

## Chart Parsing with Active Charts

Informatics 2A: Lecture 17

Bonnie Webber

School of Informatics  
University of Edinburgh  
bonnie@inf.ed.ac.uk

3 November 2009

- 1 Adding Prediction to the Chart
  - From constituents to productions
  - Dotted Rules
- 2 Active Chart Parsers
  - Adding Edges
  - Bottom-up Active Parsing
  - Top-down Active Parsing
  - Earley Algorithm
  - Visualizing the Chart
  - Comparing Earley and CYK

Reading:

Same as for Lecture 16

### Chart entries

As we saw in the last lecture, the CYK algorithm avoids redundant work by **memo-izing** (ie, storing in a chart) all the constituents it finds.

What it hasn't recorded is any justification for a chart entry – why it was built.

So if we have two VP rules:

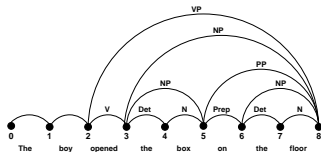
$VP \rightarrow V NP$   
 $VP \rightarrow VP PP$

and the input string

*The boy opened the box on the floor*

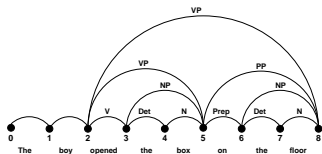
### Chart entries

We don't know which production the VP arc [2, 8] represents (here using a graph representation of the chart)



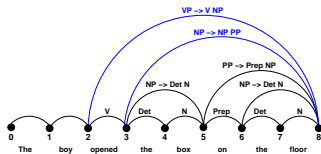
## Chart entries

We don't know which production the VP arc [2, 8] represents (here using a graph representation of the chart)



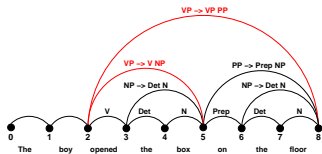
## Chart entries

If the entire **production** were recorded, rather than just its LHS (ie, the constituent that it analyses), then we'd know.

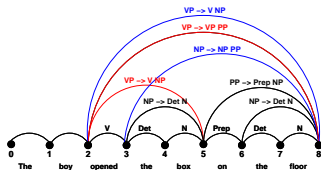


## Chart entries

If the entire **production** were recorded, rather than just its LHS (ie, the constituent that it analyses), then we'd know.



## Chart entries: Both analyses



## Chart entries

But if we record **completed productions** (ie, ones whose entire RHS have been recognized), we could also consider recording **incomplete productions** (ie, ones for which there may so far be only partial evidence).

**Incomplete productions** (aka **incomplete constituents**) are effectively **predictions** about what might come next and what will be learned from finding it.

**Incomplete constituents** can be represented using an extended form of production rule called a **dotted rule**.

The **dot** indicates how much of the RHS has already been found. The rest is a prediction of what is to come.

## Examples of Dotted Rules

The grammar rule  $VP \rightarrow V NP$  yields the following dotted rules:

$VP \rightarrow . V NP$  **incomplete edge**

$VP \rightarrow V . NP$  **incomplete edge**

$VP \rightarrow V NP .$  **complete edge**

## Dotted Rules

Just as we can record in the chart the production rules for complete constituents, we can record the dotted rules for incomplete constituents, noting

- the production rule used in the analysis;
- the part of the rule already found (to the left of the dot), and the part still predicted to be found (to the right of the dot);
- the start and end position of the material already found.

The chart could record for the substring ...<sub>5</sub> on<sub>6</sub> the<sub>7</sub> floor<sub>8</sub>

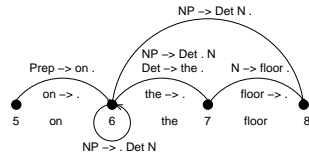
$NP \rightarrow Det . N, [6, 7]$

indicating the word between 6 and 7 is spanned by Det, and an N is predicted next. If found, we get an NP starting at 6.

