Parsing I:
Earley Parser

CMSC 35100

Natural Language Processing

May 1, 2003

Roadmap

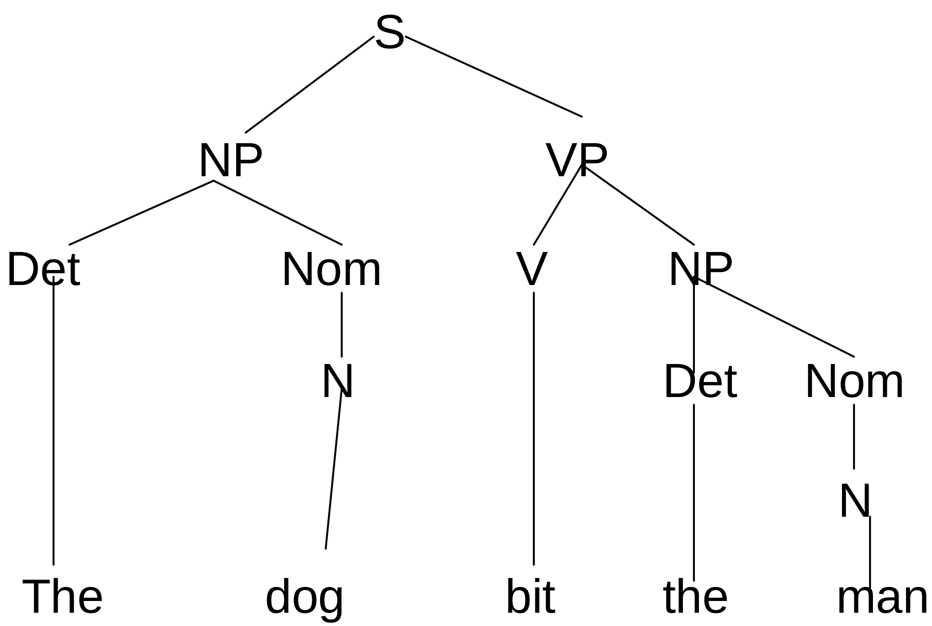
- Parsing:
  - Accepting & analyzing
  - Combining top-down & bottom-up constraints
    - Efficiency
  - Earley parsers
- Probabilistic CFGs
  - Handling ambiguity – more likely analyses
  - Adding probabilities
    - Grammar
    - Parsing: probabilistic CYK
    - Learning probabilities: Treebanks & Inside-Outside
    - Issues with probabilities

Representation: Context-free Grammars

- CFGs: 4-tuple
  - A set of terminal symbols: Σ
  - A set of non-terminal symbols: N
  - A set of productions P: of the form A -> α
    - Where A is a non-terminal and α in (Σ U N)*
  - A designated start symbol S
- L = W|w in Σ* and S=>*w
  - Where S=>*w means S derives w by some seq

Representation: Context-free Grammars

- Partial example
  - Σ: the, cat, dog, bit, bites, man
  - N: NP, VP, AdjP, Nominal
  - P: S-> NP VP; NP -> Det Nom; Nom-> N Nom|N
  - S



Parsing Goals

- Accepting:
  - Legal string in language?
    - Formally: rigid
    - Practically: degrees of acceptability
- Analysis
  - What structure produced the string?
    - Produce one (or all) parse trees for the string

Parsing Search Strategies

- Top-down constraints:
  - All analyses must start with start symbol: S
  - Successively expand non-terminals with RHS
  - Must match surface string
- Bottom-up constraints:
  - Analyses start from surface string
  - Identify POS
  - Match substring of ply with RHS to LHS
  - Must ultimately reach S

Integrating Strategies

- Left-corner parsing:
  - Top-down parsing with bottom-up constraints
  - Begin at start symbol
  - Apply depth-first search strategy
    - Expand leftmost non-terminal
    - Parser can not consider rule if current input can not be first word on left edge of some derivation
    - Tabulate all left-corners for a non-terminal

