Faculty of Computer Science, Dalhousie University
27-Nov-2024

CSCI 4152/6509 — Natural Language Processing

Lecture 21: CYK Algorithm and PCFGs

Location: Carleton Tupper Building Theatre C
Instructor: Vlado Keselj
Time: 16:05 – 17:25

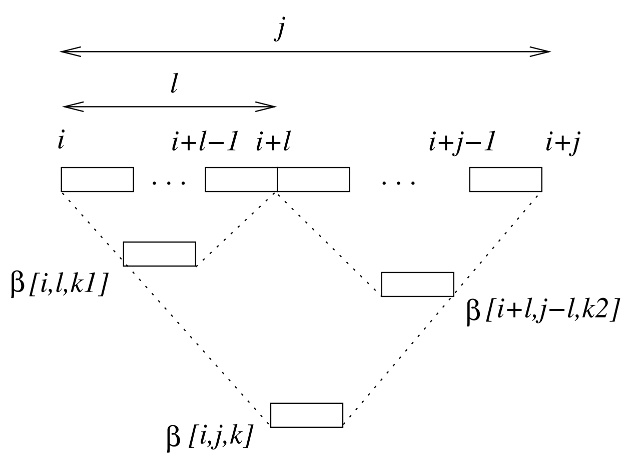
Previous Lecture

- Phrase structure in English (continued):
  - NP, VP, PP, ADJP, ADVP
- Heads and dependency, dependency tree
- **CYK Chart Parsing Algorithm**
- Chomsky Normal Form (CNF)
- CYK algorithm example

Implementation

The example implies that we need to use a two-dimensional table to store chart entries. Using a two-dimensional table is a possible solution, but in that case the table entries would be quite complex since each of them needs to store a set of non-terminals. To make the solution simpler, we can use a three-dimensional table, such that the third dimension corresponds to all different non-terminals.

Explanation of Index Use in CYK



CYK Algorithm

Let all nonterminals be: $N^1, \ldots N^m$.

In the standard CYK algorithm, we have a two dimensional table β in which only the entries β_{ij}, $1 \le i \le i+j-1 \le n$, are used. Each entry β_{ij} contains a set of nonterminals that can produce substring $w_i \ldots w_{i+j-1}$ using the grammar rules, i.e., $\beta_{ij} = \{N | N \Rightarrow^* w_i \ldots w_{i+j-1}\}$.

If we enumerate all nonterminals: N^1, N^2, ..., N^m, then each set of nonterminals β_{ij} can be represented by extending β to be a 3-dimensional table β_{ijk}, in which $\beta_{ijk} = 1$ means that N^k can produce substring $w_i \ldots w_{i+j-1}$, and $\beta_{ijk} = 0$ that it cannot.

Algorithm 1 CYK Parsing Algorithm

Require: sentence $= w_1 \ldots w_n$, and a CFG in CNF with nonterminals $N^1 \ldots N^m$,
  N^1 is the start symbol
Ensure: parsed sentence

- allocate matrix $\beta \in \{0,1\}^{n \times n \times m}$ and initialize all entries to 0
- **for** $i \leftarrow 1$ to n **do**
  - **for all** rules $N^k \rightarrow w_i$ **do**
    - $\beta[i,1,k] \leftarrow 1$
- **for** $j \leftarrow 2$ to n **do**
  - **for** $i \leftarrow 1$ to $n-j+1$ **do**
    - **for** $l \leftarrow 1$ to $j-1$ **do**
      - **for all** rules $N^k \rightarrow N^{k_1} N^{k_2}$ **do**
        - $\beta[i,j,k] \leftarrow \beta[i,j,k]$ OR $(\beta[i,l,k_1]$ AND $\beta[i+l,j-l,k_2])$
- **return** $\beta[1,n,1]$

The line $\beta[i,j,k] \leftarrow \beta[i,j,k]$ OR $(\beta[i,l,k_1]$ AND $\beta[i+l,j-l,k_2])$ in the algorithm is essentially is a shorthand expression for:

if $\beta[i,l,k_1]$ AND $\beta[i+l,j-l,k_2]$ **then**
  $\beta[i,j,k] \leftarrow 1$

26 Efficient Inference in PCFG Model

Efficient Inference in PCFG Model

Let us consider the marginalization task:

P(sentence) =?

If 'sentence' is the following sequence of words: $w_1 w_2 \ldots w_n$, then P(sentence) is the following conditional probability:

$$\mathrm{P}(\text{sentence}) = \mathrm{P}(w_1 w_2 \ldots w_n | S)$$

i.e., it is the probability of generating the sentence given that we start from S, i.e, it is $\mathrm{P}(S \Rightarrow^* w_1 \ldots w_n)$.

