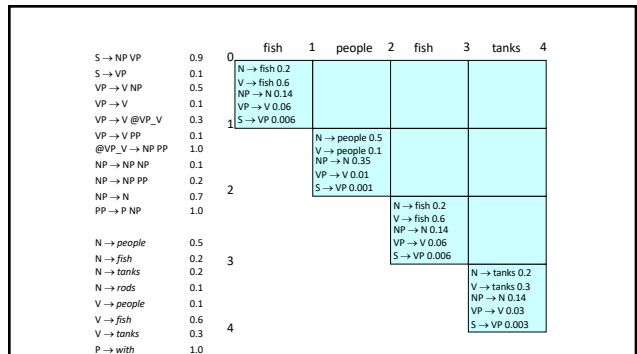
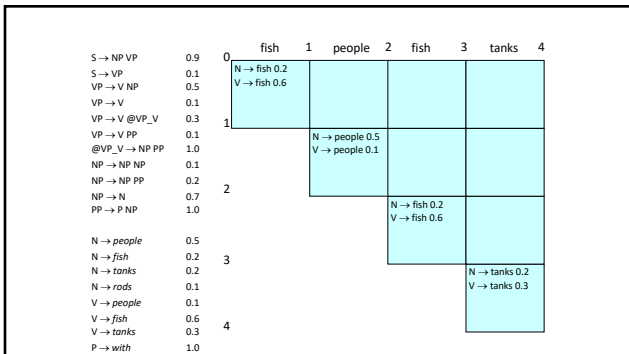
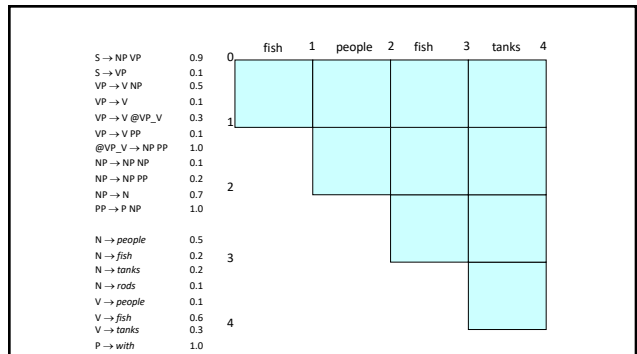
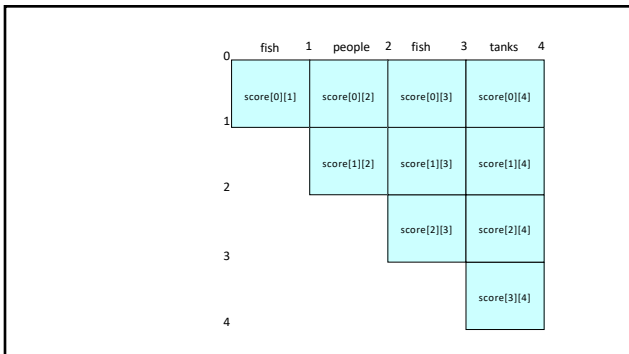


CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang  
 College of Computer and Information Science  
 Northeastern University  
 Webpage: [www.ccs.neu.edu/home/luwang](http://www.ccs.neu.edu/home/luwang)

The grammar

S → NP VP	0.9	N → people	0.5
S → VP	0.1	N → fish	0.2
VP → V NP	0.5	N → tanks	0.2
VP → V	0.1	N → rods	0.1
VP → V @VP_V	0.3	V → people	0.1
VP → V PP	0.1	V → fish	0.6
@VP_V → NP PP	1.0	V → tanks	0.3
NP → NP NP	0.1	P → with	1.0
NP → NP PP	0.2		
NP → N	0.7		
PP → P NP	1.0		





S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1		N → fish 0.2							
VP → V NP	0.5		V → fish 0.6							
VP → V	0.1									
VP → V @VP_V	0.3									
VP → V PP	0.1	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									
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									

// handle unaries  
boolean added = true  
while added  
added = false  
for A, B in nonterms  
if score[(i+1)B] > 0 && A->B in grammar  
prob = P(A->B) \* score[(i+1)B]  
if (prob > score[(i)A])  
score[(i+1)A] = prob  
back[begin](end) = B  
added = true