## Dotted Rules

The extreme dotted rule has the dot at the left end of the RHS, meaning **nothing** on the RHS has yet been found, and everything is predicted.

Example of a chart labelled with dotted rules (in graph representation):



## Adding Edges

An **active chart parser** can add an edge for any of three reasons:

- 1 The **input** can license an edge. In particular, each word  $w_i$  in the input licenses the complete edge  $[w_i \rightarrow \bullet, (i, i + 1)]$ ;
- 2 The **grammar** can license an edge. In particular, each grammar production  $A \rightarrow \alpha$  licenses a self-loop edge  $[A \rightarrow \bullet \alpha, (i, i)]$  for every  $i, 0 \leq i < n$ , where  $n$  is the length of the string.
- 3 The **contents of the current chart** can license an edge.

It rarely makes sense to add **all** licensed edges to a chart, since many of them will not be used in any complete parse.

## Fundamental Rule

The rules used by an active chart parser to decide when an edge should be added to a chart, along with a specification of when rules should be applied, forms the parser's **strategy**.

Every active chart parser uses what's called the **Fundamental Rule** to combine an incomplete edge with a complete one.

If the chart contains the edges

$$[A \rightarrow \alpha \bullet B \beta, (i, j)]$$

$$[B \rightarrow \gamma \bullet, (j, k)]$$

then add the new edge

$$[A \rightarrow \alpha B \bullet \beta, (i, k)]$$

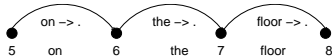
where  $\alpha, \beta$ , and  $\gamma$  are (possibly empty) sequences of terminals or non-terminals.

## Bottom-up Active Parsing

As with the CYK Parser, **Bottom-up Active Parsing** starts from the input string, identifying sequences of words and phrases that correspond to the RHS of a grammar production.

### Bottom-up Initialization Rule

For every word  $w_i$ , add the edge  $[w_i \rightarrow \bullet, (i, i + 1)]$ :



## Bottom-up Active Parsing

### Bottom-up Predict Rule

If the chart contains the complete edge

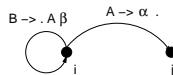
$$[A \rightarrow \alpha \bullet, (i, j)]$$

and the grammar contains the production

$$B \rightarrow A \beta$$

then add the self-loop edge

$$[B \rightarrow \bullet A \beta, (i, i)]$$



## Bottom-up Active Parsing Strategy

- Create an empty chart spanning the sentence.
- Apply the **Bottom-Up Initialization Rule** to each word.
- Until no more edges are added:
  - Apply the **Bottom-Up Predict Rule** everywhere it applies.
  - Apply the **Fundamental Rule** everywhere it applies.
- Return all of the parse trees corresponding to the parse edges in the chart.

```
>>> from nltk.app import chartparser
>>> chartparser()
```

## Top-down Active Parsing

Like Bottom-up Active Parsing, TD Active Parsing uses an **Initialization Rule** and the **Fundamental Rule**. But it also has a rule for making more self-looping TD hypotheses (**TD Expand**) and a rule for consuming words in the input (**TD Match**).

### Top-Down Initialization Rule

For every grammar production of the form:

$$S \rightarrow \alpha$$

add the self-loop edge:

$$[S \rightarrow \bullet \alpha, (0, 0)]$$



## Top-down Active Parsing

Top-Down Expansion adds more self-looping edges.

### Top-Down Expand Rule

If the chart contains the incomplete edge

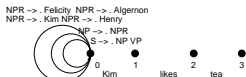
$$[A \rightarrow \alpha \bullet B \beta, (i, j)]$$

then for each grammar production

$$B \rightarrow \gamma$$

add the edge:

$$[B \rightarrow \bullet \gamma, (j, j)]$$



## Top-down Active Parsing

Top-Down Match Rule consumes a word from the input string.

