CSCI 5832 Natural Language Processing

Jim Martin
Lecture 14

3/11/08 1

Today 3/4

- Parsing
  - CKY again
  - Earley

3/11/08 2

Sample Grammar

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

3/1 3

Dynamic Programming

- DP methods fill tables with partial results and
  - Do not do too much avoidable repeated work
  - Solve exponential problems in polynomial time (sort of)
  - Efficiently store ambiguous structures with shared sub-parts.

3/11/08 4

CKY Parsing

- First we'll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule A -> BC
  - If there is an A in the input then there must be a B followed by a C in the input.
  - If the A spans from i to j in the input then there must be some k st. i<k<j
    - Ie. The B splits from the C someplace.

3/11/08 5

CKY

- So let's build a table so that an A spanning from i to j in the input is placed in cell [i,j] in the table.
- So a non-terminal spanning an entire string will sit in cell [0, n]
- If we build the table bottom up we'll know that the parts of the A must go from i to k and from k to j

3/11/08 6

CKY

- Meaning that for a rule like A -> B C we should look for a B in [i,k] and a C in [k,j].
- In other words, if we think there might be an A spanning i,j in the input… AND
- A -> B C is a rule in the grammar THEN
- There must be a B in [i,k] and a C in [k,j] for some i<k<j

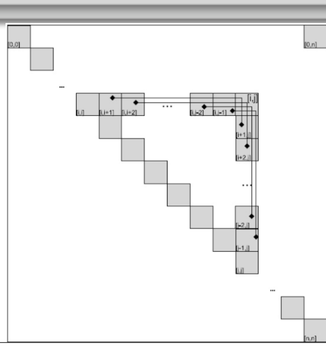
3/11/08 7

CKY

- So to fill the table loop over the cell[i,j] values in some systematic way
  - What constraint should we put on that?
  - For each cell loop over the appropriate k values to search for things to add.

3/11/08 8

CKY Table



3/11/08 9

CKY Algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup \{A \mid A \rightarrow BC \in grammar, B \in table[i,k], C \in table[k,j]\}$

3/11/08 10

CKY Parsing

- Is that really a parser?

3/11/08 11

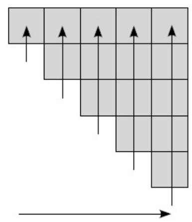
Note

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
  - This assures us that whenever we're filling a cell, the parts needed to fill it are already in the table (to the left and below)

3/11/08 12

Example

Book	the	flight	through	Houston
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]



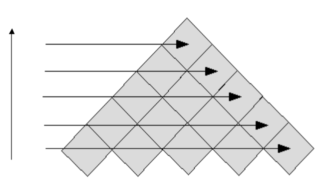
3/11/08 13

Other Ways to Do It?

- Are there any other sensible ways to fill the table that still guarantee that the cells we need are already filled?

3/11/08 14

Other Ways to Do It?



3/11/08 15

Sample Grammar

S → NP VP	*Det → that | this | a*
S → Aux NP VP	*Noun → book | flight | meal | money*
S → VP	*Verb → book | include | prefer*
NP → Pronoun	*Pronoun → I | she | me*
NP → Proper-Noun	*Proper-Noun → Houston | TWA*
NP → Det Nominal	*Aux → does*
Nominal → Noun	*Preposition → from | to | on | near | through*
Nominal → Nominal Noun	
Nominal → Nominal PP	
VP → Verb	
VP → Verb NP	
VP → Verb NP PP	
VP → Verb PP	
VP → VP PP	
PP → Preposition NP	

3/11/08 16

Problem

- What if your grammar isn't binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary… any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - But the resulting derivations (trees) are different.

3/11/08 17

Problem

- More specifically, rules have to be of the form

  A -> B C

  Or

  A -> *w*

  That is, rules can expand to either 2 non-terminals or to a single terminal.

3/11/08 18

Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with length > 2 over several rules. So…

  S -> A B C
  - Turns into

  S -> X C

  X - A B

  Where X is a symbol that doesn't occur anywhere else in the the grammar.

