Chart Parsing

Ewan Klein
ewan@inf.ed.ac.uk

ICL — 7 November 2005
Review Top-down Parsing

Chart Parsing
  Overview
  Charts as Graphs
  The Basic Idea
  Example States

The Earley Algorithm
  Parsing Example
  Left Recursion
Parsing

Parsing with a CFG is the task of assigning a correct tree (or derivation) to a string given some grammar. A correct tree is:

▶ consistent with the grammar, and
▶ the leaves of the tree cover all and only the words in the input.

There may be a very large number of correct trees for any given input . . .
Problems that arise

- Left Recursion
- Ambiguity
- Inefficiencies due to backtracking
Common substructures

- Despite ambiguity and backtracking there are common substructures to be taken advantage of.
- Consider parsing the following NP:
  
  a flight from Indianapolis to Houston on TWA

  with the following rules:

  \[
  NP \rightarrow \text{Det} \ Nom \\
  Nom \rightarrow \text{Nom} \ PP
  \]

- What happens with a top-down parser?
Backtracking

```
NP
  Det a
  Nom flight from Indianapolis to Houston on TWA
```

```
NP
  Det a
  Nom flight from Indianapolis to Houston on TWA
```
Backtracking, cont

```
Det
  a

NP
  Nom
    Nom
      Nom
        flight
        from Indianapolis
    PP
      to Houston

on TWA
```

Ewan Klein ewan@inf.ed.ac.uk
Backtracking, cont

```
NP
  / \
Det  Nom
   /  \
a   Nom
      /  \
      Nom PP
          /  \
         PP PP
"on TWA"

Nom Nom
     /  /
    "flight" "from Indianapolis"
    /  /
   "to Houston"
```
Dynamic Programming

Our current algorithm builds valid trees, discards them during backtracking, then rebuilds them.

- the subtree for *a flight* was derived 4 times.

Dynamic programming is one answer to problems that have sub-problems that get solved again and again. We’ll consider an algorithm that fills a table with solutions to sub-problems that:

- does a parallel top-down search with bottom-up filtering
- does not do repeated work
- solves the left-recursion problem
Dynamic Programming and Parsing

▶ Systematically fill in tables of solutions to subproblems.
▶ When complete, the table contains all possible solutions to all of the subproblems needed to solve the whole problem.
▶ For parsing:
  ▶ the table stores subtrees for constituents.
  ▶ Solves reparsing inefficiencies, because subtrees are not reparsed but looked up.
  ▶ Solves ambiguity explosions, because the table *implicitly* stores all parses.
  ▶ Each subtree is represented only once and shared by all parses that need it.
Lexical Edges
Nonterminal Edges: NP

Ewan Klein ewan@inf.ed.ac.uk
Nonterminal Edges: VP

- book
- that
- flight

NP → VP
V → book
Det → that
N → flight
Nonterminal Edges: Rules

- Useful to label arcs with rules rather than categories:

```
V → book
Det → that
N → flight
```

```
NP → Det N
VP → V NP
```
Incomplete Edges

- Recall that the parser can make **predictions** on the basis of rules:
  - I’m trying to expand an VP, and I’ve found a V so I’ll start looking for an NP.

- Record incomplete constituents with **dotted rules**:
  - Dot on the RHS of a rule shows what we’ve found already:
    - $\text{VP} \rightarrow \text{V} \bullet \text{NP}$
  - We can use this as a label for an incomplete constituent.
Incomplete Edges, cont.

- **VP → V NP**
- **NP → Det N**
- **VP → V * NP**
- **V → book**
- **Det → that**
- **N → flight**
Dynamic Programming and Parsing

The Earley algorithm:

- fills a table (the chart) in a single left-to-right pass over the input.
- The chart will be size $N + 1$, where $N$ is the number of words in the input.
- Chart entries are associated with the gaps between the words — like slice indexing in Python.
- For each word position in the sentence, the chart contains a set of edges representing the partial parse trees generated so far.
- So Chart[0] is the set of edges representing the parse before looking at a word.
States

J&M call the chart entries states.
The chart entries represent three distinct kinds of things:
- completed constituents;
- in-progress constituents; and
- predicted constituents

The three kinds of states correspond to different dot locations in dotted rules:

- **Completed:** \( VP \rightarrow V \ NP \bullet \)
- **In-progress:** \( NP \rightarrow \text{Det} \bullet \text{Nom} \)
- **Predicted:** \( S \rightarrow \bullet \ VP \)
Incomplete NP Edge: Self-loop

NP → * Det Nom

V → book
Det → that
N → flight

0 book 1 that 2 flight 3
States, cont.

