CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Two views of linguistic structure:
1. Constituency (phrase structure)
   • Phrase structure organizes words into nested constituents.
     • Fed raises interest rates
Two views of linguistic structure:
1. Constituency (phrase structure)

• Phrase structure organizes words into nested constituents.
Two views of linguistic structure:  
1. Constituency (phrase structure)

- Phrase structure organizes words into nested constituents.

- How do we know what is a constituent? (Not that linguists don’t argue about some cases.)
  - Distribution: a constituent behaves as a unit that can appear in different places:
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about

- Substitution/expansion/pronoun:
  - I sat [on the box/right on top of the box/there].
Analysts said Mr. Stronach wants to resume a more influential role in running the company.
Headed phrase structure

• Context-free grammar
• VP → ... VB* ...
• NP → ... NN* ...
• ADJP → ... JJ* ...
• ADVP → ... RB* ...

• SBAR(Q) → S | SINV | SQ → ... NP VP ...

• Plus minor phrase types:
  • QP (quantifier phrase in NP), CONJP (multi word constructions: as well as), INTJ (interjections), etc.
Two views of linguistic structure:
2. Dependency structure

• Dependency structure shows which words depend on (modify or are arguments of) which other words.

*The boy put the tortoise on the rug*
Two views of linguistic structure:

2. Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.
Phrase Chunking

• Find all non-recursive noun phrases (NPs) and verb phrases (VPs) in a sentence.
  • [NP I] [VP ate] [NP the spaghetti] [PP with] [NP meatballs].
  • [NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only 1.8 billion] [PP in] [NP September]
Phrase Chunking as Sequence Labeling

• Tag individual words with one of 3 tags
  • B (Begin) word starts new target phrase
  • I (Inside) word is part of target phrase but not the first word
  • O (Other) word is not part of target phrase

• Sample for NP chunking
  • He reckons the current account deficit will narrow to only 1.8 billion in September.

Begin    Inside    Other
Evaluating Chunking

Per token accuracy does not evaluate finding correct full chunks. Instead use:

Precision = \frac{\text{Number of correct chunks found}}{\text{Total number of chunks found}}

Recall = \frac{\text{Number of correct chunks found}}{\text{Total number of actual chunks}}

Take harmonic mean to produce a single evaluation metric called F measure.

\[ F_1 = \frac{1}{\left( \frac{1}{P} + \frac{1}{R} \right)/2} = \frac{2PR}{P + R} \]
Current Chunking Results

• Best system for NP chunking: $F_1=96\%$
• Typical results for finding range of chunk types (CONLL 2000 shared task: NP, VP, PP, ADV, SBAR, ADJP) is $F_1=92−94\%$
A Brief Parsing History
Pre 1990 ("Classical") NLP Parsing

• Wrote symbolic grammar (CFG or often richer) and lexicon

  S → NP VP
  NP → (DT) NN
  NP → NN NNS
  NP → NNP
  VP → V NP

  NN → interest
  NNS → rates
  NNS → raises
  VBP → interest
  VBZ → rates

• Used grammar systems to prove parses from words

• This scaled very badly. For sentence:

  Fed raises interest rates 0.5% in effort to control inflation

  • Minimal grammar: 36 parses
  • Simple 10 rule grammar (with a lexicon): 592 parses
  • Real-size broad-coverage grammar: millions of parses
Syntactic Parsing

• Produce the correct syntactic parse tree for a sentence.
Classical NLP Parsing:
The problem and its solution

• Adding constraints to grammars to limit unlikely/weird parses for sentences
  • But the attempt make the grammars not robust
    • In traditional systems, commonly 30% of sentences in even an edited text would have no parse.

• A less constrained grammar can parse more sentences
  • But simple sentences end up with ever more parses with no way to choose between them

• We need mechanisms that allow us to find the most likely parse(s) for a sentence
  • Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)
The rise of annotated data: The Penn Treebank

[Marcus et al. 1993, Computational Linguistics]
The rise of annotated data

• Starting off, building a treebank seems a lot slower and less useful than building a grammar

• But a treebank gives us many things
  • Reusability of the labor
    • Many parsers, POS taggers, etc.
    • Valuable resource for linguistics
  • Broad coverage
  • Frequencies and distributional information
  • A way to evaluate systems
Two problems to solve for parsing:
1. Repeated work...
Two problems to solve for parsing:
1. Repeated work...
Two problems to solve for parsing:
2. Choosing the correct parse