An obvious way to calculate this marginal probability is to find all parse trees of a sentence and sum their probabilities, i.e:

$$\mathrm{P}(\text{sentence}) = \sum_{t \in T} \mathrm{P}(t),$$

where T is the set of all parse trees of the sentence 'sentence'. However, this may be very inefficient. We also need a way to find all parse trees.

As an example illustrating that the above direct approach may lead to an exponential algorithm, consider a CFG with only two rules $S \Rightarrow S\ S$ and $S \Rightarrow a$. The sentences a^n have as many parse trees as there are binary trees with n leaves, which is a well-known Catalan number, $\approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$ as $n \rightarrow \infty$.

An algorithm for efficient marginalization can be derived from the CYK algorithm.

PCFG Marginalization

The CKY algorithms is adapted to solve the problem of efficient PCFG marginalization, but replacing entries of the table β with numbers between 0 and 1. These numbers are called inside probabilities, and they represent the following probabilities:

$$\beta[i, j, k] = \mathrm{P}(w_i \dots w_{i+j-1} | N^k)$$

So, $\beta[i, j, k]$ is the probability that the string $w_i \dots w_{i+j-1}$ is generated in a derivation where the starting nonterminal is N^k. Algorithm 2 is the probabilistic CYK algorithm for calculating $\mathrm{P}(\text{sentence})$.

Algorithm 2 Probabilistic CYK for $\mathrm{P}(\text{sentence})$

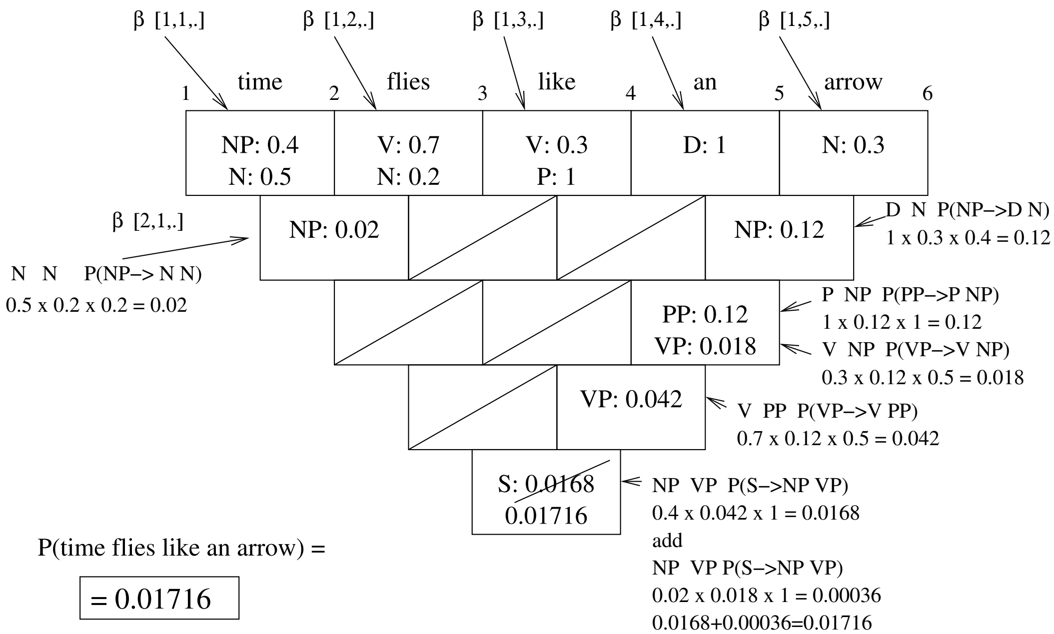
Require: sentence $= w_1 \dots w_n$, and a PCFG in CNF with nonterminals $N^1 \dots N^m$, N^1 is the start symbol
Ensure: P(sentence) is returned

allocate $\beta \in \mathbb{R}^{n \times n \times m}$ and initialize all entries to 0
for $i \leftarrow 1$ to n **do**
  for all rules $N^k \rightarrow w_i$ **do**
    $\beta[i, 1, k] \leftarrow \mathrm{P}(N^k \rightarrow w_i)$
for $j \leftarrow 2$ to n **do**
  for $i \leftarrow 1$ to $n - j + 1$ **do**
    for $l \leftarrow 1$ to $j - 1$ **do**
      for all rules $N^k \rightarrow N^{k_1} N^{k_2}$ **do**
        $\beta[i, j, k] \leftarrow \beta[i, j, k] +\ \mathrm{P}(N^k \rightarrow N^{k_1} N^{k_2}) \cdot \beta[i, l, k_1] \cdot \beta[i + l, j - l, k_2]$
return $\beta[1, n, 1]$