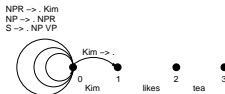
### Top-Down Match Rule

If the chart contains the incomplete edge

$$[A \rightarrow \alpha \bullet w_j \beta, (i, j)]$$

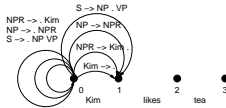
where  $w_j$  is the  $j^{\text{th}}$  word of the input string, add a new complete edge

$$[w_j \rightarrow \bullet, (j, j + 1)]$$



## Top-down Active Parsing

The **fundamental rule** allows self-loops to combine with and bridge the matched input word.



- Create an empty chart spanning the sentence.
- Apply the Top-Down Initialization Rule.
- Until no more edges are added:
  - Apply the Top-Down Expand Rule everywhere it applies.
  - Apply the Top-Down Match Rule everywhere it applies.
  - Apply the Fundamental Rule everywhere it applies.

Return all of the parse trees corresponding to the parse edges in the chart.

```
>>> from nltk.app import chartparser
>>> chartparser()
```

## Earley Algorithm

The Earley algorithm is a bottom-up chart parser with top-down prediction.

- **Predictor:** TD Initialization Rule + TD Expansion Rule
- **Scanner:** TD/BU Match Rule
- **Completer:** Fundamental Rule

Unlike the BU Active Chart Parser, edges are added in strictly left-to-right order.

if  $A \rightarrow X \cdot B$ ,  $[i, j]$  is added before  $C \rightarrow Y \cdot D$ ,  $[i', j']$  then  $j \leq j'$ .

**Scanner Rule** matching uses **parts of speech** rather than words.

### Scanner Rule

If the chart contains the incomplete edge

$$[A \rightarrow \alpha \bullet P \beta, (i, j)]$$

and  $w_j$  is the  $j^{\text{th}}$  word of the input string and  $P$  is a valid part of speech for  $w_j$ , then add the new complete edges:

$$[P \rightarrow w_j \bullet, (j, j + 1)]$$

$$[w_j \rightarrow \bullet, (j, j + 1)]$$

## Earley Algorithm

The Scanner Rule efficiently packages up a sequence of three rule applications:

- 1 BU Initialization for  $[w_j \rightarrow \bullet, (j, j + 1)]$
- 2 TD Expand for  $[P \rightarrow \bullet w_j, (j, j)]$
- 3 Fundamental Rule for  $[P \rightarrow w_j \bullet, (j, j + 1)]$

## Visualizing the Chart

### Grammatical rules

$S \rightarrow NP VP$   
 $NP \rightarrow Det Nom$   
 $NP \rightarrow Nom$   
 $Nom \rightarrow N SRel$   
 $Nom \rightarrow N$   
 $VP \rightarrow TV NP$   
 $VP \rightarrow IV PP$   
 $VP \rightarrow IV$   
 $PP \rightarrow Prep NP$   
 $SRel \rightarrow Relpro VP$

### Lexical rules

$Det \rightarrow a | the$  (determiner)  
 $N \rightarrow fish | frogs | soup$  (noun)  
 $Prep \rightarrow in | for$  (preposition)  
 $TV \rightarrow saw | ate$  (transitive verb)  
 $IV \rightarrow fish | swim$  (intransitive verb)  
 $Relpro \rightarrow that$  (relative pronoun)

## Step 0

$j = 0$	label	begin	end	reason
0	$\gamma \rightarrow \cdot S$	0	0	TD initialise
1	$S \rightarrow \cdot NP VP$	0	0	TD expand
2	$NP \rightarrow \cdot Det Nom$	0	0	predict from 1
3	$NP \rightarrow \cdot Nom$	0	0	predict from 1
4	$Nom \rightarrow \cdot N SRel$	0	0	predict from 3
5	$Nom \rightarrow \cdot N$	0	0	predict from 3

## step 1

$j = 1$	label	begin	end	reason
6	$N \rightarrow fish \cdot$	0	1	scan
7	$IV \rightarrow fish \cdot$	0	1	scan
8	$Nom \rightarrow N \cdot SRel$	0	1	complete from 4 using 6
9	$Nom \rightarrow N \cdot$	0	1	complete from 5 using 6
10	$NP \rightarrow Nom \cdot$	0	1	complete from 3 using 9
11	$SRel \rightarrow \cdot Relpro VP$	1	1	predict from 8
12	$S \rightarrow NP \cdot VP$	0	1	complete from 1 using 10
13	$VP \rightarrow \cdot TV NP$	1	1	predict from 12
14	$VP \rightarrow \cdot IV$	1	1	predict from 12
15	$VP \rightarrow \cdot IV PP$	1	1	predict from 12