- How do we work out the correct attachment:
  - She saw the man with a telescope
  - Words are good predictors of attachment, even absent full understanding
    - Moscow **sent** more than 100,000 soldiers **into** Afghanistan ...
    - Sydney Water breached an **agreement with** NSW Health ...
- Our statistical parsers will try to exploit such statistics.
Statistical parsing applications

Statistical parsers are now robust and widely used in larger NLP applications:

• High precision question answering [Pasca and Harabagiu SIGIR 2001]
• Improving biological named entity finding [Finkel et al. JNLPBA 2004]
• Syntactically based sentence compression [Lin and Wilbur 2007]
• Extracting opinions about products [Bloom et al. NAACL 2007]
• Improved interaction in computer games [Gorniak and Roy 2005]
• Helping linguists find data [Resnik et al. BLS 2005]
• Source sentence analysis for machine translation [Xu et al. 2009]
• Relation extraction systems [Fundel et al. Bioinformatics 2006]
(Probabilistic) Context-Free Grammars

- CFG
- PCFG
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Phrase structure grammars
= context-free grammars (CFGs)

• $G = (T, N, S, R)$
  • $T$ is a set of terminal symbols
  • $N$ is a set of nonterminal symbols
  • $S$ is the start symbol ($S \in N$)
  • $R$ is a set of rules/productions of the form $X \rightarrow \gamma$
    • $X \in N$ and $\gamma \in (N \cup T)^*$

• A grammar $G$ generates a language $L$. 
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.
Phrase structure grammars in NLP

- \( G = (T, C, N, S, L, R) \)
  - \( T \) is a set of terminal symbols
  - \( C \) is a set of preterminal symbols
  - \( N \) is a set of nonterminal symbols
  - \( S \) is the start symbol (\( S \in N \))
  - \( L \) is the lexicon, a set of items of the form \( X \rightarrow x \)
    - \( X \in C \) and \( x \in T \)
  - \( R \) is the grammar, a set of items of the form \( X \rightarrow \gamma \)
    - \( X \in N \) and \( \gamma \in (N \cup C)^* \)
- By usual convention, \( S \) is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write \( e \) for an empty sequence, rather than nothing
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Probabilistic – or stochastic – context-free grammars (PCFGs)

- \( G = (T, N, S, R, P) \)
  - \( T \) is a set of terminal symbols
  - \( N \) is a set of nonterminal symbols
  - \( S \) is the start symbol (\( S \in N \))
  - \( R \) is a set of rules/productions of the form \( X \rightarrow \gamma \)
  - \( P \) is a probability function
    - \( P: R \rightarrow [0,1] \)
    - \( \forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1 \)

- A grammar \( G \) generates a language model \( L \).
A PCFG

S → NP VP 1.0
VP → V NP 0.6
VP → V NP PP 0.4
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0

N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0

[With empty NP removed so less ambiguous]
The probability of trees and strings

• $P(t)$ – The probability of a tree $t$ is the product of the probabilities of the rules used to generate it.
• $P(s)$ – The probability of the string $s$ is the sum of the probabilities of the trees which have that string as their yield

\[
P(s) = \sum_t P(s, t) \quad \text{where } t \text{ is a parse of } s
\]

\[
= \sum_t P(t)
\]
$$t_1:$$

```
NP_{0.7} -> N_{0.5} people, V_{0.6} fish, NP_{0.7}, PP_{1.0} tanks with N_{0.1} rods
```

$$t_2:$$

```
NP_{0.7} -> N_{0.5} people, V_{0.6} fish, NP_{0.7}, PP_{1.0} tanks with N_{0.1} rods
```
Tree and String Probabilities

- \( s = \text{people fish tanks with rods} \)
- \( P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 \)
  \( = 0.0008232 \)
- \( P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 \)
  \( = 0.00024696 \)
- \( P(s) = P(t_1) + P(t_2) \)
  \( = 0.0008232 + 0.00024696 \)
  \( = 0.00107016 \)
Chomsky Normal Form

• All rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$
  • $X, Y, Z \in N$ and $w \in T$

• A transformation to this form doesn’t change the generative capacity of a CFG
  • That is, it recognizes the same language
    • But maybe with different trees

• Empties and unaries are removed recursively

• $n$-ary rules are divided by introducing new nonterminals ($n > 2$)
A phrase structure grammar

