CMPT-825

Natural Language Processing

Anoop Sarkar

http://www.cs.sfu.ca/~anoop

Why are parsing algorithms important?

- A linguistic theory is implemented in a formal system to generate the set of grammatical strings and rule out ungrammatical strings.
- Such a formal system has computational properties.
- One such property is a simple decision problem: given a string, can it be generated by the formal system *(recognition)*.
- If it is generated, what were the steps taken to recognize the string *(parsing)*.

Why are parsing algorithms important?

- Consider the recognition problem: find algorithms for this problem for a particular formal system.
- The algorithm must be decidable.
- Preferably the algorithm should be polynomial: enables computational implementations of linguistic theories.
- Elegant, polynomial-time algorithms exist for formalisms like CFG

A recognition algorithm for CFGs

- Consider the CFG G:

  1. $S \rightarrow S\ S$
  2. $S \rightarrow a$

  $L(G) = a^i$ for $i >= 1$

- The recognition question: does the string aaa belong to $L(G)$?
  - Input: aaa
  - Output: $\{$*yes*, *no*$\}$

Parsing algorithm for CFGs

- If the answer is *yes* then parsing involves extraction of the parse tree (the *proof* of why the string was accepted)
- Similar to the extraction of the min edit distance alignment
- Just as in that case, we have to extract all possible parse trees (just as we could extract multiple alignments)

Top-down, depth-first, left to right parsing

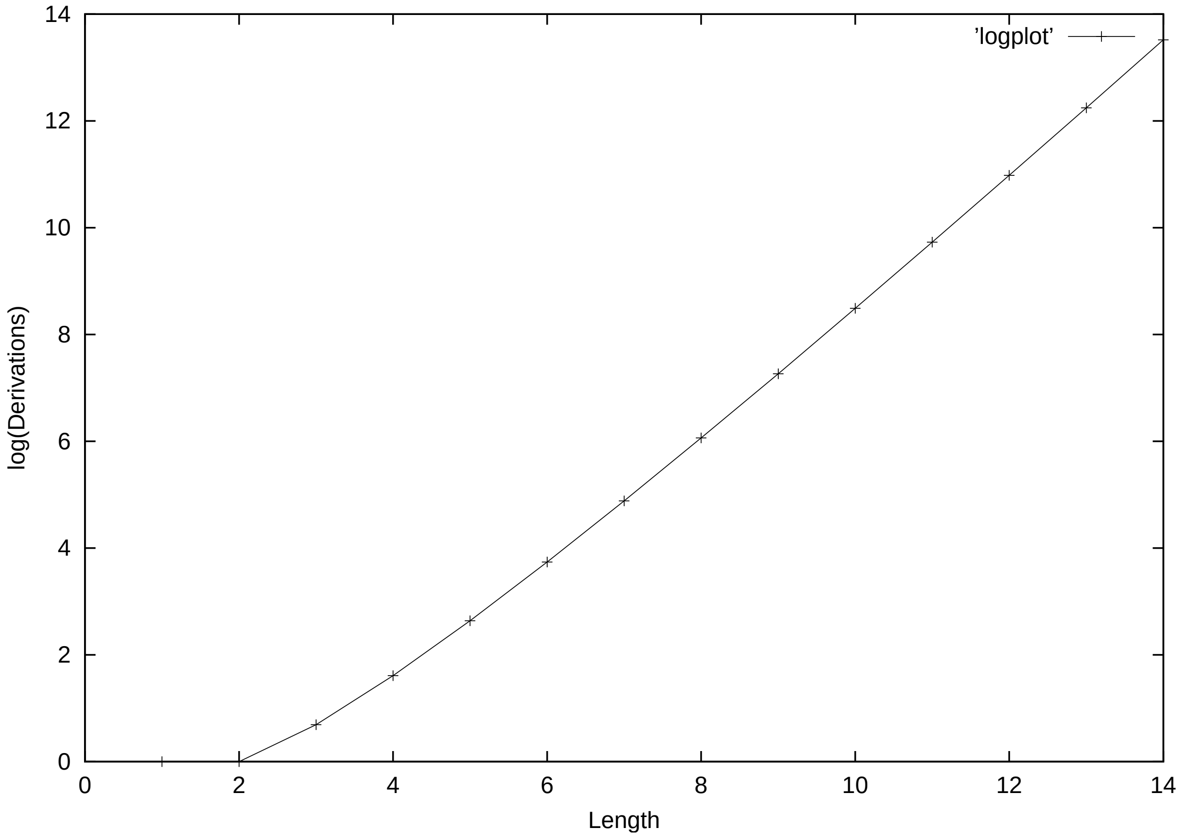
$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
NP & \rightarrow & Det\ N\ PP \\
VP & \rightarrow & V \\
VP & \rightarrow & V\ NP \\
VP & \rightarrow & V\ NP\ PP \\
PP & \rightarrow & P\ NP \\
NP & \rightarrow & I \\
Det & \rightarrow & a \mid the \\
V & \rightarrow & saw \\
N & \rightarrow & park \mid dog \mid man \mid telescope \\
P & \rightarrow & in \mid with
\end{array}
$$

