Parsing

Parsing is the field of NLP interested in automatically determining the syntactic structure of a sentence. Parsing can also be thought of as determining what sentences are “valid” English sentences.

We have a grammar, determine the possible parse tree(s)

Let’s start with parsing with a CFG (no probabilities)

\[
\begin{align*}
S & \to NP \ VP \\
NP & \to PRP \\
NP & \to N \ PP \\
VP & \to V \ NP \\
VP & \to V \ NP \ PP \\
PP & \to IN \ N \\
PREP & \to \text{I} \\
V & \to \text{eat} \\
N & \to \text{sushi} \\
N & \to \text{tuna} \\
IN & \to \text{with}
\end{align*}
\]

I eat sushi with tuna

approaches? algorithms?
### Parsing

#### Top-down parsing
- ends up doing a lot of repeated work
- doesn’t take into account the words in the sentence until the end!

#### Bottom-up parsing
- constrain based on the words
- avoids repeated work (dynamic programming)
- doesn’t take into account the high-level structure until the end!
- CKY parser

---

#### Parsing Example

```
S
  VP
    Vbs
      Det
        Nominal
          book
            that
          flight
```

---

#### Top Down Parsing

```
S
  NP
      VP
        Pronoun
```

---
Top Down Parsing

S
NP VP
Det Nominal
book

Top Down Parsing

S
Aux NP VP

Top Down Parsing

S
Aux NP VP

Top Down Parsing

S
VP

Top Down Parsing

S
Aux NP VP
Top Down Parsing

S
  VP
  Verb

Top Down Parsing

S
  VP
  Verb
  book

Top Down Parsing

S
  VP
  Verb
  book

Top Down Parsing

S
  VP
  Verb
  NP
Top Down Parsing

S

VP

Verb
NP

book

ProperNoun

that

Top Down Parsing

S

VP

Verb
NP

book

Det
Nominal

that

Noun

Top Down Parsing

S

VP

Verb
NP

book

Det
Nominal

that

Noun
Top Down Parsing

S
  \_ VP
    \_ Noun
      \_ Nominal
        \_ book
          \_ Det
            \_ that
              \_ Noun
                  \_ flight

Bottom Up Parsing

book that flight

Bottom Up Parsing

book that flight

Bottom Up Parsing

book that flight
Bottom Up Parsing

S
  V
  book

NP
  that
  flight

Nominal

Bottom Up Parsing

S
  V
  book

NP
  that
  flight

Nominal

Bottom Up Parsing

VP
  PP
  book

NP
  that
  flight

Nominal

Bottom Up Parsing

VP
  PP
  book

NP
  that
  flight

Nominal
Bottom Up Parsing

**Pros/Cons?**
- **Top-down:**
  - Only examines parses that could be valid parses (i.e. with an S on top)
  - Doesn’t take into account the actual words
- **Bottom-up:**
  - Only examines structures that have the actual words as the leaves
  - Examines sub-parses that may NOT result in a valid parse
Why is parsing hard?

Actual grammars are large

Lots of ambiguity!
- Most sentences have many parses
- Some sentences have a lot of parses
- Even for sentences that are not ambiguous, there is often ambiguity for subtrees (i.e. multiple ways to parse a phrase)

I saw the man on the hill with the telescope

What are some interpretations?

Structural Ambiguity Can Give Exponential Parses

“Me see a man the telescope the hill”
“Me see a man that has a telescope on the hill”
“Me see a man who was on a hill that has a telescope on it”
“Using a telescope, I saw a man who was on a hill”

Dynamic Programming Parsing

To avoid extensive repeated work you must cache intermediate results, specifically found constituents.

Caching (memoizing) is critical to obtaining a polynomial time parsing algorithm for CFGs.

Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where $n$ is the length of the input string.
Dynamic Programming Parsing Methods

**CKY** (Cocke-Kasami-Younger) algorithm based on bottom-up parsing and requires first normalizing the grammar.

**Earley parser** is based on top-down parsing and does not require normalizing grammar but is more complex.

These both fall under the general category of **chart parsers** which retain completed constituents in a chart.