Issues

- Left recursion
  - If the first non-terminal of RHS is recursive ->
    - Infinite path to terminal node
    - Could rewrite
- Ambiguity: pervasive (costly)
  - Lexical (POS) & structural
    - Attachment, coordination, np bracketing
- Repeated subtree parsing
  - Duplicate subtrees with other failures

Earley Parsing

- Avoid repeated work/recursion problem
  - Dynamic programming
    - Store partial parses in “chart”
      - Compactly encodes ambiguity
    - $O(N^3)$
- Chart entries:
  - Subtree for a single grammar rule
  - Progress in completing subtree
  - Position of subtree wrt input

Earley Algorithm

- Uses dynamic programming to do parallel top-down search in (worst case) $O(N^3)$ time
- First, left-to-right pass fills out a chart with N+1 states
  - Think of chart entries as sitting between words in the input string keeping track of states of the parse at these positions
  - For each word position, chart contains set of states representing all partial parse trees generated to date. E.g. chart[0] contains all partial parse trees generated at the beginning of the sentence

Chart Entries

Represent three types of constituents:

- predicted constituents
- in-progress constituents
- completed constituents

Progress in parse represented by Dotted Rules

- Position of • indicates type of constituent
- $_0$ Book $_1$ that $_2$ flight $_3$
  - S → • VP, [0,0] (predicted)
  - NP → Det • Nom, [1,2] (in progress)
  - VP →V NP •, [0,3] (completed)
- [x,y] tells us what portion of the input is spanned so far by this rule
- **Each State s_i:**
  <dotted rule>, [<back pointer>,<current position>]

$_0$ Book $_1$ that $_2$ flight $_3$

S → • VP, [0,0]

- First 0 means S constituent begins at the start of input
- Second 0 means the dot here too
- So, this is a top-down prediction

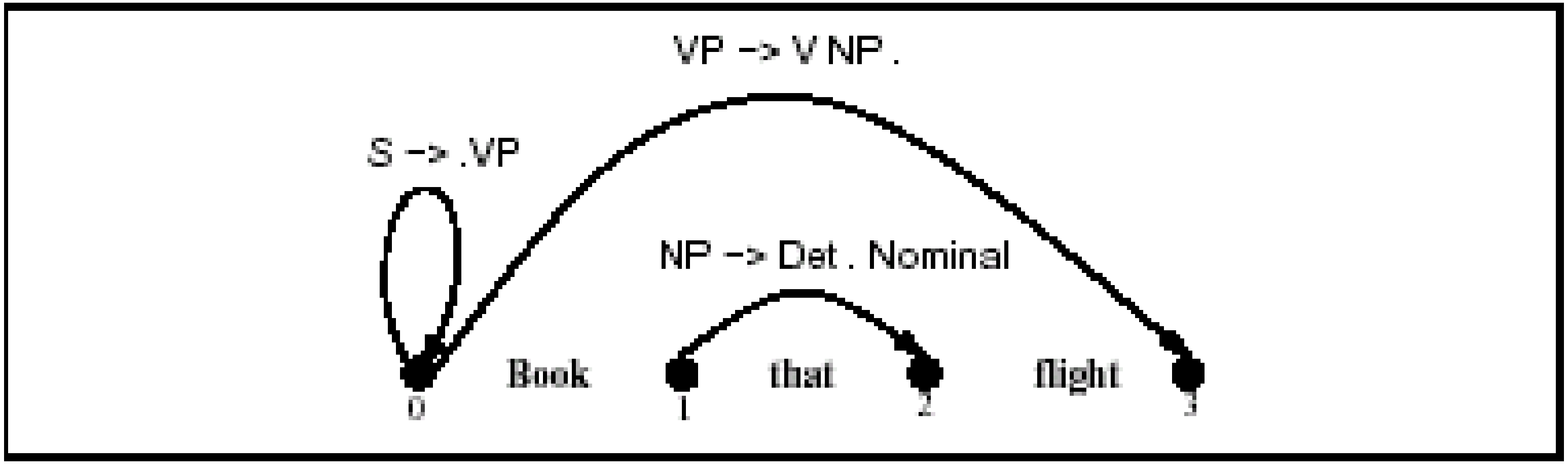
NP → Det • Nom, [1,2]