PCFG Marginalization Example (grammar)

S	$\rightarrow$	NP VP	/1	VP	$\rightarrow$	V NP	/.5	N	$\rightarrow$	time	/.5
NP	$\rightarrow$	time	/.4	VP	$\rightarrow$	V PP	/.5	N	$\rightarrow$	arrow	/.3
NP	$\rightarrow$	N N	/.2	PP	$\rightarrow$	P NP	/1	N	$\rightarrow$	flies	/.2
NP	$\rightarrow$	D N	/.4					D	$\rightarrow$	an	/1
V	$\rightarrow$	like	/.3								
V	$\rightarrow$	flies	/.7								
P	$\rightarrow$	like	/1								

PCFG Marginalization Example (chart)



Conditioning

The conditioning computational problem in the PCFG model becomes the task of finding the conditional probability $\mathrm{P}(\text{tree}|\text{sentence})$, for a particular sentence and a particular parse three of the given sentence. Using the definition of the conditional probability, we have:

$$\mathrm{P}(\text{tree}|\text{sentence}) = \frac{\mathrm{P}(\text{tree}, \text{sentence})}{\mathrm{P}(\text{sentence})}$$

and since the sentence is a part of the parse tree, we can further write:

$$\mathrm{P}(\text{tree}|\text{sentence}) = \frac{\mathrm{P}(\text{tree}, \text{sentence})}{\mathrm{P}(\text{sentence})} = \frac{\mathrm{P}(\text{tree})}{\mathrm{P}(\text{sentence})}$$

Slide notes:

Conditioning

- Conditioning in the PCFG model: $\mathrm{P}(\text{tree}|\text{sentence})$
- Use the formula:

$$\mathrm{P}(\text{tree}|\text{sentence}) = \frac{\mathrm{P}(\text{tree}, \text{sentence})}{\mathrm{P}(\text{sentence})} = \frac{\mathrm{P}(\text{tree})}{\mathrm{P}(\text{sentence})}$$

- $\mathrm{P}(\text{tree})$ — directly evaluated
- $\mathrm{P}(\text{sentence})$ — marginalization

$\mathrm{P}(\text{tree})$ is calculated by multiplying probabilities of all rules in the tree, and $\mathrm{P}(\text{sentence})$ is calculated by the Algorithm 2 used for marginalization.

Completion

The completion task becomes the parsing problem; i.e., the problem of finding the most probably parse tree give the sentence, which can be expressed as:

$$\arg\max_{\text{tree}} \mathrm{P}(\text{tree}|\text{sentence})$$

Slide notes:

Completion

- Finding the most likely parse tree of a sentence:

$$\arg\max_{\text{tree}} \mathrm{P}(\text{tree}|\text{sentence})$$

- Use the CYK algorithm in which line 9 is replaced with:
  9: $\beta[i,j,k] \leftarrow \max(\beta[i,j,k], \mathrm{P}(N^k \rightarrow N^{k_1}N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l,j-l,k_2])$
- Return the most likely tree

The most probable completion is computed in a similar way to the marginalization algorithm (Algorithm 2). The difference is that the line 9 is replaced by the line

9: $\beta[i,j,k] \leftarrow \max(\beta[i,j,k], \mathrm{P}(N^k \rightarrow N^{k_1}N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l,j-l,k_2])$

Additionally in step 10, we are not just interested in $\beta[1,n,1]$, which is the probability of the most probable tree, but we also want to obtain the actual tree.

Algorithm 3 CYK-based Completion Algorithm for $\arg\max_t \mathrm{P}(t|\text{sentence})$

Require: sentence $= w_1 \dots w_n$, and a PCFG in CNF with nonterminals $N^1 \dots N^m$, N^1 is the start symbol
Ensure: The most likely parse tree is returned

1: allocate $\beta \in \mathbb{R}^{n\times n\times m}$ and initialize all entries to 0
2: **for** $i \leftarrow 1$ to n **do**
3:     **for all** rules $N^k \rightarrow w_i$ **do**
4:         $\beta[i,1,k] \leftarrow \mathrm{P}(N^k \rightarrow w_i)$
5: **for** $j \leftarrow 2$ to n **do**
6:     **for** $i \leftarrow 1$ to $n-j+1$ **do**
7:         **for** $l \leftarrow 1$ to $j-1$ **do**
8:             **for all** rules $N^k \rightarrow N^{k_1}N^{k_2}$ **do**
9:                 $\beta[i,j,k] \leftarrow \max(\beta[i,j,k], \mathrm{P}(N^k \rightarrow N^{k_1}N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l,j-l,k_2])$
10: **return** Reconstruct$(1,n,1,\beta)$

