Last Class: Parsing Intro

1. Grammars and parsing
2. Top-down and bottom-up parsing

Today: Chart Parsing

1. Chart parsers
2. Bottom-up chart parsing
3. The Earley Algorithm

Grammars and Parsing

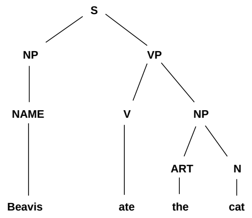
Need a **grammar:** a formal specification of the structures allowable in the language.

Need a **parser**: algorithm for assigning syntactic structure to an input sentence.

Sentence

Beavis ate the cat.

Parse Tree



General Parsing Strategies

Grammar	Top-Down	Bottom-Up
1. S → NP VP	S → NP VP	→ NAME ate the cat
2. VP → V NP	→ NAME VP	→ NAME V the cat
3. NP → NAME	→ Beav VP	→ NAME V ART cat
4. NP → ART N	→ Beav V NP	→ NAME V ART N
5. NAME → Beavis	→ Beav ate NP	→ NP V ART N
6. V → ate	→ Beav ate ART N	→ NP V NP
7. ART → the	→ Beav ate the N	→ NP VP
8. N → cat	→ Beav ate the cat	→ S

Problems with the Top-Down Parser

1. Only judges grammaticality.
2. Stops when it finds a single derivation.
3. No semantic knowledge employed.
4. No way to rank the derivations.
5. Problems with left-recursive rules.
6. Problems with ungrammatical sentences.

Efficient Parsing

The top-down parser is terribly inefficient.

> *Have the first year Phd students in the computer science department take the Q-exam.*
>
> *Have the first year Phd students in the computer science department taken the Q-exam?*

Chart Parsers

chart: data structure that stores partial results of the parsing process in such a way that they can be reused. The chart for an n-word sentence consists of:

- $n + 1$ **vertices**
- a number of **edges** that connect vertices

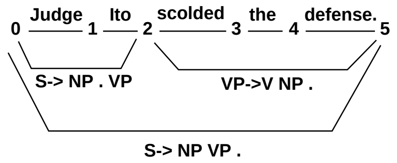


Chart Parsing: The General Idea

The process of parsing an n-word sentence consists of forming a chart with $n + 1$ vertices and adding edges to the chart one at a time.

- Goal: To produce a complete edge that spans from vertex 0 to n and is of category S.
- There is no backtracking.
- Everything that is put in the chart stays there.
- Chart contains all information needed to create parse tree.

Bottom-UP Chart Parsing Algorithm

Do until there is no input left:

1. If the agenda is empty, get next word from the input, look up word categories, add to agenda (as constituent spanning two postions).
2. Select a constituent from the agenda: constituent C from p_1 to p_2.
3. Insert C into the chart from position p_1 to p_2.
4. For each rule in the grammar of form $X \rightarrow C\ X_1 \ldots X_n$, add an active edge of form $X \rightarrow C \circ X_1 \ldots X_n$ from p_1 to p_2.

5. Extend existing edges that are looking for a C.
   (a) For any active edge of form $X \rightarrow X_1 \ldots \circ C X_n$ from p_0 to p_1, add a new active edge $X \rightarrow X_1 \ldots C \circ X_n$ from p_0 to p_2.
   (b) For any active edge of form $X \rightarrow X_1 \ldots X_n \circ C$ from p_0 to p_1, add a new (completed) constituent of type X from p_0 to p_2 to the agenda.

Grammar and Lexicon

Grammar:

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V NP

Lexicon:

the: ART

old: ADJ, N

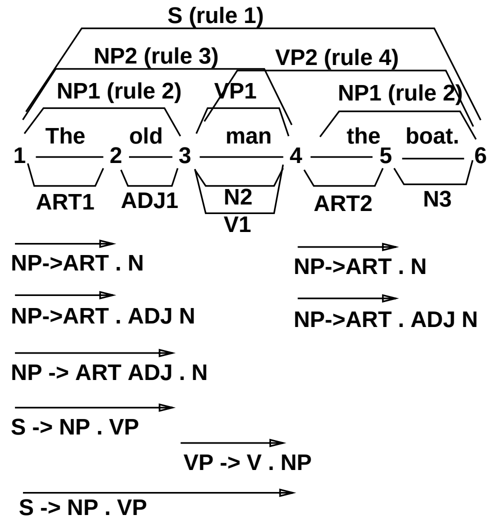
man: N, V

boat: N

Sentence: $_1$ The $_2$ old $_3$ man $_4$ the $_5$ boat $_6$

Example

[See .ppt slides]



Bottom-up Chart Parser

Is it any less naive than the top-down parser?

1. Only judges grammaticality.[fixed]
2. Stops when it finds a single derivation.[fixed]
3. No semantic knowledge employed.
4. No way to rank the derivations.
5. Problems with ungrammatical sentences.[better]
6. Terribly inefficient.

Slide CS474–24

Efficient Parsing

n = sentence length
Time complexity for naive algorithm: exponential in n
Time complexity for bottom-up chart parser: $\bigcirc(n^3)$

Options for improving efficiency:

1. Don't do twice what you can do once.
2. Don't represent distinctions that you don't need.

   Fall leaves fall and spring leaves spring.

3. Don't do once what you can avoid altogether.

   The can holds the water. ("can": AUX, V, N)

Slide CS474–25