Topics in Computational Linguistics — Parsing and Generation —

Stephan Oepen
Dan Flickinger

Center for the Study of Language and Information
Stanford University — Spring Quarter 2003

{oe|dan}@csli.stanford.edu

http://lingo.stanford.edu/courses/03/pg/
Review: Chart Parsing

Basic Notions

• Use chart to record partial analyses, indexing them by string positions;
• count inter-word vertices; CKY: chart row is start, column end vertex;
• treat multiple ways of deriving the same category for some substring as equivalent; pursue only once when combining with other constituents.

Key Benefits

• Dynamic programming (memoization): avoid recomputation of results;
• efficient indexing of constituents: no search by start or end positions;
• compute parse forest with exponential ‘extension’ in polynomial time.
The CKY (Cocke, Kasami, & Younger) Algorithm

for (0 ≤ i < |input|) do 
  chart \([i, i+1]\) \(\leftarrow\) \(\{\alpha \mid \alpha \rightarrow \text{input}_i \in P\}\);
for (0 ≤ l < |input|) do 
  for (0 ≤ i < |input| − l) do 
    for (1 ≤ j ≤ l) do 
      if \(\alpha \rightarrow \beta_1 \beta_2 \in P \land \beta_1 \in \text{chart}[i, i+j] \land \beta_2 \in \text{chart}[i+j, i+l+1]\) then 
        chart \([i, i+l+1]\) \(\leftarrow\) chart \([i, i+l+1]\) \(\cup\) \(\{\alpha\}\);

0 1 2 3 4 5
0 NP S S 
1 V VP VP 
2 NP NP 
3 P PP 
4 NP 

Kim 1 adores 2 snow 3 in 4 Oslo 5
Quantifying the Complexity of the Parsing Task

Recursive Function Calls

Kim adores snow (in Oslo)^n

<table>
<thead>
<tr>
<th>n</th>
<th>trees</th>
<th>calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>46</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>170</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>593</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>2,093</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>7,539</td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>27,627</td>
</tr>
<tr>
<td>6</td>
<td>429</td>
<td>102,570</td>
</tr>
<tr>
<td>7</td>
<td>1430</td>
<td>384,566</td>
</tr>
<tr>
<td>8</td>
<td>4862</td>
<td>1,452,776</td>
</tr>
</tbody>
</table>

...
The Benefits of Ambiguity Packing

(Recursive) Function Calls

- naive bottom-up parser
- memoized bottom-up
- Cocke-Kasami-Younger

Number of Prepositional Phrases (n)

0 1 2 3 4 5 6 7 8

0 2500 5000 7500 10000

Parsing and Generation (48)
Limitations of the CKY Algorithm

**Built-In Assumptions**

- *Chomsky Normal Form* grammars: $\alpha \rightarrow \beta_1\beta_2$ or $\alpha \rightarrow \gamma$ ($\beta_i \in C$, $\gamma \in \Sigma$);
- breadth-first (aka exhaustive): always compute all values for each cell;
- rigid control structure: bottom-up, left-to-right (one diagonal at a time).

**Generalized Chart Parsing**

- Liberate order of computation: no assumptions about earlier results;
- *active edges* encode partial rule instantiations, ‘waiting’ for additional (adjacent and passive) constituents to complete: $[1, 2, \text{VP} \rightarrow \text{V} \bullet \text{NP}]$;
- parser can fill in chart cells in any order and guarantee completeness.
An Example of a (Near-)Complete Chart

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NP → NP • PP</td>
<td>S → NP • VP</td>
<td>NP → kim •</td>
<td>S → NP VP •</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VP → V • NP</td>
<td>VP → VP • PP</td>
<td>VP → VP • PP</td>
<td>VP → VP • PP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>V → adores •</td>
<td>VP → V NP •</td>
<td>VP → V NP •</td>
<td>VP → V PP •</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NP → NP • PP</td>
<td>NP → snow •</td>
<td>NP → NP • PP</td>
<td>NP → NP • PP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PP → P • NP</td>
<td>PP → P NP •</td>
<td>PP → P NP •</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 Kim 1 adores 2 snow 3 in 4 Oslo 5
(Even) More Active Edges

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S → • NP VP</td>
<td>S → NP • VP</td>
<td></td>
<td>S → NP VP •</td>
</tr>
<tr>
<td></td>
<td>NP → • NP PP</td>
<td>NP → NP • PP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NP → • kim</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>VP → • VP PP</td>
<td>VP → V • NP</td>
<td>VP → VP • PP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VP → • V NP</td>
<td>V → adores •</td>
<td>VP → V NP •</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V → • adores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>NP → • NP PP</td>
<td>NP → NP • PP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NP → • snow</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Include all grammar rules as *epsilon* edges in each *chart* \([i, i]\) cell.
- after initialization, apply *fundamental rule* until fixpoint is reached.
Our ToDo List: Keeping Track of Remaining Work

The Abstract Goal
- Any chart parsing algorithm needs to check all pairs of adjacent edges.