Top-down, depth-first, left to right parsing

- Consider the input string: *the dog saw a man in the park*
- S . . . (S (NP VP)) . . . (S (NP Det N) VP) . . . (S (NP (Det the) N) VP) . . . (S (NP (Det the) (N dog)) VP) . . .
- (S (NP (Det the) (N dog)) VP) . . . (S (NP (Det the) (N dog)) (VP V NP PP)) . . . (S (NP (Det the) (N dog)) (VP (V saw) NP PP)) . . .
- (S (NP (Det the) (N dog)) (VP (V saw) (NP Det N) PP)) . . .
- (S (NP (Det the) (N dog)) (VP (V saw) (NP (Det a) (N man)) (PP (P in) (NP (Det the) (N park)))))

Number of derivations grows exponentially

e.g. $L(G) = a+$ using CFG rules $\{ S \rightarrow S\ S, S \rightarrow a \}$



Syntactic Ambiguity: (Church and Patil 1982)

- Algebraic character of parse derivations

- Power Series for grammar for coordination (more general than PPs):

  $\texttt{NP} \rightarrow \texttt{cabbages} \mid \texttt{kings} \mid \texttt{NP and NP}$

```
NP = cabbages + cabbages and kings
     + 2 (cabbages and cabbages and kings)
     + 5 (cabbages and kings and cabbages and kings)
     + 14 ...
```

Syntactic Ambiguity: (Church and Patil 1982)

- Coefficients equal the number of parses for each NP string

- These ambiguity coefficients are Catalan numbers:

$$Cat(n) = \binom{2n}{n} - \binom{2n}{n-1}$$

- $\binom{a}{b}$ is the *binomial coefficient*

$$\binom{a}{b} = \frac{a!}{(b!(a-b)!)}$$

Syntactic Ambiguity: (Church and Patil 1982)

- Why Catalan numbers? Cat(n) is the number of ways to parenthesize an expression of length n with two conditions:

  1. there must be equal numbers of open and close parens

  2. they must be properly nested so that an open precedes a close

- So the first term counts $2n$ parens with equal number of open and close, while the second term subtracts those that are not properly nested:

$$Cat(n) = \binom{2n}{n} - \binom{2n}{n-1}$$

Syntactic Ambiguity: (Church and Patil 1982)

- $Cat(n)$ also provides exactly the number of parses for the sentence:

```
John saw the man
         on the hill
         with the telescope
```

in the above sentence there are 2 PPs, so number of parse trees = Cat$(2 + 1)$ = 5

with 8 PPs: Cat(9) $= 4862$ parse trees

- Other sub-grammars are simpler:

$$
\begin{aligned}
ADJP &\rightarrow adj\ ADJP \mid \epsilon \\
ADJP &= 1 + adj + adj^2 + adj^3 + \ldots \\
ADJP &= \frac{1}{1-adj}
\end{aligned}
$$

Syntactic Ambiguity: (Church and Patil 1982)

- Now consider power series of combinations of sub-grammars:

  `S = NP · VP`

  ```
  ( The number of products over sales ... )
  ( is near the number of sales ... )
  ```

