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

The grammar

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

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

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	score[0][1]	score[0][2]	score[0][3]	score[0][4]
1		score[1][2]	score[1][3]	score[1][4]
2			score[2][3]	score[2][4]
3				score[3][4]
4				

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
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

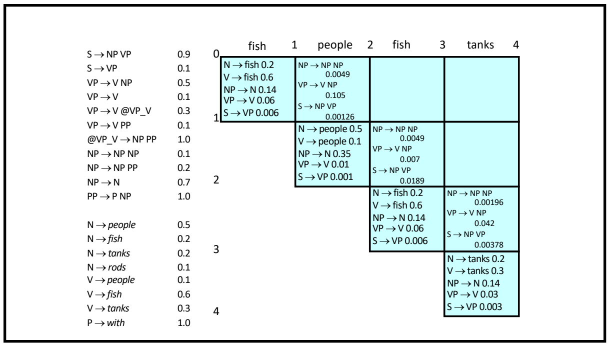
	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0				
1				
2				
3				
4				

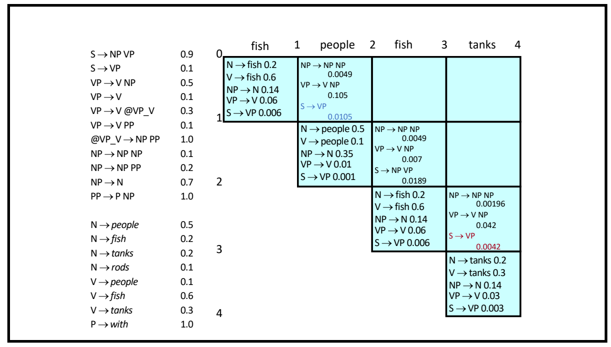
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

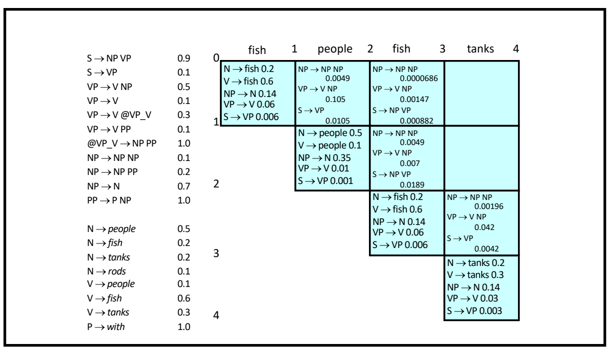
	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6			
1		N → people 0.5 V → people 0.1		
2			N → fish 0.2 V → fish 0.6	
3				N → tanks 0.2 V → tanks 0.3
4				

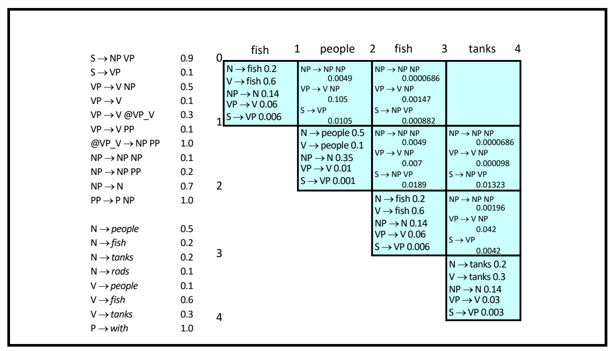
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

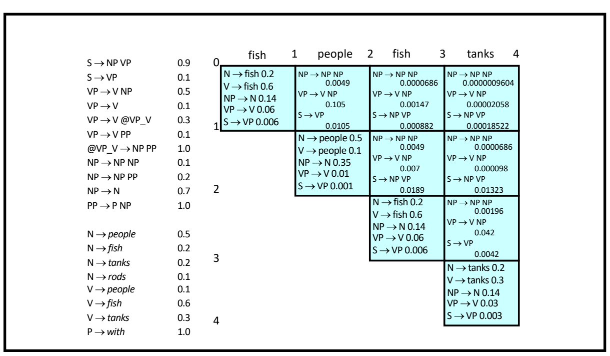
	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006			
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001		
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	
3				N → tanks 0.2 V → tanks 0.3 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