- the NP begins at position 1
- the dot is at position 2
- so, Det has been successfully parsed
- Nom predicted next

$_0$ Book $_1$ that $_2$ flight $_3$ (continued)

VP → V NP •, [0,3]

– Successful VP parse of entire input



Successful Parse

- Final answer found by looking at last entry in chart
- If entry resembles $S \rightarrow \alpha \bullet$ [nil,N] then input parsed successfully
- Chart will also contain record of all possible parses of input string, given the grammar

Parsing Procedure for the Earley Algorithm

- Move through each set of states in order, applying one of three operators to each state:
  - **predictor:** add predictions to the chart
  - **scanner:** read input and add corresponding state to chart
  - **completer:** move dot to right when new constituent found
- Results (new states) added to current or next set of states in chart
- No backtracking and no states removed: keep complete history of parse

States and State Sets

- **Dotted Rule s_i** represented as <dotted rule>, [<back pointer>, <current position>]

- **State Set S_j** to be a collection of states s_i with the same <current position>.

Earley Algorithm (simpler!)

1. Add Start → · S, [0,0] to state set 0
   Let i=1
2. **Predict** all states you can, adding new predictions to state set 0
3. **Scan** input word i—add all matched states to state set S_i.
   Add all new states produced by **Complete** to state set S_i
   Add all new states produced by **Predict** to state set S_i
   Let i = i + 1
   Unless i=*n*, repeat step 3.
4. At the end, see if state set *n* contains Start → S ·, [nil,n]

3 Main Sub-Routines of Earley Algorithm

- **Predictor**: Adds predictions into the chart.
- **Completer**: Moves the dot to the right when new constituents are found.
- **Scanner**: Reads the input words and enters states representing those words into the chart.

Predictor

- Intuition: create new state for top-down prediction of new phrase.
- Applied when non part-of-speech non-terminals are to the right of a dot: **S → • VP [0,0]**
- Adds new states to ***current*** chart
  - One new state for each expansion of the non-terminal in the grammar
    VP → • V [0,0]
    VP → • V NP [0,0]
- Formally:
  S_j: A → α · B β, [i,j]
  S_j: B → · γ, [j,j]

Scanner

- Intuition: Create new states for rules matching part of speech of next word.
- Applicable when part of speech is to the right of a dot: VP → • V NP [0,0] ‘Book…’
- Looks at current word in input
- If match, adds state(s) to ***next*** chart
  VP → V • NP [0,1]
- Formally:
  S_j: A → $\alpha \cdot B \beta$, [i,j]
  S_{j+1}: A → $\alpha B \cdot \beta$, [i,j+1]

Completer

- Intuition: parser has finished a new phrase, so must find and advance states all that were waiting for this
- Applied when dot has reached right end of rule

  NP → Det Nom • [1,3]

- Find all states w/dot at 1 and expecting an NP: VP → V • NP [0,1]
- Adds new (completed) state(s) to ***current*** chart : VP → V NP • [0,3]
- Formally: S_k: B → δ · , [j,k]

  S_k: A → α B · β, [i,k],

  where: S_j: A → α · B β, [i,j].