- Both the `NP` subgrammar and the `VP` subgrammar power series have Catalan coefficients

Syntactic Ambiguity: (Church and Patil 1982)

- The power series for the `S` $\rightarrow$ `NP VP` grammar is the multiplication:

$$(N \sum_i Cat_i (P N)^i) \cdot (is \sum_j Cat_j (P N)^j)$$

- In a parser for this grammar, this leads to a cross-product:

$$L \times R = \{(l, r) \mid l \in L \;\&\; r \in R \}$$

Syntactic Ambiguity: (Church and Patil 1982)

- A simple change:

```
Is ( The number of products over sales ... )
   ( near the number of sales ... )
```

$$
\begin{aligned}
&= \texttt{Is}\ N \sum_i Cat_i\,(\,P\ N\,)^i)\ \cdot\ (\sum_j Cat_j(\,P\ N\,)^j) \\
&= \texttt{Is}\ N \sum_i \sum_j Cat_i\ Cat_j(\,P\ N\,)^{i+j} \\
&= \texttt{Is}\ N \sum_{i+j} Cat_{i+j+1}\,(\,P\ N\,)^{i+j}
\end{aligned}
$$

Syntactic Ambiguity and Parsing

- Clearly, a top-down parser will take exponential time (since there are exponentially many parses)

- How can we deal with this problem? *Dynamic programming*

- Store an item which corresponds to a multiple parses, which encapsulates all the ambiguity for a sub-parse

- Now we combine these items in subsequent steps

Chomsky Normal Form (CNF)

- *Every* CFG can be converted such that all rules are either of the form $A \rightarrow B\,C$ or $A \rightarrow a$, where A, B, C are non-terminals (not necessarily distinct) and a is a terminal symbol
  - ϵ removal: $A \rightarrow B\,C$, $C \rightarrow \epsilon \mid C\,D \mid a \ldots A \rightarrow B \mid B\,C\,D \mid B\,a$
  - eliminate chain rules: $A \rightarrow B\,C \mid C\,D\,C$, $C \rightarrow D \ldots$
    $A \rightarrow B\,D \mid D\,D\,D$
  - eliminate terminals from right hand sides with non-terminals:
    $A \rightarrow B\,a\,C\,d \ldots$
    $A \rightarrow B\,N_1\,C\,N_2$, $N_1 \rightarrow a$, $N_2 \rightarrow d$
  - binarize right hand side: $A \rightarrow B\,C\,D\,E \ldots$

$$A \to B\ N_3,\ N_3 \to C\ N_4,\ N_4 \to D\ E$$

Dynamic Programming and Context-Free Parsing (Cocke-Younger-Kasami: CYK algorithm)

$\{ S \rightarrow S\ S\ , S \rightarrow a \}$

a	a	a	a
$S_{0,1} \rightarrow$ a	$S_{1,2} \rightarrow$ a	$S_{2,3} \rightarrow$ a	$S_{3,4} \rightarrow$ a
$S_{0,1} + S_{1,2}$ $= S_{0,2} \rightarrow$ S S	$S_{1,2} + S_{2,3}$ $= S_{1,3} \rightarrow$ S S	$S_{2,3} + S_{3,4}$ $= S_{2,4} \rightarrow$ S S	
$S_{0,1} + S_{1,3}$ OR $S_{0,2} + S_{2,3}$ $= S_{0,3} \rightarrow$ S S	$S_{1,2} + S_{2,4}$ OR $S_{1,3} + S_{3,4}$ $= S_{1,4} \rightarrow$ S S		
What goes in this cell? ?? $= S_{0,4}$			

Shift-Reduce Parsing

- Every CFG has an equivalent pushdown automata: a finite state machine which has additional memory in the form of a stack
  - Consider the grammar: $NP \rightarrow Det\ N$, $Det \rightarrow$ *the*, $N \rightarrow$ *dog*
  - Consider the input: *the dog*
  - shift the first word *the* into the stack, check if the top n symbols in the stack matches the right hand side of a rule in which case you can **reduce** that rule, or optionally you can shift another word into the stack
  - reduce using the rule $Det \rightarrow$ *the*, and push Det onto the stack