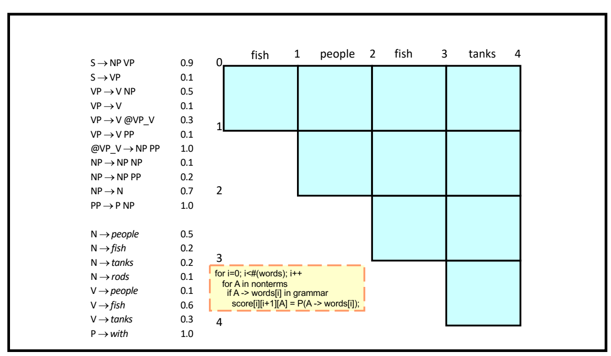

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
fish 1 people 2 fish 3 tanks 4
0 1 2 3 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → NP VP 0.00126
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → NP VP 0.00378
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003

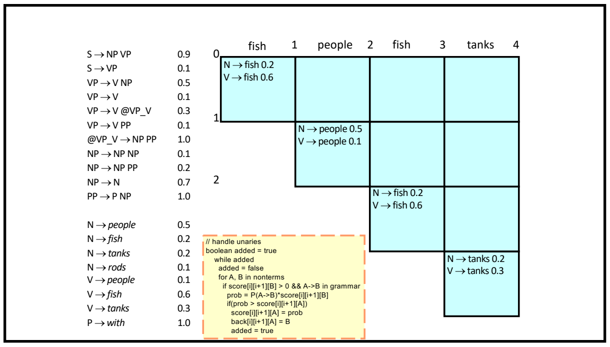

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
fish 1 people 2 fish 3 tanks 4
0 1 2 3 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003

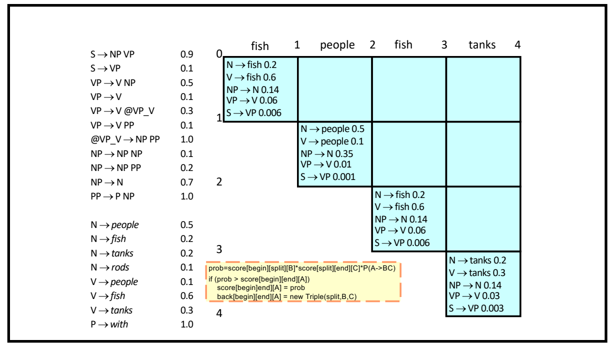

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
fish 1 people 2 fish 3 tanks 4
0 1 2 3 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
NP → NP NP 0.0000686
VP → V NP 0.00147
S → NP VP 0.000882
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003

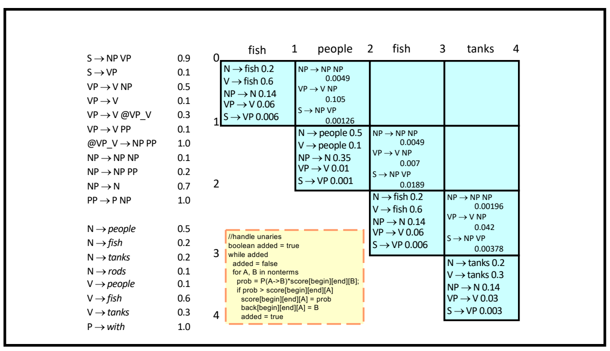

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
fish 1 people 2 fish 3 tanks 4
0 1 2 3 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
NP → NP NP 0.0000686
VP → V NP 0.00147
S → NP VP 0.000882
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
NP → NP NP 0.0000686
VP → V NP 0.000098
S → NP VP 0.01323
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003

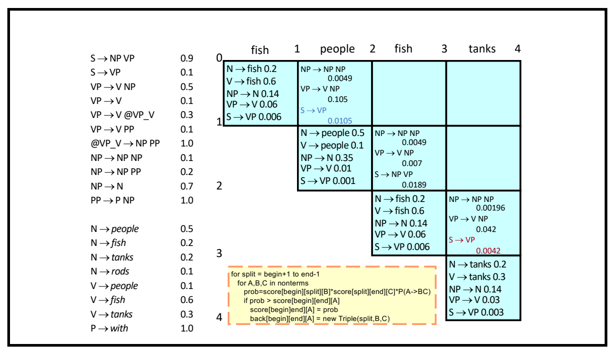

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
fish 1 people 2 fish 3 tanks 4
0 1 2 3 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
NP → NP NP 0.0000686
VP → V NP 0.00147
S → NP VP 0.000882
NP → NP NP 0.0000009604
VP → V NP 0.00002058
S → NP VP 0.00018522
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
NP → NP NP 0.0000686
VP → V NP 0.000098
S → NP VP 0.01323
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003