3/11/08 19

CNF Conversion

S → NP VP	S → NP VP
S → Aux NP VP	S → X1 VP
	X1 → Aux NP
S → VP	S → book | include | prefer
	S → Verb NP
	S → X2 PP
	S → Verb PP
	S → VP PP
NP → Pronoun	NP → I | she | me
NP → Proper-Noun	NP → TWA | Houston
NP → Det Nominal	NP → Det Nominal
Nominal → Noun	Nominal → book | flight | meal | money
Nominal → Nominal Noun	Nominal → Nominal Noun
Nominal → Nominal PP	Nominal → Nominal PP
VP → Verb	VP → book | include | prefer
VP → Verb NP	VP → Verb NP
VP → Verb NP PP	VP → X2 PP
	X2 → Verb NP
VP → Verb PP	VP → Verb PP
VP → VP PP	VP → VP PP
PP → Preposition NP	PP → Preposition NP

3/11/08 20

CKY Algorithm

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
     table[j−1, j] ← {A | A → words[j] ∈ grammar }
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
           table[i,j] ← table[i,j] ∪
                        {A | A → BC ∈ grammar,
                             B ∈ table[i,k],
                             C ∈ table[k,j] }
```

3/11/08 21

Example

Book	the	flight	through	Houston
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP, X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3	[1,4]	[1,5]
		]Nominal, Noun [2,3]	[2,4]	[2,5]
			Prep [3,4]	[3,5]
				NP, Proper-Noun [4,5]

1

Filling column 5

Example

Book	the	flight	through	Houston
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	[2,5]
			Prep [3,4]	PP
				NP, Proper-Noun [4,5]

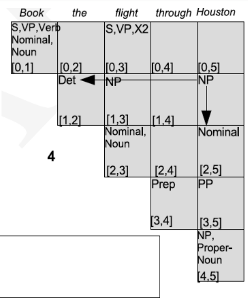
2

Example

Book	the	flight	through	Houston
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3	[1,4]	[1,5]
		]Nominal, Noun [2,3]	[2,4]	Nominal
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

3

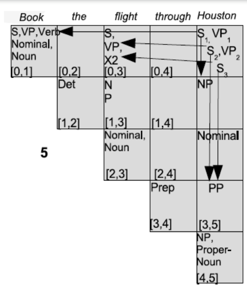
Example



3/11/08

25

Example



3/11/08

26

CKY Notes

- Since it's bottom up, CKY populates the table with a lot of phantom constituents.
  - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  - To avoid this we can switch to a top-down control strategy or
  - We can add some kind of filtering that blocks constituents where they can not happen in a final analysis.

3/11/08

27

Break

- Quiz pushed back to Tues 3/18
  - Schedule
    - Today: CKY and Earley
    - Thursday: Partial parsing, chunking and more on statistical sequence processing
    - Next week: statistical parsing

3/11/08 28

Earley Parsing

- Allows arbitrary CFGs
- Top-down control
- Fills a table in a single sweep over the input words
  - Table is length N+1; N is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

3/11/08 29

States

- The table-entries are called states and are represented with dotted-rules.

S -> · VP	A VP is predicted
NP -> Det · Nominal	An NP is in progress
VP -> V NP ·	A VP has been found

3/11/08 30

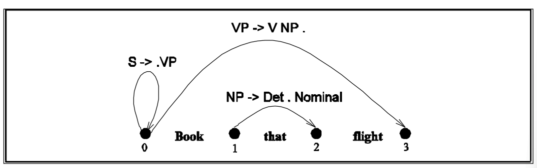
States/Locations

- S -> ● VP [0,0]
- NP -> Det ● Nominal [1,2]
- VP -> V NP ● [0,3]

- A VP is predicted at the start of the sentence
- An NP is in progress; the Det goes from 1 to 2
- A VP has been found starting at 0 and ending at 3

3/11/08 31

Graphically



3/11/08 32

Earley

- As with most dynamic programming approaches, the answer is found by looking in the table in the right place.
- In this case, there should be an S state in the final column that spans from 0 to n and is complete.
- If that’s the case you’re done.
  - S –> α ● [0,n]

3/11/08 33

Earley

- So sweep through the table from 0 to n…
  - New predicted states are created by starting top-down from S
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - New complete states are created in the same way.

3/11/08 34

Earley

- More specifically…
  1. *Predict* all the states you can upfront
  2. Read a word
     1. Extend states based on matches
     2. Generate new predictions
     3. Go to step 2
  3. Look at n to see if you have a winner

3/11/08 35

Earley Code

