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)

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

5

Parsing Example

S
  VP
    | book that flight
    | book that flight
    | book NP
      | Det Nominal
      | that Noun
      | flight

7

Top Down Parsing

S
  NP
  VP
    | Pronoun

8
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
 Aux  NP  VP

book

book
Top Down Parsing

21

Top Down Parsing

book

22

Top Down Parsing

book

23

Top Down Parsing

book

24
Top Down Parsing

Bottom Up Parsing

Bottom Up Parsing
Bottom Up Parsing

33

Book, that, flight

Nominal
Nominal
Noun
Noun
book
that
flight

Bottom Up Parsing

34

Nominal
Nominal
Noun
X
book
that
flight

Bottom Up Parsing

35

Nominal
Nominal
PP
Noun
book
that
flight

Bottom Up Parsing

36

Nominal
Nominal
PP
Noun
Det
book
that
flight
Bottom Up Parsing

49

VP

book

that

Nominal

flight

50

Bottom Up Parsing

VP

book

that

Nominal

flight

51

Bottom Up Parsing

52

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)

Structural Ambiguity Can Give Exponential Parses

"I was on the hill that has a telescope when I saw a man."

"I saw a man who was on the hill that has a telescope."  

"Using a telescope, I saw a man who was on a hill."  

"I was on the hill when I used the telescope to see a man."

I saw the man on the hill with the telescope

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

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

VP -> VB NP
NP -> DT NN

Film the man with trust

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

What combinations do we need to consider when trying to put constituents here?
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 man” with any for “with trust”

See if we can make a new constituent combining any for “Film the man with” with any for “trust”
**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>i=j=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=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 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

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

**CKY parser: the chart**

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

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>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**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
CKY parser: the chart

<table>
<thead>
<tr>
<th>i</th>
<th>j=0</th>
<th>j=1</th>
<th>j=2</th>
<th>j=3</th>
<th>j=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NN</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>DT</td>
<td>VP2</td>
<td>VP2</td>
<td>VP</td>
<td>VP2</td>
</tr>
<tr>
<td>2</td>
<td>NN</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>3</td>
<td>IN</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>VN</td>
<td>VN</td>
<td>VN</td>
<td>VN</td>
</tr>
</tbody>
</table>

Film the man with trust

S → VP
VP → VB NP
VP → VP2 PP
VP2 → VB NP
NP → DT NN
NP → NN
NP → NP PP
PP → IN NP
DT → the
IN → with
VB → film
VB → man
VB → trust
NN → man
NN → film
NN → trust

77 78
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

CKY: retrieving the parse
Where do these arrows/references come from?

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

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**What about ambiguous parses?**

<table>
<thead>
<tr>
<th>DT</th>
<th>PP</th>
<th>VB</th>
<th>NN</th>
<th>NP</th>
<th>IN</th>
<th>NP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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?**

### CKY: some things to think about

<table>
<thead>
<tr>
<th>CNF</th>
<th>Actual grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow VP</td>
<td>S \rightarrow VP</td>
</tr>
<tr>
<td>VP \rightarrow VB NP</td>
<td>VP \rightarrow VB NP</td>
</tr>
<tr>
<td>VP2 \rightarrow VB NP</td>
<td>VP2 \rightarrow VB NP</td>
</tr>
<tr>
<td>NP \rightarrow DT NN</td>
<td>NP \rightarrow DT NN</td>
</tr>
<tr>
<td>NP \rightarrow NN</td>
<td>NP \rightarrow NN</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

**Ideas?**

###Parsing ambiguity

<table>
<thead>
<tr>
<th>S</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How can we decide between these?
A Simple PCFG

Probabilities!

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.7</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.3</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
</tr>
<tr>
<td>V → saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → astronomers</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → ears</td>
<td>0.18</td>
</tr>
<tr>
<td>NP → stars</td>
<td>0.18</td>
</tr>
<tr>
<td>NP → telescope</td>
<td>0.1</td>
</tr>
</tbody>
</table>

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?
- 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 | Chomsky Normal Form
---|---
S → NP VP 0.8 | S → NP VP 0.8
S → Aux NP VP 0.1 | S → Xi VP 0.1
S → VP 0.1 | S → book | include | prefer 0.01 0.004 0.006
S → Verb NP 0.1 | S → VP PP 0.03
S → VP PP 0.03 |
NP → Pronoun 0.2 | NP → I | he | she | me 0.1 0.02 0.02 0.06
NP → Proper-Noun 0.2 | NP → Houston | NWA 0.15 .04
NP → Det-Nominal 0.6 | NP → Det Nominal 0.6
Nominal → Noun 0.3 | Nominal → book | flight | meal | money 0.03 0.05 0.06 0.06
Nominal → Nominal Noun 0.2 | Nominal → Nominal Noun 0.2
Nominal → Nominal PP 0.5 | Nominal → Nominal PP 0.5
VP → Verb NP 0.5 | VP → book | include | prefer 0.1 0.04 0.06
VP → VP PP 0.3 | VP → VP PP 0.3
PP → Prep NP 1.0 | PP → Prep NP 1.0

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

What is the probability of the NP?

Book       the        flight    through Houston
S :.01, VP:.1, Verb:.5
Nominal:.03
Noun:.1
Det:.6
Nominal:.15
Noun:.5
None
VP → Verb NP 0.5

What is the probability of the VP?
Probabilistic CKY Parser

Book       the        flight    through  Houston

S :.05, VP:.5, Verb:.5, Nominal:.15, Noun:.5
Det:.65
Nominal:.15, Noun:.5
None
NP:.6*.6*.15
= .054
VP:.5*.5*.054
= .0135
S:.05*.5*.054
= .00135

VP → Verb NP  0.5
Probabilistic CKY Parser

<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through Houston</th>
</tr>
</thead>
</table>

\[ S \rightarrow VP \quad 0.03 \]
\[ S \rightarrow Verb \quad 0.05 \]

Which parse do we pick?

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