The tree can be reconstructed from the table using algorithm 4:

10: **return** Reconstruct$(1,n,1,\beta)$

PCFG Completion Example (grammar)

S	$\rightarrow$	NP VP	/1	VP	$\rightarrow$	V NP	/.5	N	$\rightarrow$	time	/.5
NP	$\rightarrow$	time	/.4	VP	$\rightarrow$	V PP	/.5	N	$\rightarrow$	arrow	/.3
NP	$\rightarrow$	N N	/.2	PP	$\rightarrow$	P NP	/1	N	$\rightarrow$	flies	/.2
NP	$\rightarrow$	D N	/.4					D	$\rightarrow$	an	/1
V	$\rightarrow$	like	/.3								
V	$\rightarrow$	flies	/.7								
P	$\rightarrow$	like	/1								

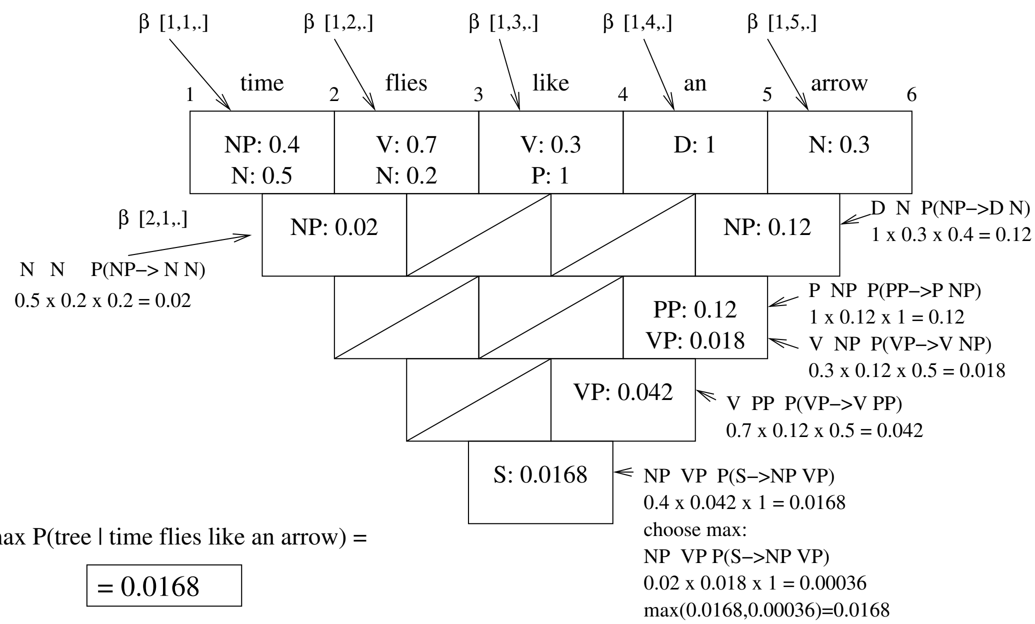
Algorithm 4 Reconstruct(i, j, k, β)

Require: β — table from CYK, i — index of the first word, j — length of sub-string sentence, k — index of non-terminal