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Example Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td></td>
</tr>
<tr>
<td>VP → V NP</td>
<td></td>
</tr>
<tr>
<td>VP → V NP PP</td>
<td></td>
</tr>
<tr>
<td>NP → NP NP</td>
<td></td>
</tr>
<tr>
<td>NP → NP PP</td>
<td></td>
</tr>
<tr>
<td>NP → N</td>
<td></td>
</tr>
<tr>
<td>NP → e</td>
<td></td>
</tr>
<tr>
<td>PP → P NP</td>
<td></td>
</tr>
<tr>
<td>N → people</td>
<td></td>
</tr>
<tr>
<td>N → fish</td>
<td></td>
</tr>
<tr>
<td>N → tanks</td>
<td></td>
</tr>
<tr>
<td>N → rods</td>
<td></td>
</tr>
<tr>
<td>V → people</td>
<td></td>
</tr>
<tr>
<td>V → fish</td>
<td></td>
</tr>
<tr>
<td>V → tanks</td>
<td></td>
</tr>
<tr>
<td>P → with</td>
<td></td>
</tr>
</tbody>
</table>
Chomsky Normal Form steps

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
S → people
V → fish
S → fish
V → tanks
S → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → people
NP → fish
NP → tanks
NP → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
PP → with
Chomsky Normal Form steps

$S \rightarrow NP \ VP$
$VP \rightarrow V \ NP$
$S \rightarrow V \ NP$
$VP \rightarrow V \ @VP_V$
$@VP_V \rightarrow NP \ PP$
$S \rightarrow V \ @S_V$
$@S_V \rightarrow NP \ PP$
$VP \rightarrow V \ PP$
$S \rightarrow V \ PP$
$NP \rightarrow NP \ NP$
$NP \rightarrow NP \ PP$
$NP \rightarrow P \ NP$
$PP \rightarrow P \ NP$

$NP \rightarrow people$
$NP \rightarrow fish$
$NP \rightarrow tanks$
$NP \rightarrow rods$
$V \rightarrow people$
$S \rightarrow people$
$VP \rightarrow people$
$V \rightarrow fish$
$S \rightarrow fish$
$VP \rightarrow fish$
$V \rightarrow tanks$
$S \rightarrow tanks$
$VP \rightarrow tanks$
$P \rightarrow with$
$PP \rightarrow with$
Chomsky Normal Form

• You should think of this as a transformation for efficient parsing

• **Binarization** is crucial for cubic time CFG parsing

• The rest isn’t necessary; it just makes the algorithms cleaner and a bit quicker
An example: before binarization…

```
ROOT
   /
  /  
S     
   /  
NP    VP
   /    /
 N    NP
  /    /
people fish tanks with rods
```
After binarization on VP

The diagram represents the sentence structure after binarization. The sentence is: "people with tanks fish with rods."
Parsing

• Given a string of terminals and a CFG, determine if the string can be generated by the CFG.
  • Also return a parse tree for the string
  • Also return all possible parse trees for the string

• Must search space of derivations for one that derives the given string.
  • **Top-Down Parsing**: Start searching space of derivations for the start symbol.
  • **Bottom-up Parsing**: Start search space of reverse derivations from the terminal symbols in the string.
Parsing Example

book that flight

S
  VP
    Verb
      book
    NP
      Det
        that
      Nominal
        Noun
          flight
Top Down Parsing

S

NP  VP

Pronoun
Top Down Parsing

```
S
  NP  VP
    
  Pronoun
    
X
  book
```
Top Down Parsing

```
S
  NP  VP
    
  ProperNoun
```
Top DownParsing

S

NP VP

ProperNoun

X

book
Top Down Parsing
Top Down Parsing

```
S
  NP VP
    Det Nominal
      book
```
Top Down Parsing

S
/     \
Aux   NP   VP
Top Down Parsing

[Diagram of a tree structure with nodes labeled S, Aux, NP, VP, and book.]
Top Down Parsing

S
| VP
Top Down Parsing

S
| /
| VP
| |
| Verb
Top Down Parsing

```
S
/ |
VP
/ |
Verb
/ |
book
```
Top Down Parsing

S
  /
VP
  /
Verb
    /
book
    /
X
    /
that
Top Down Parsing
Top Down Parsing

S
/
VP
/
Verb NP
/
book
Top Down Parsing

```
S
  /\  
 VP
  /\  
 Verb  NP
     /\  
    book Pronoun
```
Top Down Parsing

S
\|-- VP
   \|-- Verb
       \|-- book
   \|-- NP
       \|-- Pronoun
           \|-- that
Top Down Parsing

```
S
  |
  VP
   |
  Verb NP
     |
  book ProperNoun
```
Top Down Parsing

```
S
├── VP
│   ├── Verb book
│   └── NP ProperNoun
|       └── that
```
Top Down Parsing