Given dotted rules like those we’ve just seen, we need to record:

- where the represented constituent is in the input, and
- what its parts are.

So we add a pair of coordinates \([x, y]\) to the state:

- \(A \rightarrow \alpha, [x, y]\)
- \(x\) indicates the position in input where the state begins
- \(y\) indicates position of dot
Example with coordinates

VP → V NP * [0,3]

NP → Det N * [1,3]

V → book

Det → that

N → flight
States, cont.

Example states in parsing *Book that flight*:

1. $S \rightarrow \bullet \ VP, [0,0]$
   - First 0 indicates that the constituent begins at the start of the input.
   - Second 0 indicates that the dot also begins at start of input, and thus indicates a top-down prediction.

2. $NP \rightarrow \text{Det} \bullet \ Nom, [1,2]$
   - the NP begins at position 1
   - the dot is at position 2
   - Det has been successfully *completed*
   - Nom is *predicted* next
1. \( VP \rightarrow V \ NP \bullet, [0,3] \)
   - VP is *completed*
   - no further *predictions* from this rule
   - a successful parse that spans the entire input
Success

The final answer is found by looking at the last column of the table. In particular, for an input of $N$ words, if we find the following kind of state in the chart then we’ve succeeded:

$$S \rightarrow \alpha \bullet, [0,N]$$
Parsing

Parsing is sweeping through the chart creating the three kinds of states as we go. States are never removed, and we never backtrack.

- New *predicted* states are based on existing table entries (predicted, or in-progress) that predict a certain constituent at that spot.
- New *in-progress* states are created by updating older states to reflect the fact that previously expected completed constituents have been located.
- New *completed* states are created when the dot in an in-progress state moves to the end.
More Specifically

1. Predict all the states you can.
2. Read an input.
   ▶ See what predictions you can match.
   ▶ Extend matched states, add new predictions.
   ▶ Go to next state (goto 2)
3. At the end, see if \textit{state}[N + 1] contains a complete S.
The Earley algorithm has three main functions that do all the work:

**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**

```
procedure PREDICTOR((A → α • B β, [i,j]))
  for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar)
    do
      ENQUEUE((B → • γ, [j,j]), chart[j])
  end
```

- Intuition: new states represent top-down expectations.
- Applied when a state has a non-terminal to the right of a dot that is not a part-of-speech.
- Generates one new state for each alternative expansion of the non-terminal in the grammar.
- Adds states to the same chart entry as generating state.
Completer

```plaintext
procedure COMPLETER((B → γ •, [j, k]))
for each (A → α • B β, [i, j]) in chart[j] do
    ENQUEUE((A → α B • β, [i, k]), chart[k])
end
```

- **Intuition:** parser has discovered a constituent, so must find and advance states that were looking for this grammatical category at this position in input.
- **Applied when dot has reached right end of rule.**
- **New states are generated by copying old state and advancing dot over expected category.**
- **Adds new states to same chart entry as generating state.**
Scanner

\[
\text{procedure \textsc{Scanner}}((A \rightarrow \alpha \cdot B \beta, [i,j])) \quad \text{if } B \in \text{Parts-of-Speech}(\text{word}[j]) \text{ then} \\
\text{Enqueue}((B \rightarrow \text{word}[j] \cdot, [j, j + 1]), chart[j+1])
\]

- New states for predicted part-of-speech.
- Applicable when part-of-speech is to the right of a dot.
- Adds states to next chart entry.

Note: Earley parser uses top-down predictions to help disambiguate part-of-speech ambiguities. Only those parts-of-speech of a word that are predicted by some state will find their way into the chart.
Mini grammar and lexicon

<table>
<thead>
<tr>
<th>mini grammar</th>
<th>lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>Aux NP VP</td>
</tr>
<tr>
<td>NP → Det Nom</td>
<td>PropN</td>
</tr>
<tr>
<td>Nom → Nom PP</td>
<td>N Nom</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>P</td>
</tr>
<tr>
<td>VP → V</td>
<td>V NP</td>
</tr>
<tr>
<td>Nom → N PP</td>
<td>N Nom</td>
</tr>
<tr>
<td>Det → that</td>
<td>this</td>
</tr>
<tr>
<td>N → book</td>
<td>flight</td>
</tr>
<tr>
<td>V → book</td>
<td>include</td>
</tr>
<tr>
<td>Aux → does</td>
<td>P</td>
</tr>
<tr>
<td>P → from</td>
<td>to</td>
</tr>
<tr>
<td>PropN → Houston</td>
<td>TWA</td>
</tr>
</tbody>
</table>
## Example: Chart[0] and Chart[1]

