithm 40 StructuredPerceptronTrain(D, MaxIter)**

1. \( w \leftarrow 0 \) // initialize weights
2. for iter = 1 ... MaxIter do
3.   for all \((x,y) \in D\) do
4.     \( \hat{y} \leftarrow \arg\max_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x, \hat{y}) \) // compute prediction
5.     if \( \hat{y} \neq y \) then
6.       \( w \leftarrow w + \phi(x, y) - \phi(x, \hat{y}) \) // update weights
7.     end if
8.   end for
9. end for
10. return \( w \) // return learned weights

Algorithm from CIML chapter 17
Comparing dependency parsing algorithms

Transition-based
• Locally trained
• Use greedy search algorithm
• Can define features over a rich history of parsing decisions

Graph-based
• Globally trained
• Use exact search algorithm
• Can only define features over a limited history of parsing decisions to maintain arc-factored assumption
Dependency Parsing: what you should know

• Interpreting dependency trees
• Transition-based dependency parsing
  • Shift-reduce parsing
  • Transition systems: arc standard, arc eager
  • Oracle algorithm: how to obtain a transition sequence given a tree
  • How to construct a multiclass classifier to predict parsing actions
  • What transition-based parsers can and cannot do
  • That transition-based parsers provide a flexible framework that allows many extensions
    • such as RNNs vs feature engineering, non-projectivity (but I don’t expect you to memorize these algorithms)
• Graph-based dependency parsing
  • Chu-Liu-Edmonds algorithm
  • Structured perceptron
Parsing with Context Free Grammars
Agenda

• Grammar-based parsing with CFGs
  • CKY algorithm
• Dealing with ambiguity
  • Probabilistic CFGs
## Sample Grammar

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

• Two naive algorithms:
  • Top-down search
  • Bottom-up search

• A “real” algorithm:
  • 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
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
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
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 (Cocke Kasami Younger) Algorithm

• Solution: Dynamic programming
• 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 binary rules in 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 $VP \rightarrow NP \; PP$
  • Problem: can’t apply CKY!
  • Solution: rewrite grammar into CNF
    • Introduce new intermediate non-terminals into the grammar

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 \mid include \mid prefer )</td>
</tr>
<tr>
<td>( S \rightarrow Verb \ NP )</td>
<td>**</td>
</tr>
<tr>
<td>( S \rightarrow X2 \ PP )</td>
<td>**</td>
</tr>
<tr>
<td>( S \rightarrow Verb \ PP )</td>
<td>**</td>
</tr>
<tr>
<td>( S \rightarrow VP \ PP )</td>
<td>**</td>
</tr>
<tr>
<td>( NP \rightarrow Pronoun )</td>
<td>( NP \rightarrow I \mid she \mid me )</td>
</tr>
<tr>
<td>( NP \rightarrow Proper-Noun )</td>
<td>( NP \rightarrow TWA \mid Houston )</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 \mid flight \mid meal \mid money )</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 \mid include \mid prefer )</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>**</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>
CKY Parsing: Intuition

• Consider the rule $D \rightarrow w$
  • Terminal (word) forms a constituent
  • Trivial to apply

• Consider the rule $A \rightarrow B \ C$
  • “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 half of an \(N \times N\) table to keep track of all spans

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

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

• In order for A to span \([i, j]\)
  • A \(\rightarrow\) B C 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 \(\rightarrow\) B C, 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>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></td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</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>
</tr>
<tr>
<td>2</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>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>4–5</td>
<td>4–6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5–6</td>
<td></td>
</tr>
</tbody>
</table>
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></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>[0,2]</td>
<td>[0,3]</td>
<td>[0,4]</td>
<td>[0,5]</td>
</tr>
<tr>
<td>[1,2]</td>
<td>Det</td>
<td>NP</td>
<td>NP</td>
<td>Nominal, Noun</td>
<td>[2,4]</td>
</tr>
<tr>
<td>[2,3]</td>
<td>Nominal, Noun</td>
<td>Prep</td>
<td>PP</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>

Diagram showing the ordering process in CKY parsing with arrows indicating the path through the matrix.
CKY Algorithm

function CKY-PARSE(words, grammar) returns table

for $j \leftarrow$ from 1 to LENGTH(words) do
  \[ table[j-1, j] \leftarrow \{ A \mid A \rightarrow \text{words}[j] \in \text{grammar} \} \]

for $i \leftarrow$ from $j-2$ downto 0 do
  for $k \leftarrow i+1$ to $j-1$ do
    \[ table[i, j] \leftarrow table[i, j] \cup \{ A \mid A \rightarrow BC \in \text{grammar}, B \in table[i, k], C \in table[k, j] \} \]
CKY: Example

Filling column 5
Recall our CNF grammar:

