NLP
Introduction to NLP

Earley Parser
Earley parser

• Problems with left recursion in top–down parsing
  – VP → VP PP

• Background
  – Developed by Jay Earley in 1970
  – No need to convert the grammar to CNF
  – Left to right

• Complexity
  – Faster than $O(n^3)$ in many cases
An Efficient Context-Free Parsing Algorithm

Jay Earley
University of California, Berkeley, California

A parsing algorithm which seems to be the most efficient general context-free algorithm known is described. It is similar to both Knuth's LR(k) algorithm and the familiar top-down algorithm. It has a time bound proportional to \( n^2 \) (where \( n \) is the length of the string being parsed) in general; it has an \( n^2 \) bound for unambiguous grammars; and it runs in linear time on a large class of grammars, which seems to include most practical context-free programming language grammars. In an empirical comparison it appears to be superior to the top-down and bottom-up algorithms studied by Griffiths and Petrick.

KEY WORDS AND PHRASES: syntax analysis, parsing, context-free grammar, compilers, computational complexity
CR CATEGORIES: 4.12, 5.22, 5.23

1. Introduction

Context-free grammars (BNF grammars) have been used extensively for describing the syntax of programming languages and natural languages. Parsing algorithms for context-free grammars consequently play a large role in the implementation of compilers and interpreters for pro-

* Computer Science Department. This work was partially supported by the Office of Naval Research under Contract No. NONR 0556(23) with the Computer Center, University of California, Berkeley, and by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-67-C-0020), monitored by the Air Force Office of Scientific Research.

Volume 13 / Number 2 / February, 1970
Earley Parser

- Looks for both full and partial constituents
- When reading word $k$, it has already identified all hypotheses that are consistent with words 1 to $k-1$
- Example:
  - $S [i,j] \rightarrow $ Aux . NP VP
  - NP $[j,k] \rightarrow $ N
  - $S [i,k] \rightarrow $ Aux NP . VP
Earley Parser

• It uses a dynamic programming table, just like CKY

• Example entry in column 1
  – [0:1] VP -> VP . PP
  – Created when processing word 1
  – Corresponds to words 0 to 1 (these words correspond to the VP part of the RHS of the rule)
  – The dot separates the completed (known) part from the incomplete (and possibly unattainable) part
Earley Parser

- Three types of entries
  - ‘scan’ – for words
  - ‘predict’ – for non-terminals
  - ‘complete’ – otherwise
Earley Parser Algorithm

```plaintext
function EARLEY-PARSE(words, grammar) returns chart

ENQUEUE((γ → • S, [0, 0]), chart[0])
for i ← from 0 to LENGTH(words) do
    for each state in chart[i] do
        if INCOMPLETE?(state) and
            NEXT-CAT(state) is not a part of speech then
            PREDICTOR(state)
        elseif INCOMPLETE?(state) and
            NEXT-CAT(state) is a part of speech then
            SCANNER(state)
        else
            COMPLETER(state)
    end
end
return(chart)
```

Figure from Jurafsky and Martin
Earley Parser Algorithm

procedure PREDICTOR((\(A \rightarrow \alpha \cdot B \beta, [i, j]\)))
  for each (\(B \rightarrow \gamma\)) in GRAMMAR-RULES-FOR(B, grammar) do
    ENQUEUE((\(B \rightarrow \cdot \gamma, [j, j]\)), chart[j])
  end

procedure SCANNER((\(A \rightarrow \alpha \cdot B \beta, [i, j]\)))
  if B \subseteq PARTS-OF-SPEECH(word[j]) then
    ENQUEUE((\(B \rightarrow \text{word}[j], [j, j + 1]\)), chart[j+1])
  end

procedure COMPLETER((\(B \rightarrow \gamma \cdot, [j, k]\)))
  for each (\(A \rightarrow \alpha \cdot B \beta, [i, j]\)) in chart[j] do
    ENQUEUE((\(A \rightarrow \alpha B \cdot \beta, [i, k]\)), chart[k])
  end

procedure ENQUEUE(state, chart-entry)
  if state is not already in chart-entry then
    PUSH(state, chart-entry)
  end

Figure from Jurafsky and Martin
S → NP VP
S → Aux NP VP
S → VP
NP → PRON
NP → Det Nom
Nom → N
Nom → Nom N
Nom → Nom PP
PP → PRP NP
VP → V
VP → V NP
VP → VP PP

Det → 'the'
Det → 'a'
Det → 'this'
PRON → 'he'
PRON → 'she'
N → 'book'
N → 'boys'
N → 'girl'
PRP → 'with'
PRP → 'in'
V → 'takes'
V → 'take'
Example created using NLTK
take this book
<table>
<thead>
<tr>
<th>take</th>
<th>this</th>
<th>book</th>
</tr>
</thead>
</table>

```
S  -> * NP VP
S  -> * Aux NP VP
S  -> * VP
VP -> * V
VP -> * V NP
VP -> * VP PP
V  -> * 'take'
NP -> * PRON
NP -> * Det Nom
```
(S (VP (V take) (NP (Det this) (Nom (N book))))))
NLTK Demo

- nltk demo:

```python
import nltk
nltk.parse.chart.demo(2, print_times=False, trace=1, sent='I saw a dog', numparses=1)
```
Notes

• CKY fills the table with phantom constituents
  – problem, especially for long sentences
• Earley only keeps entries that are consistent with the input up to a given word
• So far, we only have a recognizer
  – For parsing, we need to add backpointers
• Just like with CKY, there is no disambiguation of the entire sentence
• Time complexity
  – $n$ iterations of size $O(n^2)$, therefore $O(n^3)$
  – For unambiguous grammars, each iteration is of size $O(n)$, therefore $O(n^2)$
NLP