A Naïve Strategy
- Keep iterating through the complete chart, combining all possible pairs, until no additional edges can be derived (i.e. the fixpoint is reached);
- frequent attempts to combine pairs multiple times: deriving ‘duplicates’.

An Agenda-Driven Strategy
- Combine each pair exactly once, viz. when both elements are available;
- maintain agenda of new edges, yet to be checked against chart edges;
- new edges go into agenda first, add to chart upon retrieval from agenda.
Backpointers: Keeping Track of the Derivation History

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2: S → • NP VP</td>
<td>10: S → 8 • VP</td>
<td>17: S → 8 15 •</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: NP → • NP PP</td>
<td>9: NP → 8 • PP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: NP → • kim</td>
<td>8: NP → kim •</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>5: VP → • VP PP</td>
<td>12: VP → 11 • NP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: VP → • V NP</td>
<td>16: VP → 15 • PP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: V → • adores</td>
<td>15: VP → 11 13 •</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7: NP → • NP PP</td>
<td>14: NP → 13 • PP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6: NP → • snow</td>
<td>13: NP → snow •</td>
<td></td>
</tr>
</tbody>
</table>

- Use edges to record derivation trees: backpointers to daughters;
- a single edge can represent multiple derivations: backpointer sets.
Chart Elements: The Edge Structure

\[
\# [ \text{id: (i-j) } \alpha \rightarrow \text{edge}_1 \ldots \text{edge}_i \cdot \beta_{i+1} \ldots \beta_n \{ \text{alternate}_1 \ldots \text{alternate}_n \}^* ]
\]

Components of the edge Structure

- \text{id}  unique edge identifier (automatically assigned my \text{make-edge}());
- \text{i} and \text{j}  starting and ending string index (chart vertices) for this edge;
- \text{\alpha}  category of this edge (from the set \text{C} of non-terminal symbols);
- \text{edge}_1 \ldots \text{edge}_i  (list of) daughter edges (for \text{\beta}_1 \ldots \text{\beta}_i) instantiated so far;
- \text{\beta}_{i+1} \ldots \text{\beta}_n  (list of) remaining categories in rule RHS to be instantiated;
- \text{alternate}_1 \ldots \text{alternate}_n  alternative derivation(s) for \text{\alpha} from \text{i} to \text{j}.

→ implemented using \text{defstruct()} plus suitable pretty printing routine.
Ambiguity Packing in the Chart

General Idea

- Maintain only one edge for each $\alpha$ from $i$ to $j$ (the ‘representative’);
- record alternate sequences of daughters for $\alpha$ in the representative.

Implementation

- Group passive edges into equivalence classes by identity of $\alpha$, $i$, and $j$;
- search chart for existing equivalent edge ($h$, say) for each new edge $e$;
- when $h$ (the ‘host’ edge) exists, pack $e$ into $h$ to record equivalence;
- $e$ not added to the chart, no derivations with or further processing of $e$; → unpacking multiply out all alternative daughters for all result edges.
Another Rerun: Local Variables

- Sometimes intermediate results need to be accessed more than once;
- `let()` and `let*()` create temporary value bindings for symbols, e.g;

  ```lisp
  ? (defparameter foo 42) → foo
  ? (let ((bar (+ foo 1))) bar) → 43
  ? bar → error
  ```

  
  ```lisp
  (let ((variable₁ sexp₁)
        ...
        (variableₙ sexpₙ))
      sexp ... sexp)
  ```

- bindings valid only in the body of `let()` (other bindings are shadowed);
- `let*()` binds sequentially, i.e. `variableᵢ` will be accessible for `variableᵢ+₁`. 
Suggested Background Activities

- Retrieve the model solution for the third exercise from the course site;
- compare our solution to your submission; how is ours better (or not)?
- compare the flawed chart of slide # 43 to the corrected version on # 50; identify the edge(s) that were erroneously omitted in the earlier version;
- write a testing `loop()`, gathering the call count data for the CKY parser;
- finally, (re-)read Chapters Nine and Ten of [Jurafsky & Martin, 2000];
- get the works on `loop()`, read Chapter Twenty Six of [Steele, 1990].