S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1		N → fish 0.2							
VP → V NP	0.5		V → fish 0.6							
VP → V	0.1		NP → N 0.14							
VP → V @VP_V	0.3		VP → V 0.06							
VP → V PP	0.1	1	S → VP 0.006							
@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									
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									

prob = score[begin](split[B]\*score[split](end)[C]\*P(A->B)]  
if (prob > score[begin](end)[A])  
score[begin](end)[A] = prob  
back[begin](end)[A] = new Triple(split,B,C)

S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1		N → fish 0.2							
VP → V NP	0.5		V → fish 0.6							
VP → V	0.1		NP → N 0.14							
VP → V @VP_V	0.3		VP → V 0.06							
VP → V PP	0.1	1	S → VP 0.006							
@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									
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									

//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

S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1		N → fish 0.2							
VP → V NP	0.5		V → fish 0.6							
VP → V	0.1		NP → N 0.14							
VP → V @VP_V	0.3		VP → V 0.06							
VP → V PP	0.1	1	S → VP 0.006							
@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									
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									

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

S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1		N → fish 0.2							
VP → V NP	0.5		V → fish 0.6							
VP → V	0.1		NP → N 0.14							
VP → V @VP_V	0.3		VP → V 0.06							
VP → V PP	0.1	1	S → VP 0.006							
@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									
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									

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

S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1		N → fish 0.2							
VP → V NP	0.5		V → fish 0.6							
VP → V	0.1		NP → N 0.14							
VP → V @VP_V	0.3		VP → V 0.06							
VP → V PP	0.1	1	S → VP 0.006							
@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									
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									

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

S → NP VP	0.9	0	fish	1	people	2	fish	3	tanks	4
S → VP	0.1	N → fish 0.2	NP → NP NP	0.0049	NP → NP NP	0.0000686	NP → NP NP	0.000009604		
VP → V NP	0.5	V → fish 0.6	VP → V NP	0.00147	VP → V NP	0.0000358	VP → V NP	0.000000000		
VP → V	0.1	NP → N 0.14	S → VP	0.105	S → VP	0.000882	S → VP	0.00018522		
VP → V @VP_V	0.3	S → VP 0.006	N → people 0.5	NP → NP NP	0.0049	NP → NP NP	0.0000686	NP → NP NP		
VP → V PP	0.1		V → people 0.3	VP → V NP	0.007	VP → V NP	0.0000998	VP → V NP		
@VP_V → NP PP	1.0		NP → N 0.35	VP → V 0.01	0.0189	VP → V 0.06	0.0323	VP → V 0.03		
NP → NP NP	0.1		S → VP 0.001	N → fish 0.2	NP → NP NP	0.00186	NP → NP NP	0.042		
NP → NP PP	0.2			V → fish 0.6	NP → N 0.14	NP → V NP	0.042	S → VP		
NP → N	0.7			NP → N 0.14	VP → V 0.06	S → VP	0.0042			
PP → P NP	1.0			S → VP 0.006	N → tanks 0.2	V → tanks 0.3				
					V → fish 0.6	NP → N 0.14	VP → V 0.03			
N → people	0.5				S → VP 0.006	S → VP 0.003				
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									

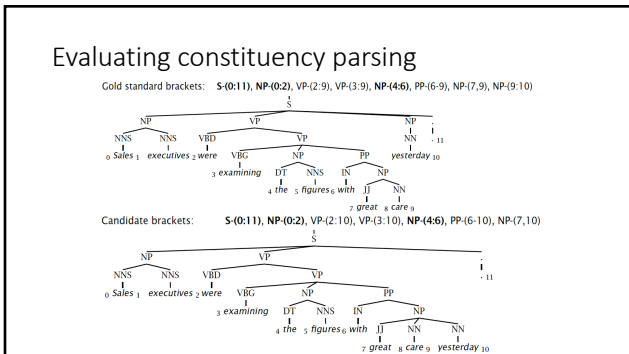
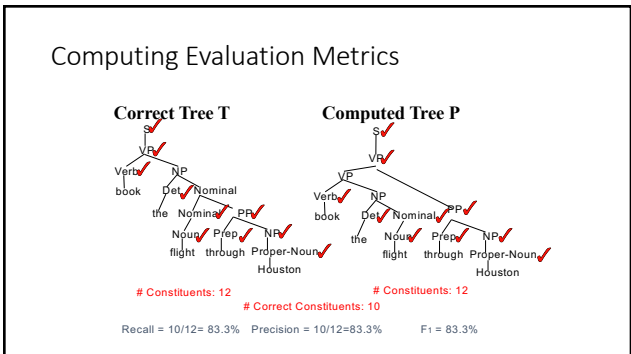
Call buildThreeScore, back to get the best parse