Example: State Set S_0 for Parsing “Book that flight” using Grammar G_0

$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
$NP \rightarrow \bullet Det\ NOMINAL$	[0,0]	Predictor
$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
$S \rightarrow \bullet VP$	[0,0]	Predictor
$VP \rightarrow \bullet Verb$	[0,0]	Predictor
$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor

Example: State Set S_1 for Parsing “Book that flight”

VP -> Verb.	[0,1]	Scanner
S -> VP.	[0,1]	Completer
VP -> Verb. NP	[0,1]	Scanner
NP -> .Det Nom	[1,1]	Predictor
NP -> .Proper-Noun	[1,1]	Predictor

Prediction of Next Rule

- When VP → V • is itself processed by the Completer, S → VP • is added to Chart[1] since VP is a left corner of S
- Last 2 rules in Chart[1] are added by **Predictor** when VP → V • NP is processed
- And so on….

Last Two States

Chart[2]

NP->Det. Nominal	[1,2]	Scanner
Nom -> .Noun	[2,2]	Predictor
Nom -> .Noun Nom	[2,2]	Predictor

Chart[3]

Nom -> Noun.	[2,3]	Scanner
Nom -> Noun. Nom	[2,3]	Scanner
NP -> Det Nom.	[1,3]	Completer
VP -> Verb NP.	[0,3]	Completer
S -> VP.	[0,3]	Completer
Nom -> .Noun	[3,3]	Predictor
Nom -> .Noun Nom	[3,3]	Predictor

How do we retrieve the parses at the end?

- Augment the Completer to add pointers to prior states it advances as a field in the current state
  - i.e. what state did we advance here?
  - Read the pointers back from the final state

Probabilistic CFGs

Handling Syntactic Ambiguity

- Natural language syntax
  - Varied, has DEGREES of acceptability
  - Ambiguous
- Probability: framework for preferences
  - – Augment original context-free rules: PCFG
  - – Add probabilities to transitions

PP $\xrightarrow{1.0}$ P NP

NP $\xrightarrow{0.2}$ N
NP $\xrightarrow{0.65}$ Det N
NP $\xrightarrow{0.10}$ Det Adj N
NP $\xrightarrow{0.05}$ NP PP

VP $\xrightarrow{0.45}$ V
VP $\xrightarrow{0.45}$ V NP
VP $\xrightarrow{0.10}$ V NP PP

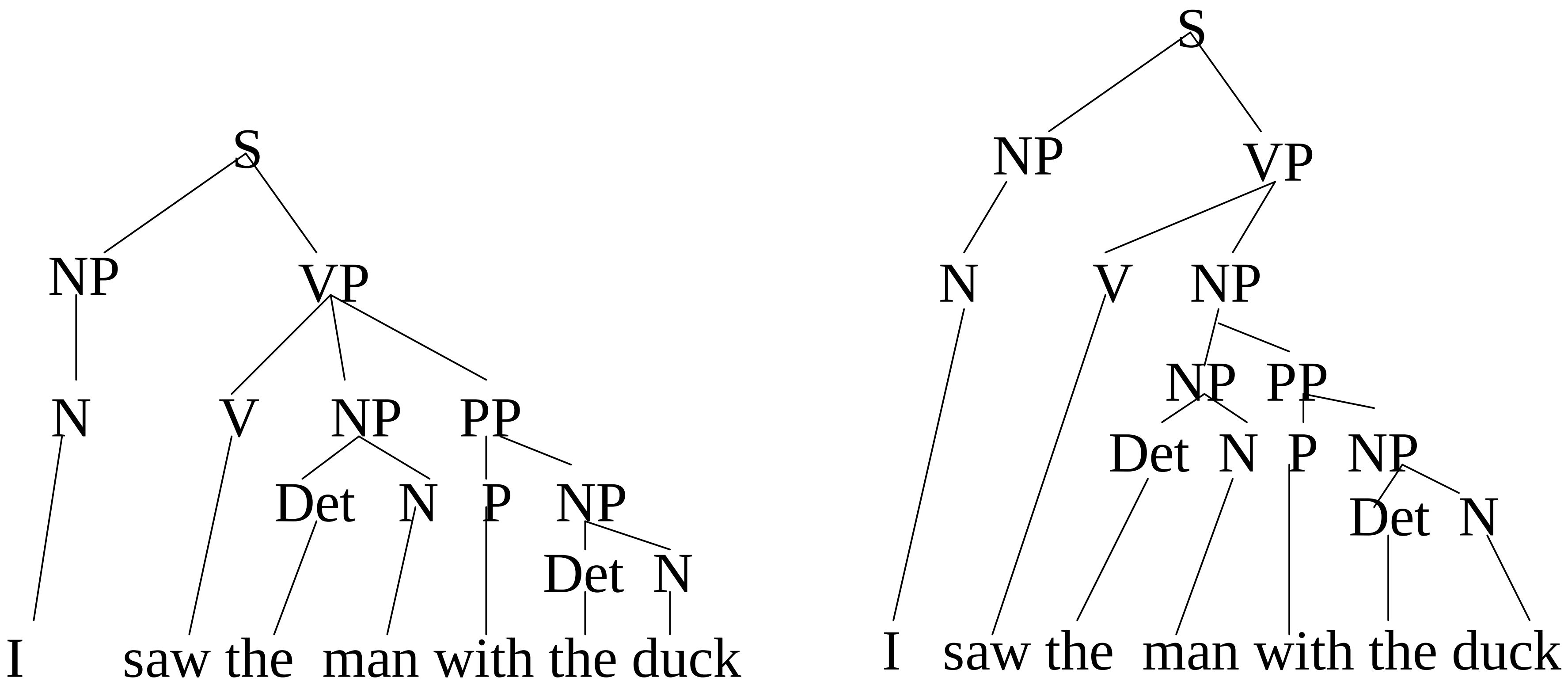
S $\xrightarrow{0.85}$ NP VP
S $\xrightarrow{0.15}$ S conj S