### Chart[0]

<table>
<thead>
<tr>
<th>Production</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma \rightarrow S$</td>
<td>[0,0]</td>
<td>Dummy start state</td>
</tr>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$S \rightarrow Aux \ NP \ VP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ NOMINAL$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$NP \rightarrow Proper-Noun$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>

### Chart[1]

<table>
<thead>
<tr>
<th>Production</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb $\rightarrow$ book</td>
<td>[0,1]</td>
<td>Scanner</td>
</tr>
<tr>
<td>VP $\rightarrow$ Verb</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S $\rightarrow$ VP</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>VP $\rightarrow$ Verb \ NP</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>NP $\rightarrow$ Det \ NOMINAL</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>
Example: Chart[1] and Chart[2]

Chart[1]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Production</th>
<th>Start Index</th>
<th>End Index</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb</td>
<td>book</td>
<td>[0,1]</td>
<td></td>
<td>Scanner</td>
</tr>
<tr>
<td>VP</td>
<td>Verb</td>
<td>[0,1]</td>
<td></td>
<td>Completer</td>
</tr>
<tr>
<td>S</td>
<td>VP</td>
<td>[0,1]</td>
<td></td>
<td>Completer</td>
</tr>
<tr>
<td>VP</td>
<td>Verb • NP</td>
<td>[0,1]</td>
<td></td>
<td>Completer</td>
</tr>
<tr>
<td>NP</td>
<td>• Det • NOMINAL</td>
<td>[1,1]</td>
<td></td>
<td>Predictor</td>
</tr>
<tr>
<td>NP</td>
<td>• Proper • Noun</td>
<td>[1,1]</td>
<td></td>
<td>Predictor</td>
</tr>
</tbody>
</table>

Chart[2]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Production</th>
<th>Start Index</th>
<th>End Index</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det</td>
<td>that</td>
<td>[1,2]</td>
<td></td>
<td>Scanner</td>
</tr>
<tr>
<td>NP</td>
<td>Det • NOMINAL</td>
<td>[1,2]</td>
<td></td>
<td>Completer</td>
</tr>
<tr>
<td>NOMINAL</td>
<td>• Noun</td>
<td>[2,2]</td>
<td></td>
<td>Predictor</td>
</tr>
<tr>
<td>NOMINAL</td>
<td>• Noun NOMINAL</td>
<td>[2,2]</td>
<td></td>
<td>Predictor</td>
</tr>
</tbody>
</table>
Example: Chart[3]

Chart[3]

<table>
<thead>
<tr>
<th>Production</th>
<th>String</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun → flight ⋄</td>
<td></td>
<td>[2,3] Scanner</td>
</tr>
<tr>
<td>NOMINAL → Noun ⋄</td>
<td></td>
<td>[2,3] Completer</td>
</tr>
<tr>
<td>NOMINAL → Noun ⋄ NOMINAL</td>
<td></td>
<td>[2,3] Completer</td>
</tr>
<tr>
<td>NP → Det NOMINAL ⋄</td>
<td></td>
<td>[1,3] Completer</td>
</tr>
<tr>
<td>VP → Verb NP ⋄</td>
<td></td>
<td>[0,3] Completer</td>
</tr>
<tr>
<td>S → VP ⋄</td>
<td></td>
<td>[0,3] Completer</td>
</tr>
<tr>
<td>NOMINAL → ⋄ Noun</td>
<td></td>
<td>[3,3] Predictor</td>
</tr>
<tr>
<td>NOMINAL → ⋄ Noun NOMINAL</td>
<td></td>
<td>[3,3] Predictor</td>
</tr>
</tbody>
</table>
Examples: Left Recursion

What about parsing the NP *a flight from Denver to Boston* with the following rules:

\[
\begin{align*}
NP & \rightarrow NP \ PP \\
NP & \rightarrow Det \ Nom \\
NP & \rightarrow Proper-Noun
\end{align*}
\]

- We construct the state \((NP \rightarrow \bullet \ NP \ PP, [0,0])\) and add it to \(chart[0]\).
- The \texttt{Predictor} function then requires us to find a rule which expands the (non-lexical) category immediately to the right of the dot.
- So let’s pick the first rule above, and \texttt{ENQUEUE} the state \((NP \rightarrow \bullet \ NP \ PP, [0,0])\).
- But this is already in the state, so we don’t add it again.
Reading

- Read section 10.4 of J&M
- Read the NLTK-Lite Tutorial on Chart Parsing