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

\[
\begin{align*}
S & \rightarrow \text{NP VP} \\
S & \rightarrow \text{X1 VP} \\
\text{X1} & \rightarrow \text{Aux NP} \\
S & \rightarrow \text{book | include | prefer} \\
S & \rightarrow \text{Verb NP} \\
S & \rightarrow \text{X2 PP} \\
S & \rightarrow \text{Verb PP} \\
S & \rightarrow \text{VP PP} \\
\text{NP} & \rightarrow \text{I | she | me} \\
\text{NP} & \rightarrow \text{TWA | Houston} \\
\text{NP} & \rightarrow \text{Det Nominal} \\
\text{Nominal} & \rightarrow \text{book | flight | meal | money} \\
\text{Nominal} & \rightarrow \text{Nominal Noun} \\
\text{Nominal} & \rightarrow \text{Nominal PP} \\
\text{VP} & \rightarrow \text{book | include | prefer} \\
\text{VP} & \rightarrow \text{Verb NP} \\
\text{VP} & \rightarrow \text{X2 PP} \\
\text{X2} & \rightarrow \text{Verb NP} \\
\text{VP} & \rightarrow \text{Verb PP} \\
\text{VP} & \rightarrow \text{VP PP} \\
\text{PP} & \rightarrow \text{Preposition NP}
\end{align*}
\]
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></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>Nominal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2,4]</td>
<td>Prep</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3,4]</td>
<td>NP, Proper-Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[0,3]</td>
<td>S, VP, X2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[0,4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1,3]</td>
<td>NP</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>[2,5]</td>
<td>PP</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>
</tbody>
</table>
CKY: Example

Recall our CNF grammar:

\[
\begin{align*}
S & \rightarrow NP \ VP \\
S & \rightarrow X1 \ VP \\
X1 & \rightarrow Aux \ NP \\
S & \rightarrow book \mid include \mid prefer \\
S & \rightarrow Verb \ NP \\
S & \rightarrow X2 \ PP \\
S & \rightarrow VP \ PP \\
np & \rightarrow I \mid she \mid me \\
np & \rightarrow TWA \mid Houston \\
np & \rightarrow Det \ Nominal \\
Nominal & \rightarrow book \mid flight \mid meal \mid money \\
Nominal & \rightarrow Nominal \ Noun \\
Nominal & \rightarrow Nominal \ PP \\
vp & \rightarrow book \mid include \mid prefer \\
vp & \rightarrow Verb \ NP \\
vp & \rightarrow X2 \ PP \\
x2 & \rightarrow Verb \ NP \\
vp & \rightarrow Verb \ PP \\
vp & \rightarrow VP \ PP \\
pp & \rightarrow Preposition \ NP \\
\end{align*}
\]
CKY: Example

```
Book   the   flight   through   Houston

S, VP, Verb
Nominal, Noun
[0,1]   [0,2]   [0,3]   [0,4]   [0,5]

Det   NP   NP
[1,2]   [1,3]   [1,4]   [1,5]

Nominal, Noun
[2,3]   [2,4]   [2,5]

Prep   PP
[3,4]   [3,5]

NP, Proper-Noun
[4,5]
```
CKY Parsing: Recognize or Parse

• Recognizer
  • Output is binary
  • Can the complete span of the sentence be covered by an S symbol?

• Parser
  • Output is a parse tree
  • From recognizer to parser: add backpointers!
Ambiguity

• CKY can return 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!
Ambiguity
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.

\[ P(T,S) = \prod_{node \in T} P(rule(n)) \]
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
  for each rule in the grammar $P(\alpha \rightarrow \beta \mid \alpha)$
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.
Parsing (Decoding)

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
Example

• Consider...
  • *Book the dinner flight*

```
  S
   VP
     Verb
       Book
     Det
       the
     Nominal
       Nominal
       Noun
       dinner
     Noun
       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>

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

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

• Store probabilities of constituents in the table
  • \text{table}[i,j,A] = \text{probability of constituent A that spans positions i through j in input}

• If \( A \) is derived from the rule \( A \rightarrow B C \):
  • \( \text{table}[i,j,A] = P(A \rightarrow B C | A) \times \text{table}[i,k,B] \times \text{table}[k,j,C] \)
  • Where
    • \( P(A \rightarrow B C | A) \) 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

• Only store the MAX probability over all the A rules.
## Probabilistic CKY

**function** PROBABILISTIC-CKY(words, grammar) **returns** most probable parse and its probability

**for** $j \leftarrow 1$ **to** LENGTH(words) **do**

**for all** \{ $A$ | $A \rightarrow$ words[$j$] $\in$ grammar \}

$\text{table}[j - 1, j, A] \leftarrow P(A \rightarrow \text{words}[j])$

**for** $i \leftarrow j - 2$ **downto** 0 **do**

**for** $k \leftarrow i + 1$ **to** $j - 1$ **do**

**for all** \{ $A$ | $A \rightarrow BC$ $\in$ grammar, and $\text{table}[i, k, B] > 0$ and $\text{table}[k, j, C] > 0$ \}

\[
\text{if} (\text{table}[i, j, A] < P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]) \text{ then}
\]

$\text{table}[i, j, A] \leftarrow P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]$

$\text{back}[i, j, A] \leftarrow \{k, B, C\}$

**return** BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
Grammar-based parsing with CFGs

**Summary**

- CKY algorithm finds all the parses of a given sentence efficiently
  - Using dynamic programming

- Probabilistic CFGs help deal with ambiguity
  - Requires computing probability of rules based on their frequency in the training data

- Lexicalized grammars help improve performance further