### Extended CKY parsing

- CKY parsing is usually done **after binarization**
  - 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

### Where to learn the probabilities: Treebanks

- **English Penn Treebank:** Standard corpus for testing syntactic parsing consists of 1.2 M words of text from the Wall Street Journal (WSJ).
- Typical to train on about 40,000 parsed sentences and test on an additional standard disjoint test set of 2,416 sentences.
- **Chinese Penn Treebank:** 100K words from the Xinhua news service.
- Other corpora existing in many languages, see the Wikipedia article "Treebank"



### Evaluating constituency parsing

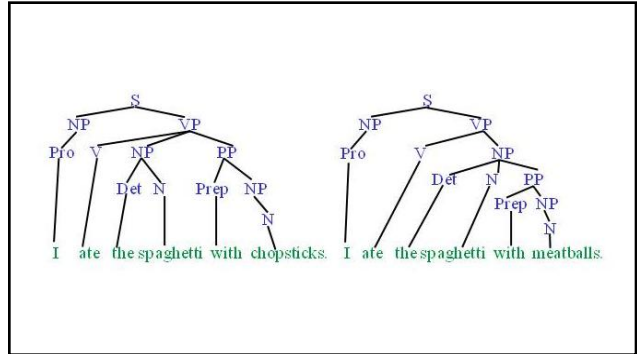
**Gold standard brackets:** S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7:9), NP-(9:10)

**Candidate brackets:** S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6:10), NP-(7,10)

Labeled Precision	3/7 = 42.9%
Labeled Recall	3/8 = 37.5%
LP/LR F1	40.0%
POS Tagging Accuracy	11/11 = 100.0%

### How good are PCFGs?

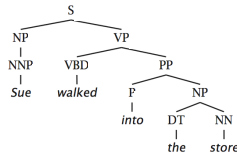
- Penn WSJ parsing accuracy: about 73% LP/LR F1 with feature-based models (state-of-the-art neural model is 91-92% F1)
- Robust
  - Usually admit everything, but with low probability
- Partial solution for grammar ambiguity
  - A PCFG gives some idea of the plausibility of a parse



### (Head) Lexicalization of PCFGs

[Magerman 1995, Collins 1997; Charniak 1997]

- The head word of a phrase gives a good representation of the phrase's structure and meaning (*head words are decided by rules*)
- Puts the properties of words back into a PCFG



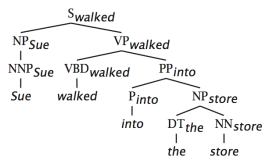
### Head Words

- Syntactic phrases usually have a word in them that is most "central" to the phrase.
- Linguists have defined the concept of a lexical **head** of a phrase.
- Simple rules can identify the head of any phrase by percolating head words up the parse tree.
  - Head of a VP is the main verb
  - Head of an NP is the main noun
  - Head of a PP is the preposition
  - Head of a sentence is the head of its VP

### (Head) Lexicalization of PCFGs

[Magerman 1995, Collins 1997; Charniak 1997]

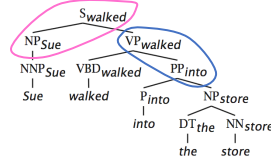
- The head word of a phrase gives a good representation of the phrase's structure and meaning
- Puts the properties of words back into a PCFG



### (Head) Lexicalization of PCFGs

[Magerman 1995, Collins 1997; Charniak 1997]

- The head word of a phrase gives a good representation of the phrase's structure and meaning
- Puts the properties of words back into a PCFG



## (Head) Lexicalization of PCFGs

[Magerman 1995, Collins 1997; Charniak 1997]

- Word-to-word affinities are useful for certain ambiguities
  - PP attachment is now (partly) captured in a local PCFG rule.