```
function EARLEY-PARSE(words, grammar) returns chart

 ADDTOCHART((γ → • S, [0,0]), chart[0])
 for i ← from 0 to LENGTH(words) do
   for each state in chart[i] do
     if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
     elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
        SCANNER(state)
     else
        COMPLETER(state)
   end
 end
 return(chart)
```

3/11/08 36

Earley Code

procedure PREDICTOR($(A \rightarrow \alpha \bullet B\,\beta,\ [i,j])$)
  for each $(B \rightarrow \gamma)$ **in** GRAMMAR-RULES-FOR(B, *grammar*) **do**
    ADDTOCHART($(B \rightarrow \bullet\,\gamma,\ [j,j])$, *chart*[$j$])
  end

procedure SCANNER($(A \rightarrow \alpha \bullet B\,\beta,\ [i,j])$)
  if $B \in$ PARTS-OF-SPEECH(*word[j]*) **then**
    ADDTOCHART($(B \rightarrow word[j]\,\bullet,\ [j,j+1])$, *chart[j+1]*)

procedure COMPLETER($(B \rightarrow \gamma\ \bullet,\ [j,k])$)
  for each $(A \rightarrow \alpha \bullet B\,\beta,\ [i,j])$ **in** *chart*[j] **do**
    ADDTOCHART($(A \rightarrow \alpha\,B \bullet \beta,\ [i,k])$, *chart[k]*)
  end

3/11/08 37

Example

- Book that flight
- We should find… an S from 0 to 3 that is a completed state…

3/11/08 38

Example

Chart[0]	S0	$\gamma \rightarrow \bullet\, S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet\, NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet\, Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet\, VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet\, Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet\, Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet\, Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet\, Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet\, Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet\, Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet\, Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet\, VP\ PP$	[0,0]	Predictor

3/11/08 39

Add To Chart

procedure ADDTOCHART(*state, chart-entry*)
  if *state* is not already in *chart-entry* **then**
    PUSH-ON-END(*state, chart-entry*)
  end

Example

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
	S13	*VP* → *Verb* •	[0,1]	Completer
	S14	*VP* → *Verb* • *NP*	[0,1]	Completer
	S15	*VP* → *Verb* • *NP PP*	[0,1]	Completer
	S16	*VP* → *Verb* • *PP*	[0,1]	Completer
	S17	*S* → *VP* •	[0,1]	Completer
	S18	*VP* → *VP* • *PP*	[0,1]	Completer
	S19	*NP* → • *Pronoun*	[1,1]	Predictor
	S20	*NP* → • *Proper-Noun*	[1,1]	Predictor
	S21	*NP* → • *Det Nominal*	[1,1]	Predictor
	S22	*PP* → • *Prep NP*	[1,1]	Predictor

Example

Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
	S24	*NP* → *Det* • *Nominal*	[1,2]	Completer
	S25	*Nominal* → • *Noun*	[2,2]	Predictor
	S26	*Nominal* → • *Nominal Noun*	[2,2]	Predictor
	S27	*Nominal* → • *Nominal PP*	[2,2]	Predictor
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	Completer
	S30	*NP* → *Det Nominal* •	[1,3]	Completer
	S31	*Nominal* → *Nominal* • *Noun*	[2,3]	Completer
	S32	*Nominal* → *Nominal* • *PP*	[2,3]	Completer
	S33	*VP* → *Verb NP* •	[0,3]	Completer
	S34	*VP* → *Verb NP* • *PP*	[0,3]	Completer
	S35	*PP* → • *Prep NP*	[3,3]	Predictor
	S36	*S* → *VP* •	[0,3]	Completer
	S37	*VP* → *VP* • *PP*	[0,3]	Completer

Efficiency

- For such a simple example, there seems to be a lot of useless stuff in there.
- Why?
  - It's predicting things that aren't consistent with the input
  - That's the flipside to the CKY problem.

3/11/08 43

Details

- As with CKY that isn't a parser until we add the backpointers so that each state knows where it came from.

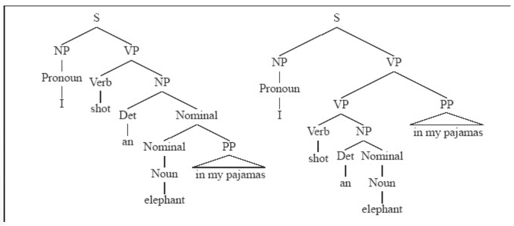
3/11/08 44

Back to Ambiguity

- Did we solve it?

3/11/08 45

Ambiguity



Ambiguity

- No…
  - Both CKY and Earley will result in multiple S structures for the [0,n] table entry.
  - They both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

Ambiguity

- In most cases, humans don't notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
- We'll try to model that with probabilities.
- But note something odd and important about the Groucho Marx example…

Next Time

- Partial Parsing and chunking
- After that we'll move on to probabilistic parsing

3/11/08 49