PCFGs

- Learning probabilities
  - Strategy 1: Write (manual) CFG,
    - Use treebank (collection of parse trees) to find probabilities
- Parsing with PCFGs
  - Rank parse trees based on probability
  - Provides graceful degradation
    - Can get some parse even for unusual constructions - low value

Parse Ambiguity

- Two parse trees



Parse Probabilities

$$P(T,S) = \prod_{n \in T} p(r(n))$$

- T(ree),S(entence),n(ode),R(ule)
- T1 = 0.85*0.2*0.1*0.65*1*0.65 = 0.007
- T2 = 0.85*0.2*0.45*0.05*0.65*1*0.65 = 0.003

- Select T1
- Best systems achieve 92-93% accuracy

Probabilistic CYK Parsing

- Augmentation of Cocke-Younger-Kasami
  - Bottom-up parsing
- Inputs
  - PCFG in CNF G={N,Σ,P,S,D}, N have indices
  - N words w1…wn
- DS:Dynamic programming array: π[i,j,a]
    - Holding max prob index a spanning i,j
- Output: Parse π[1,n,1] with S and w1..wn

Probabilistic CYK Parsing

- Base case: Input strings of length 1
  - In CNF, prob must be from A=>wi
- Recursive case: For strings > 1, A=>*wij iff there is rule A->BC and some k, 1<=k<j st B derives the first k symbols and C the last j-k. Since len < |wij|, probability in table. Multiply subparts; compute max over all subparts.

Inside-Outside Algorithm

- EM approach
  - Similar to Forward-Backward training of HMM
- Estimate number of times production used
  - Base on sentence parses
  - Issue: Ambiguity
    - Distribute across rule possibilities
  - Iterate to convergence

Issues with PCFGs

- Non-local dependencies
  – Rules are context-free; language isn't
- Example:
  – Subject vs non-subject NPs
    - Subject: 90% pronouns (SWB)
    - NP-> Pron vs NP-> Det Nom: doesn't know if subj
- Lexical context:
  – Verb subcategorization:
    - Send NP PP vs Saw NP PP
  – One approach: lexicalization