– shift *dog*, and then reduce using $N \rightarrow$ *dog* and push N onto the stack

– the stack now contains *Det, N* which matches the rhs of the rule $NP \rightarrow Det\ N$ which means we can reduce using this rule, pushing NP onto the stack

– If NP is the start symbol and since there is no more input left to shift, we can accept the string

Shift-Reduce Parsing

- Sometimes humans can be “led down the garden-path” when processing a sentence (from left to right)

- Such garden-path sentences lead to a situation where one is forced to backtrack because of a commitment to only one out of many possible derivations

- For example, in the sentence *The horse raced past the barn fell*, once you process the word *fell* you are forced to reanalyze the previous word *raced* as being a verb inside a *relative clause*: *raced past the barn*, meaning *the horse that was raced past the barn*

- Notice however that other examples with the same structure but different words do not behave the same way. For example: *the flowers delivered to the patient arrived*

Earley Parsing

- A *dotted rule* is a way to get around the explicit conversion of a CFG to Chomsky Normal Form
- Since natural language grammars are quite large, and are often modified to be able to parse more data, avoiding the explicit conversion to CNF is an advantage
- A dotted rule denotes that the right hand side of a CF rule has been partially recognized/parsed

Earley Parsing

- $S \rightarrow \bullet NP\ VP$ indicates that once we find an NP and a VP we have recognized an S

- $S \rightarrow NP\ \bullet\ VP$ indicates that we've recognized an NP and we need a VP

- $S \rightarrow NP\ VP\ \bullet$ indicates that we have a complete S

- Consider the dotted rule $S \rightarrow \bullet NP\ VP$ and assume our CFG contains a rule $NP \rightarrow John$
  Because we have such an NP rule we can **predict** a new dotted rule
  $NP \rightarrow \bullet\ John$

Earley Parsing

- If we have the dotted rule: $NP \rightarrow \bullet\ John$ and the next input symbol on our *input tape* is the word *John* we can **scan** the input and create a new dotted rule $NP \rightarrow John\ \bullet$

- Consider the dotted rule $S \rightarrow \bullet NP\ VP$ and $NP \rightarrow John\ \bullet$
  Since NP has been completely recognized we can **complete**
  $S \rightarrow NP\ \bullet\ VP$

- These three steps: *predictor*, *scanner* and *completer* form the *Earley parsing algorithm* and can be used to parse using any CFG without conversion to CNF
  Note that we have not accounted for ϵ in the *scanner*

Earley Parsing

- A *state* is a dotted rule plus a span over the input string, e.g. $(S \rightarrow NP \bullet VP, [4,8])$ implies that we have recognized an NP

- We store all the states in a *chart* – typically, in *chart[i]* we store all states of the form: $(A \rightarrow \alpha \bullet \beta, [i, j])$ *or* states of the form: $(A \rightarrow \alpha \bullet \beta, [j, i])$, where $\alpha, \beta \in (N \cup T)^*$

- Note that $(S \rightarrow NP \bullet VP, [0,8])$ implies that in the chart there are two states $(NP \rightarrow \alpha \bullet, [0,8])$ and $(S \rightarrow \bullet NP\ VP, [0,0])$ — this is the *completer* rule, the heart of the Earley parser

- Also if we have state $(S \rightarrow \bullet\, NP\ VP, [0,0])$ in the chart, then we always *predict* the state $(NP \rightarrow \bullet\, \alpha, [0,0])$ for all rules $NP \rightarrow \alpha$ in the grammar

Earley Parsing

$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \mid NP\ PP \mid John \\
Det & \rightarrow & the \\
N & \rightarrow & cookie \mid table \\
VP & \rightarrow & VP\ PP \mid V\ NP \mid V \\
V & \rightarrow & ate \\
PP & \rightarrow & P\ NP \\
P & \rightarrow & on
\end{array}
$$

Consider the input: *0 John 1 ate 2 on 3 the 4 table 5*

What can we predict from the state $(S \rightarrow \bullet\ NP\ VP, [0, 0])$?

What can we complete from the state $(V \rightarrow ate\ \bullet, [1, 2])$?