---

**CKY parser: the chart**

- **Cell** \([i,j]\) contains all constituents covering words \(i\) through \(j\)
- **All constituents spanning 1-3 or “the man with”**
- **What does this cell represent?**

---

**CKY parser: the chart**

- **Cell** \([i,j]\) contains all constituents covering words \(i\) through \(j\)
- **How could we figure this out?**
### CKY parser: the chart

<table>
<thead>
<tr>
<th></th>
<th>Film</th>
<th>the</th>
<th>man</th>
<th>with</th>
<th>trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell[i,j] contains all constituents covering words i through j

Key: rules are binary and only have two constituents on the right hand side

- VP -> VB NP
- NP -> DT NN

See if we can make a new constituent combining any for "the" with any for "man with"

### CKY parser: the chart

<table>
<thead>
<tr>
<th></th>
<th>Film</th>
<th>the</th>
<th>man</th>
<th>with</th>
<th>trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell[i,j] contains all constituents covering words i through j

See if we can make a new constituent combining any for "the" with any for "man with"

### CKY parser: the chart

<table>
<thead>
<tr>
<th></th>
<th>Film</th>
<th>the</th>
<th>man</th>
<th>with</th>
<th>trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell[i,j] contains all constituents covering words i through j

What combinations do we need to consider when trying to put constituents here?
CKY parser: the chart

Cell[i,j] contains all constituents covering words i through j

See if we can make a new constituent combining any for “Film” with any for “the man with trust”

See if we can make a new constituent combining any for “Film the” with any for “man with trust”

See if we can make a new constituent combining any for “Film the man” with any for “trust”
What if our rules weren’t binary?

See if we can make a new constituent combining any for “Film” with any for “the man” with any for “with trust”

What order should we fill the entries in the chart?

Our dependencies are left and down
CKY parser: the chart

Cell\[i,j\] contains all constituents covering words \(i\) through \(j\)

From bottom to top, left to right

---

CKY parser: unary rules

Often, we will leave unary rules rather than converting to CNF

Do these complicate the algorithm?

Must check whenever we add a constituent to see if any unary rules apply

---

Film the man with trust

Top-left along the diagonals moving to the right
<table>
<thead>
<tr>
<th>Film</th>
<th>the</th>
<th>man</th>
<th>with</th>
<th>trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i=0)</td>
<td>0</td>
<td>NP</td>
<td>VP</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DT</td>
<td>NP</td>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>2</td>
<td>VB</td>
<td>NN</td>
<td>NP</td>
<td>VP</td>
</tr>
<tr>
<td>3</td>
<td>IN</td>
<td></td>
<td></td>
<td>VP</td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>NN</td>
<td>VP</td>
<td>VP</td>
</tr>
</tbody>
</table>

**CKY parser: the chart**
CKY parser: the chart

After we fill in the chart, how do we know if there is a parse?
- If there is an S in the upper right corner

What if we want an actual tree/parse?

CKY: some things to talk about

CKY: retrieving the parse
To add a constituent in a cell, we're applying a rule.

The references represent the smaller constituents we used to build this constituent.
CKY: retrieving the parse

We can store multiple derivations of each constituent

This representation is called a “parse forest”

It is often convenient to leave it in this form, rather than enumerate all possible parses. Why?

What about ambiguous parses?

CKY: some things to think about

CNF                   Actual grammar
\[ S \rightarrow VP \]
\[ VP \rightarrow V \_ NP \]
\[ NP \rightarrow DT \_ NN \]
\[ \_ \rightarrow \_ \]

We get a CNF parse tree but want one for the actual grammar

Ideas?

Parsing ambiguity

How can we decide between these?
A Simple PCFG

Probabilities!