## Lexicalized parsing was seen as *the* parsing breakthrough of the late 1990s

- Eugene Charniak, 2000 JHU workshop: "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter:

- $p(\text{VP} \rightarrow \text{V NP NP}) = 0.00151$
- $p(\text{VP} \rightarrow \text{V NP NP} \mid \text{said}) = 0.00001$
- $p(\text{VP} \rightarrow \text{V NP NP} \mid \text{gave}) = 0.01980$  "

- Michael Collins, 2003 COLT tutorial: "Lexicalized Probabilistic Context-Free Grammars ... perform vastly better than PCFGs (88% vs. 73% accuracy)"

## Lexicalization models argument selection by sharpening rule expansion probabilities

- The probability of different verbal complement frames (i.e., "subcategorizations") depends on the verb:

Local Tree	come	take	think	want
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	<b>32.1%</b>	0.2%	13.9%
VP → V PP	<b>34.5%</b>	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	<b>73.0%</b>	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

## Human Parsing

- Computational parsers can be used to predict human reading time as measured by tracking the time taken to read each word in a sentence.
- Psycholinguistic studies show that words that are more probable given the preceding lexical and syntactic context are read faster.
  - John put the dog in the pen with a **lock**.
  - John put the dog in the pen with a **bone** in the car.
  - John liked the dog in the pen with a **bone**.
- Modeling these effects requires an **incremental** statistical parser that incorporates one word at a time into a continuously growing parse tree.

## Garden Path Sentences

- People are confused by sentences that seem to have a particular syntactic structure but then suddenly violate this structure, so the listener is "lead down the garden path".
  - The horse raced past the barn fell.
    - vs. The horse raced past the barn broke his leg.
  - The complex houses married students.
  - The old man the sea.
  - While Anna dressed the baby spit up on the bed.
- Incremental computational parsers can try to predict and explain the problems encountered parsing such sentences.

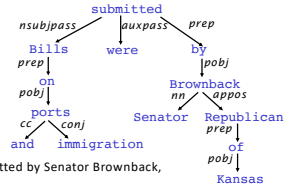
## Center Embedding

- Nested expressions are hard for humans to process beyond 1 or 2 levels of nesting.
  - The rat the cat chased died.
  - The rat the cat the dog bit chased died.
  - The rat the cat the dog the boy owned bit chased died.
- Requires remembering and popping incomplete constituents from a stack and strains human short-term memory.
- Equivalent "tail embedded" (tail recursive) versions are easier to understand since no stack is required.
  - The boy owned a dog that bit a cat that chased a rat that died.

### Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly **typed** with the name of grammatical relations (subject, prepositional object, apposition, etc.)



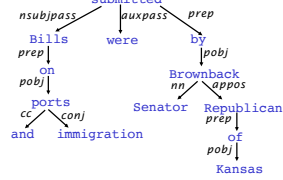
Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas.

### Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)

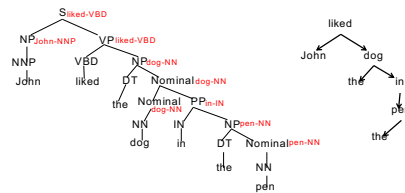


### Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, ...) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/...
  - The head of a Verb Phrase is a verb/modal/...
- The head rules can be used to extract a dependency parse from a CFG parse

### Dependency Graph from Parse Tree

- Can convert a phrase structure parse to a dependency tree by making the head of each non-head child of a node depend on the head of the head child.



### Methods of Dependency Parsing

1. **Dynamic programming** (like in the CKY algorithm)
  - You can do it similarly to lexicalized PCFG parsing: an  $O(n^3)$  algorithm
  - Eisner (1996) gives a clever algorithm that reduces the complexity to  $O(n^3)$ , by producing parse items with heads at the ends rather than in the middle
2. **Graph algorithms**
  - You create a Maximum Spanning Tree for a sentence
  - McDonald et al.'s (2005) MSTParser scores dependencies independently using a ML classifier (he uses MIRA, for online learning, but it could be MaxEnt)
3. **Constraint Satisfaction**
  - Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
4. **"Deterministic parsing"**
  - Greedy choice of attachments guided by machine learning classifiers
  - MaltParser (Nivre et al. 2008) – discussed in the next segment