Ling/CSE 472: Introduction to Computational Linguistics

5/14/12
Parsing, unification, parsing with unification

Overview

- Earley parser
- CKY parser
- Unification
- Earley with unification

Parsing

- Given a CFG and a sentence, whether the CFG accepts it, with what and how many structures, is a mathematical fact
- Given a CFG and a sentence, determining whether the CFG accepts it, with what and how many structures, is a *search problem*
- Parsing algorithms can be:
  - Top-down or bottom-up
  - Breadth-first or depth-first
  - “Best”-first or exhaustive

The Earley Parser

- For a sentence of N words, the chart contains N+1 cells.
- Each cell contains a list of states.
- A state consists of: a local subtree (‘edge’), information about the degree of completion of that subtree, and information about how much of the string corresponds to the subtree.
- For example (in dotted-rule notation):
    - S → •VP, [0, 0]
    - NP → Det • Nominal, [1, 2]
    - VP → V NP•, [0, 3]

The Earley Chart Parser, Outline

- Add the initial S (gamma → •S, [0,0]) to the chart at position 0.
- Loop, for each of the rest of the cells in the chart:
    - If the state is incomplete, and the category to the right of the dot is not a POS, add (if not redundant) new states to the chart in the current position for each rule that expands that category. These new states all have the dot at the beginning of the rule, and a span that starts and ends at the current position. (PREDICTOR)

The Earley Chart Parser, Outline

- Loop (continued):
  - If the state is incomplete, and the category to the right of the dot ('B') is a POS, and B is a possible POS for the next word in the string, add the rule B → word • (if not redundant) to the next cell in the chart. (SCANNER)
  - If the state is complete (dot all the way to the right), look in the cell corresponding to the beginning of the current state's span for states which are currently seeking a daughter of the same category as the mother of this state. For each one of those, add a state (if not redundant) to the current cell, with the dot moved over one, and the span increased to the end of the current word. (COMPLETER)

Example: Kim adores snow in Oslo

- Toy grammar

$$S \rightarrow \text{NP VP}$$
$$\text{VP} \rightarrow \text{V NP}$$
$$\text{VP} \rightarrow \text{V}$$
$$\text{VP} \rightarrow \text{VP PP}$$
$$\text{PP} \rightarrow \text{P NP}$$
$$\text{NP} \rightarrow \text{NOM PP}$$
$$\text{NP} \rightarrow \text{NOM}$$

$$\text{NOM} \rightarrow \text{Kim}$$
$$\text{NOM} \rightarrow \text{snow}$$
$$\text{NOM} \rightarrow \text{Oslo}$$
$$\text{V} \rightarrow \text{adores}$$
$$\text{V} \rightarrow \text{snores}$$
$$\text{P} \rightarrow \text{in}$$

The Earley Chart Parser

- In which cell of the chart does one find the spanning edge(s)?
- Is the Earley algorithm top-down or bottom-up?
- Best-first or exhaustive?
- Breadth-first or depth-first?
- How does it handle ambiguity?
- How does it avoid inefficient re-parsing of subtrees?
- Returning the trees is still potentially exponential. How would this alogrithm return the trees?

CKY

- Make things simpler by assuming CNF (each rule is either binary, with two non-terminals as daughters, or unary, with a terminal as a daughter)
- Index spaces between words in the input: *0 Kim 1 adores 2 snow 3 in 4 Oslo 5*
- Recognition: Store edges in a two-dimensional table
  - Initialize lexical edges: For each A -> w grammar rule, put A in the cell corresponding to w
  - For each cell corresponding to larger strings: For every way of dividing the corresponding string in two, check whether the corresponding cells give categories that could be the daughters of a rule. If so, add the resulting category to the chart

CKY

- Parsing:
    - In addition to storing categories it the cells, store pointers to the daughers.
    - Read parse trees off starting from S in cell [0,n]
    - Returning all parses is exponential in the length of the sentence

Example: Kim adores snow in Oslo

- Toy grammar (CNF)

$$\begin{array}{ll}
\text{S} \rightarrow \text{NP VP} & \text{NOM} \mid \text{NP} \rightarrow \text{Kim} \\
\text{VP} \rightarrow \text{V NP} & \text{NOM} \mid \text{NP} \rightarrow \text{snow} \\
\text{VP} \rightarrow \text{VP PP} & \text{NOM} \mid \text{NP} \rightarrow \text{Oslo} \\
\text{PP} \rightarrow \text{P NP} & \text{V} \mid \text{VP} \rightarrow \text{adores} \\
\text{NP} \rightarrow \text{NOM PP} & \text{V} \mid \text{VP} \rightarrow \text{snores} \\
\text{NP} \rightarrow \text{NOM} & \text{P} \rightarrow \text{in}
\end{array}$$

Unification

- Input: Two feature structures
- Output: Failure signal or the unique most general feature structure containing all info from both inputs

Unification algorithm

- Use graphs to represent feature structures
- Augment these structures with another layer, adding features ‘pointer’ and ‘content’
- Use pointers to merge the graphs representing the two input feature structures (why?)
- Unification is recursive (why?)
- Unification is destructive (why?)

Unification example

$$\begin{bmatrix} \text{A} & \boxed{1}\begin{bmatrix} \text{B} & \text{c} \end{bmatrix} \\ \text{D} & \begin{bmatrix} \text{E} & \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{D} \begin{bmatrix} \text{E} \begin{bmatrix} \text{F} & \text{g} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- Represent each feature structure as a DAG
- Add the ‘pointer’ and ‘contents’ features
- Step through the unification algorithm to produce the result
- How would we have to alter this to handle typed feature structures?

Parsing with unification

- Associate feature structure constraints with rules
- Associate feature structures with edges
- Could just check after CFG parsing is done, but this is inefficient (why?)
- Instead: invoke unification when combining edges (COMPLETER)
- When deciding whether an edge to add is redundant, test is now *subsumption* (rather than identity): don't add edges that are *subsumed* by something already in the chart

Evaluation slide

- When we are evaluating parsing algorithms (rather than parsers incorporating specific grammars), what are we evaluating them for?
- What kinds of tests could we perform?
- What inputs do we need for the tests?

Overview

- Earley parser
- CKY parser
- Unification
- Earley with unification
- Next time: probabilistic parsing