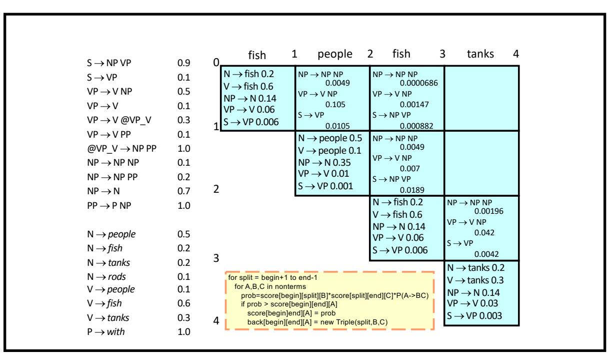

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
fish 1 people 2 fish 3 tanks 4
0 1 2 3 4
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]);

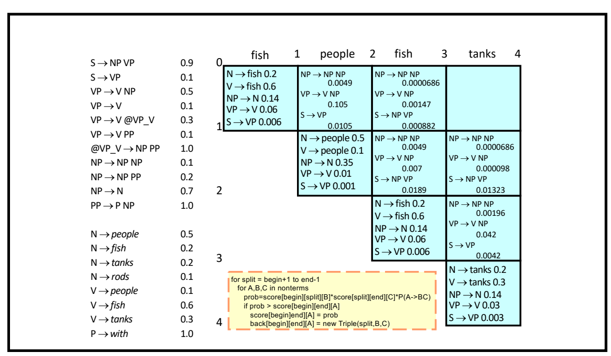

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0
0 fish 1 people 2 fish 3 tanks 4
N → fish 0.2
V → fish 0.6
N → people 0.5
V → people 0.1
N → fish 0.2
V → fish 0.6
N → tanks 0.2
V → tanks 0.3
// 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


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
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
0 fish 1 people 2 fish 3 tanks 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003
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)


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
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
0 fish 1 people 2 fish 3 tanks 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → NP VP 0.00126
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → NP VP 0.00378
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003
//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
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
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
0 fish 1 people 2 fish 3 tanks 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003
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)


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
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
0 fish 1 people 2 fish 3 tanks 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
NP → NP NP 0.0000686
VP → V NP 0.00147
S → NP VP 0.000882
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003
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)


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
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
0 fish 1 people 2 fish 3 tanks 4
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.0049
VP → V NP 0.105
S → VP 0.0105
NP → NP NP 0.0000686
VP → V NP 0.00147
S → NP VP 0.000882
N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001
NP → NP NP 0.0049
VP → V NP 0.007
S → NP VP 0.0189
NP → NP NP 0.0000686
VP → V NP 0.000098
S → NP VP 0.01323
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006
NP → NP NP 0.00196
VP → V NP 0.042
S → VP 0.0042
N → tanks 0.2
V → tanks 0.3
NP → N 0.14
VP → V 0.03
S → VP 0.003
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)

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
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

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.3 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

Call buildTree(score, back) to get the best parse

Extended CKY parsing

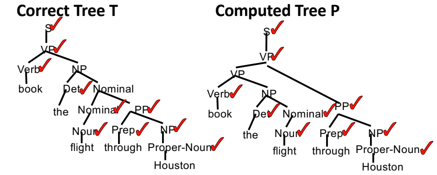
- CKY parsing is usually done after binarization
- Unaries can be incorporated into the algorithm
  - Messy, but doesn't increase algorithmic complexity
- Empties can be incorporated
  - Doesn't increase complexity; essentially like unaries
- Binarization is *vital*
  - Without binarization, you don't get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar

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"

Computing Evaluation Metrics

Correct Tree T **Computed Tree P**



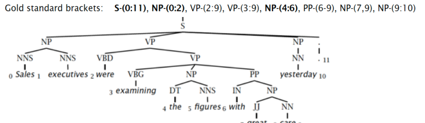
Constituents: 12 # Constituents: 12

Correct Constituents: 10

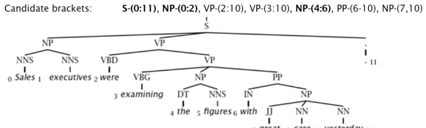
Recall = 10/12= 83.3% Precision = 10/12=83.3% F_1 = 83.3%

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)



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%

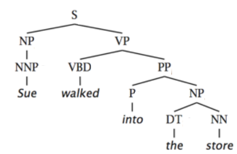
How good are PCFGs?