## Step 2

$j = 2$	label	begin	end	reason
16	IV → <b>swim</b> .	1	2	scan
17	VP → IV .	1	2	complete from 14 using 16
18	S → NP VP .	0	2	complete from 12 using 17
19	VP → IV . PP	1	2	complete from 15 using 16
20	PP → . Prep NP	2	2	predict from 19

## Step 3

$j = 3$	label	begin	end	reason
21	Prep → <b>in</b> .	2	3	scan
22	PP → Prep . NP	2	3	complete from 20 using 21
23	NP → . Det Nom	3	3	predict from 22
24	NP → . Nom	3	3	predict from 22
25	Nom → . N SRel	3	3	predict from 24
25	Nom → . N	3	3	predict from 24

## Step 4

$j = 4$	label	begin	end	reason
26	Det → <b>the</b> .	3	4	scan
27	NP → Det . Nom	3	4	complete from 23 using 26
28	Nom → . N SRel	4	4	predict from 27
29	Nom → . N	4	4	predict from 27

## Step 5

$j = 5$	label	begin	end	reason
30	N → <b>soup</b> .	4	5	scan
31	Nom → N . SRel	4	5	complete from 28 using 30
32	SRel → . Relpro NP	5	5	predict from 31
33	Nom → N .	4	5	complete from 29 using 30
34	NP → Det Nom .	3	5	complete from 27 using 33
35	PP → Prep NP .	2	5	complete from 22 using 34
36	VP → IV PP .	1	5	complete from 19 using 35
37	S → NP VP .	0	5	complete from 12 using 36

## Comparing Earley and CYK

Both Earley and CYK are **bottom-up** chart parsing algorithms, but Earley also uses **top-down** prediction to avoid building edges that will not lead to a valid parse.

To illustrate a difference between Earley and CYK, consider an NP with an **object relative clause**, allowed by adding to the grammar, rules such as:

The milk that we drank	Nom $\rightarrow$ N Relpro NP TV
The milk which we drank	Nom $\rightarrow$ N NP TV
The milk we drank yesterday	Relpro $\rightarrow$ that   which

## Comparing Earley and CYK

An object relative clause can lead, in English, to a form of **local ambiguity** called a **garden path sentence**:

The horse raced past the barn fell

This comes from the **lexical ambiguity** of *raced*  $\subset$  TV, IV. (The sentence means the same as *The horse fell (that was) raced past the barn.*)

Object relative clauses also allow a kind of **center embedding** that is particularly difficult to understand:

The cheese was moldy.  
The cheese the rat ate was moldy.  
The cheese the rat the cat chased ate was moldy .

## Comparing Earley and CYK

Consider the string:

0 The <sub>1</sub> ride <sub>2</sub> the <sub>3</sub> horse <sub>4</sub> gave <sub>5</sub> was <sub>6</sub> wild <sub>7</sub>

where *ride*  $\subset$  N, TV, IV.

What happens when CYK gets to *ride?* to *horse?*

What happens when Earley gets to *ride?* to *horse?*

With Earley, edges are only added when they can serve to extend a possible parse tree. So Earley won't analyse the string *ride the horse* as a VP.

## Summary

- Active chart algorithms add prediction to chart parsing via incomplete edges (using dotted rules). This avoids building unnecessary structure.
- We've seen three different types of active chart algorithms:
  - strictly bottom-up
  - strictly top-down
  - Earley algorithm (bottom-up, but with top-down prediction)
- All active chart algorithms
  - use the fundamental rule;
  - have a way of making hypotheses and a way of expanding hypotheses;
  - link at some point to the input string.