S → NP VP 1.0  
NP → NP PP 0.4  
VP → V NP 0.7  
NP → astronomers 0.1  
VP → VP PP 0.3  
NP → ears 0.18  
PP → P NP 1.0  
P → with 1.0  
NP → stars 0.18  
V → saw 1.0  
NP → telescope 0.1

Parsing with PCFGs

How does this change our CKY algorithm?
- We need to keep track of the probability of a constituent.

How do we calculate the probability of a constituent?
- Product of the PCFG rule times the product of the probabilities of the sub-constituents (right hand sides).
- Building up the product from the bottom-up.

What if there are multiple ways of deriving a particular constituent?
- \( \text{max} \); pick the most likely derivation of that constituent.

Probabilistic CKY

Include in each cell a probability for each non-terminal.

Cell \([i,j]\) must retain the most probable derivation of each constituent (non-terminal) covering words \(i\) through \(j\).

When transforming the grammar to CNF, must set production probabilities to preserve the probability of derivations.
Probabilistic Grammar Conversion

Original Grammar

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.8</td>
</tr>
<tr>
<td>S → Aux NP VP</td>
<td>0.1</td>
</tr>
<tr>
<td>S → VP</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Chomsky Normal Form

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.8</td>
</tr>
<tr>
<td>S → X1 VP</td>
<td>0.1</td>
</tr>
<tr>
<td>S → book</td>
<td>0.01</td>
</tr>
<tr>
<td>S → include</td>
<td>0.04</td>
</tr>
<tr>
<td>S → prefer</td>
<td>0.006</td>
</tr>
<tr>
<td>NP → Pronoun</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → Aux NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → Proper-Noun</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → book</td>
<td>0.05</td>
</tr>
<tr>
<td>NP → Houston</td>
<td>0.03</td>
</tr>
<tr>
<td>NP → NWA</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Probabilistic CKY Parser

Book       the        flight    through  Houston
S :.01, VP:.1,
Verb:.5
Nominal:.03
Noun:.1
Det:.6
Nominal:.15
Noun:.5
None
NP:.6*.6*.15
= .054
NP → Det Nominal

What is the probability of the NP?

VP → Verb NP
NP → VP PP
PP → Prep NP

VP :.01, Verb:.5
Nominal:.03
Noun:.1
Det:.6
Nominal:.15
Noun:.5
None
VP → book | include | prefer
0.1      0.04        0.06
What is the probability of the VP?
Probabilistic CKY Parser

Book the flight through Houston

VP → Verb NP 0.5

NP → Det Nominal 0.5

Nominal → Nominal .15

Noun → Noun .5

Det → Det .6

Nominal → Nominal .15

Noun → Noun .5

None

VP → Verb NP 0.5

NP → Det Nominal 0.5

Nominal → Nominal .15

Noun → Noun .5

Det → Det .6

Nominal → Nominal .15

Noun → Noun .5

None

Prep → Prep .2

NP → PropNoun .8

PP → Prep NP 1.0*.2*.16 = .032
Probabilistic CKY Parser

Book the flight through Houston

S :.01, VP:.1, Verb:.5, Nominal:.03, Noun:.1, Det:.6, Nominal:.15, Noun:.5, None

NP:.6*.6*.15 =.054
VP:.5*.5*.054 =.0135
S:.05*.5*.054 =.00135

Prep:.2, NP:.16, PropNoun:.8

PP:1.0*.2*.16 =.032
Nominal:.5*.15*.032 =.0024

Which parse do we pick?

S → VP PP 0.03
S → Verb NP 0.05

Pick most probable parse, i.e. take max to combine probabilities of multiple derivations of each constituent in each cell.
Generic PCFG Limitations

PCFGs do not rely on specific words or concepts, only general structural disambiguation is possible (e.g. prefer to attach PPs to Nominals)

- Generic PCFGs cannot resolve syntactic ambiguities that require semantics to resolve, e.g. "ate with": fork vs. meatballs

Smoothing/dealing with out of vocabulary

MLE estimates are not always the best