S
  /  
VP
  /  
Verb NP
    /  
book Det Nominal
Top Down Parsing

S
  /    
VP
   /   
Verb  NP
    /    
  book  Det  Nominal
     /    
    that
Top Down Parsing
Top Down Parsing

```
S
  VP
    Verb  NP
    book  Det  Nominal
    that  Noun
    flight
```
Bottom Up Parsing

book  that  flight
Bottom Up Parsing

Noun

book that flight
Bottom Up Parsing

Nominal
  Noun
    book that flight
Bottom Up Parsing

```
Nominal
  /\     /
Nominal Noun
  /
Noun
  /
book  that  flight
```
Bottom Up Parsing

Nominal

  Nominal

    Noun

      book

    Noun

      flight

    that
Bottom Up Parsing
Bottom Up Parsing

```
Nominal
   /   
Nominal PP
       /     
Noun Det
   book that flight
```
Bottom Up Parsing

```
Nominal
  Nominal     PP
    Noun     Det  Nominal
       book   that  flight
```
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing

Nominal
  /    
Nominal  PP
     /   
Noun    Det  Nominal
   /     /    
book  that  Noun
    /     /
   flight
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing

```
NP
  /   \\  
Det  Nominal
 /    /
that  Noun
      /   \
      flight
   /     \
book
```

Verbs: book

Detectors: that

Nominals: flight
Bottom Up Parsing

VP
  Verb
    book

NP
  Det
    that
  Nominal
    Noun
      flight
Bottom Up Parsing
Bottom Up Parsing

S
  VP
     Verb
       book

X
  NP
     Det
       that
     Nominal
       Noun
         flight
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing

S
  /  
VP  NP
  /  
Verb  Det  Nominal
  /  
book  that  Noun
       
flight
Top Down vs. Bottom Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.
Dynamic Programming Parsing

• To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.

• Caching (memorizing) is critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.

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

PCFG

<table>
<thead>
<tr>
<th>Rule Prob $\theta_i$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP\ VP$</td>
<td>$\theta_0$</td>
</tr>
<tr>
<td>$NP \rightarrow NP\ NP$</td>
<td>$\theta_1$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$N \rightarrow fish$</td>
<td>$\theta_{42}$</td>
</tr>
<tr>
<td>$N \rightarrow people$</td>
<td>$\theta_{43}$</td>
</tr>
<tr>
<td>$V \rightarrow fish$</td>
<td>$\theta_{44}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Cocke-Kasami-Younger (CKY) Constituency Parsing

fish people fish tanks
Viterbi (Max) Scores

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
Extended CKY parsing

• Unaries can be incorporated into the algorithm
  • Messy, but doesn’t increase algorithmic complexity

• Empties can be incorporated
  • Doesn’t increase complexity; essentially like unaries

• Binarization is *vital*
  • Without binarization, you don’t get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar
The CKY algorithm (1960/1965) ... extended to unaries

function CKY(words, grammar) returns [most_probable_parse, prob]
    score = new double[#(words)+1][#(words)+1][#(nonterms)]
    back = new Pair[#(words)+1][#(words)+1][#(nonterms)]
    for i=0; i<#(words); i++
        for A in nonterms
            if A -> words[i] in grammar
                score[i][i+1][A] = P(A -> words[i])
        //handle unaries
        boolean added = true
        while added
            added = false
            for A, B in nonterms
                if score[i][i+1][B] > 0 && A->B in grammar
                    prob = P(A->B)*score[i][i+1][B]
                    if prob > score[i][i+1][A]
                        score[i][i+1][A] = prob
                        back[i][i+1][A] = B
                        added = true
The CKY algorithm (1960/1965) … extended to unaries

for span = 2 to #(words)
  for begin = 0 to #(words) - span
    end = begin + span
    for split = begin+1 to end-1
      for A, B, C in nonterms
        prob = score[begin][split][B] * score[split][end][C] * P(A->BC)
        if prob > score[begin][end][A]
          score[begin][end][A] = prob
          back[begin][end][A] = new Triple(split, B, C)

// handle unaries
boolean added = true
while added
  added = false
  for A, B in nonterms
    prob = P(A->B) * score[begin][end][B];
    if prob > score[begin][end][A]
      score[begin][end][A] = prob
      back[begin][end][A] = B
      added = true
return buildTree(score, back)