Ling/CSE 472: Introduction to Computational Linguistics

5/4/17
Parsing

Reminders

- Revised project plan due tomorrow
- Assignment 4 is available

Overview

- Syntax v. parsing
- Earley
- CKY (briefly)
- Chart parsing (demo)
- Reading questions
- Summary questions

Parsing = making explicit structure that is inherent (implicit) in natural language strings

- What is that structure?
- Why would we need it?

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)

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?
- How does it handle ambiguity?
- How does it avoid inefficient re-parsing of subtrees?
- How would this algorithm return the trees?

CKY

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k,j]}
```

Figure 13.10 The CKY algorithm.

CKY

Noun, Nominal, Verb, VP, S [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Noun, Nominal [2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]
Book	*the*	*flight*	*through*	*Houston*

CKY

Noun, Nominal, Verb, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Noun, Nominal [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]
Book	*the*	*flight*	*through*	*Houston*

CKY Parsing

- In which cell of the chart does one find the spanning edge(s)?
- Is the CKY algorithm top-down or bottom-up?
- Best-first or exhaustive?
- How does it handle ambiguity?
- How does it avoid inefficient re-parsing of subtrees?
- How would this algorithm return the trees?

Chart parsing

- -> Demo with LKB

Summary questions

- How is natural language syntax and semantics different from that of programming languages?

- How does this affect the design of parsing algorithms and parsing systems more generally?

- How does this affect the application of parsing algorithms?

- How does computational syntax differ from theoretical syntax?

Reading questions

- Earley: How does the Scanner operator identify what parts of speech a current word can be. I understand that it checks the current word to the left of the dot against the expectation created by the Predictor operator, but how does it know if the current word can actually match the expected part of speech?

- At the top of page 435: "A backtracking approach expands the search space incrementally by systematically exploring one state at a time. The state chosen for expansion can be based on simple systematic strategies, such as depth-first or breadth-first methods..." So breadth-first and depth-first are both ways of implementing a backtracking strategy? How are "exploring all possible parse trees in parallel" and backtracking different from breadth-first and depth-first search, respectively?

Reading questions

- When I was reading the differences between top down parsing methods and bottom up methods, it seemed clear to me at first that bottom up should generally work faster as you can eliminate poor options much earlier in the process (i.e. when you go to combine a set of phrasal categories, you know what ways they can't work by the rules of the CFG). Is the reason this isn't the case because of local ambiguity? Because this could create multiple part of speech tags for the words and thus make us assume an entirely different set of phrasal categories.

- How does parsing works in case of movement?

Reading questions

- The CKY dynamic parsing method is described as requiring CNF in order to function, but this also causes problems in practice. What is the process for correcting these issues?

- I don't quite understand what the text means by CNF complicating semantic analysis. Does this mean X-bar (that's what CNF is, right?) is bad at semantic analysis?

- Finally, is there any benefit to using a CNF in the Earley Algorithm or does that just make the process longer and more convoluted?

Reading questions

- I understand that finite state rule based chunking (pg 452) isn't perfect, but wouldn't it be possible to end up with a structure that can't be a sentence (possibly due to words that can fall under multiple part of speech categories)? If the algorithm outputs something that is definitely false, is there any way to try to correct it?

- What is the purpose of "chunking", how is it separate from POS tagging, and why is it useful for parsing, despite lacking the use of recursion?

- What does chunking tell us other than the proportion of certain types of phrases to others in a sentence?

- Why does partial parsing help with resolving ambiguities?