- Penn WSJ parsing accuracy: about 73% LP/LR F1 (state-of-the-art 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
- Give a probabilistic language model
  - But in the simple case it performs worse than a trigram model
- The problem seems to be that PCFGs lack the lexicalization of a trigram model

(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



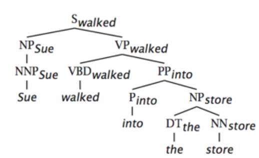
Head Words

- 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]

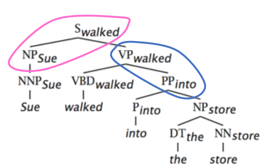
- The head word of a phrase gives a good representation of the phrase's structure and meaning
- Puts the properties of words back into a PCFG



(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.

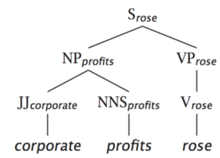
  - Also useful for: coordination scope, verb complement patterns

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

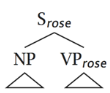
- Eugene Charniak, 2000 JHU workshop: "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter:
  - *p*(VP → V NP NP) = 0.00151
  - *p*(VP → V NP NP | said) = 0.00001
  - *p*(VP → V NP NP | gave) = 0.01980 "
- Michael Collins, 2003 COLT tutorial: "Lexicalized Probabilistic Context-Free Grammars … perform vastly better than PCFGs (88% vs. 73% accuracy)"

Lexicalization of PCFGs: Charniak (1997)

- A very straightforward model of a lexicalized PCFG
- Probabilistic conditioning is "top-down" like a regular PCFG
  - But actual parsing is bottom-up, somewhat like the CKY algorithm we saw



Charniak (1997) example
Probabilities that can be modeled (more info)

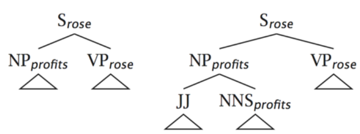


a. h = *profits*; c = NP

b. ph = *rose*; pc = S

c. $P(h|ph, c, pc)$

d. $P(r|h, c, pc)$



Lexicalization models argument selection by sharpening rule expansion probabilities

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

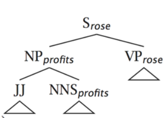
Local Tree	*come*	*take*	*think*	*want*
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	**32.1%**	0.2%	13.9%
VP → V PP	**34.5%**	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	**73.0%**	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

Lexicalization sharpens probabilities: Predicting heads

"Bilexical probabilities"

- P(prices | n-plural) = .013
- P(prices | n-plural, NP) = .013
- P(prices | n-plural, NP, S) = .025
- P(prices | n-plural, NP, S, v-past) = .052
- P(prices | n-plural, NP, S, v-past, fell) = .146

Charniak (1997) linear interpolation/shrinkage



$$\begin{aligned}\hat{P}(h|ph, c, pc) \;=\; & \lambda_1(e)P_{\text{MLE}}(h|ph, c, pc) \\ & +\lambda_2(e)P_{\text{MLE}}(h|C(ph), c, pc) \\ & +\lambda_3(e)P_{\text{MLE}}(h|c, pc) + \lambda_4(e)P_{\text{MLE}}(h|c)\end{aligned}$$

- $\lambda_i(e)$ is here a function of how much one would expect to see a certain occurrence, given the amount of training data, word counts, etc.
- $C(ph)$ is semantic class of parent headword
- Techniques like these for dealing with data sparseness are vital to successful model construction

Charniak (1997) shrinkage example

	$P(\text{prft}\mid\text{rose}, \text{NP}, \text{S})$	$P(\text{corp}\mid\text{prft}, \text{JJ}, \text{NP})$
$P(h\mid ph, c, pc)$	0	0.245
$P(h\mid C(ph), c, pc)$	0.00352	0.0150
$P(h\mid c, pc)$	0.000627	0.00533
$P(h\mid c)$	0.000557	0.00418

- Allows utilization of rich highly conditioned estimates, but smoothes when sufficient data is unavailable
- One can't just use MLEs: one commonly sees previously unseen events, which would have probability 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.

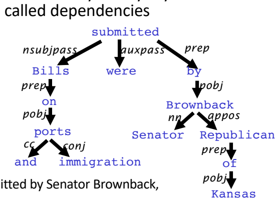
Center Embedding

- 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.)



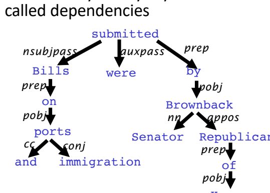
Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas.

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)

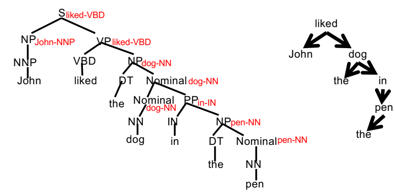


Relation between phrase structure and dependency structure

- 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 $O(n^5)$ 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