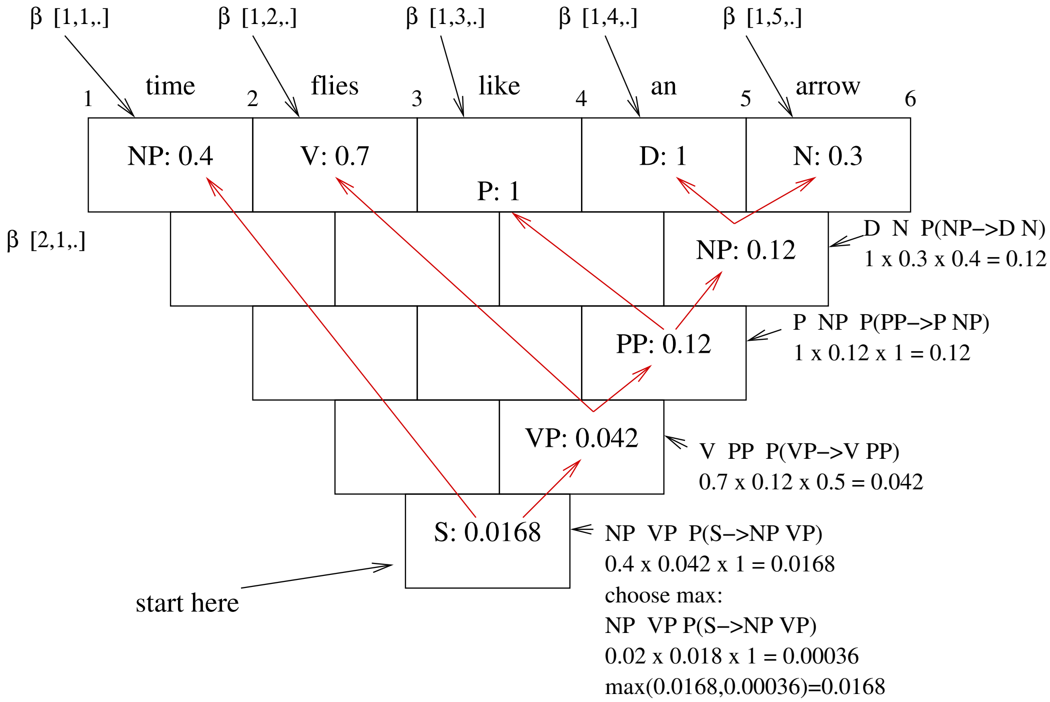
Ensure: a most probable tree with root N^k and leaves $w_i \ldots w_{i+j-1}$ is returned

if $j = 1$ **then**
    return tree with root N^k and child w_i
for $l \leftarrow 1$ to $j-1$ **do**
    for all rules $N^k \to N^{k_1} N^{k_2}$ **do**
        if $\beta[i, j, k] = \mathrm{P}(N^k \to N^{k_1} N^{k_2}) \cdot \beta[i, l, k_1] \cdot \beta[i+l, j-l, k_2]$ **then**
            create a tree t with root N^k
            t.*left_child* $\leftarrow$ Reconstruct(i, l, k_1, β)
            t.*right_child* $\leftarrow$ Reconstruct$(i+l, j-l, k_2, \beta)$
            return t

PCFG Completion Example (chart)

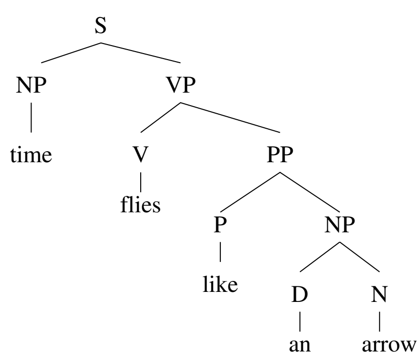


PCFG Completion Example (tree reconstruction)



PCFG Completion Example (final tree)

The most probable three:



Topics related to PCFGs

– An interesting open problem is whether the inference in PCFGs can be reduced to a message-passing-style algorithm as used in Bayesian Networks?

Issues with PCFGs

The Probabilistic Context-Free Grammars were shown to perform quite well in parsing English, but usually with some additional mechanisms to address certain issues. Two most prominent issues in using PCFGs to parse nature languages are the inability of PCFGs to capture structural and lexical dependencies.

Structural dependencies are rule dependencies on the position in a parse tree. For example, pronouns occur more frequently as subjects than objects in sentences, so the rule choice between NP $\rightarrow$ PRP and NP $\rightarrow$ DT NN should depend on the position of a noun phrase in a tree. Generally, NL parse trees are usually deeper at their right side than the left side, and this propertly is typically not modeled well with PCFGs.

Lexical dependencies are rule dependencies on the words that are eventually derived from those rules, particularly phrase head words. As an example, the PP-attachment problem is resolved based on the rule probabilities of the rules applied higher in the parse tree, such as NP $\rightarrow$ NP PP and VP $\rightarrow$ VB NP PP, while they truly frequently depend on the verb being used and other word, particularly head words.

Slide notes:

PP-Attachment Example

- Consider sentences:
  - "Workers dumped sacks into a bin." and
  - "Workers dumped sacks of fish."
- and rules:
  - NP $\rightarrow$ NP PP
  - VP $\rightarrow$ VBD NP
  - VP $\rightarrow$ VBD NP PP

As an example, let us consider simple sentences:

- "Workers dumped sacks into a bin." (from [JM]), and
- "Workers dumped sacks of fish."

At some level of parsing, we can see both of these sentences as:

- NP VBD NP PP

and now the question is whether the "NP PP" should be combined to make an NP, or the sequence "VBD NP PP" should be combined to make a VP. In a PCFG this will depend only on the probability of the rules: NP $\rightarrow$ NP PP, VP $\rightarrow$ VBD NP, and VP $\rightarrow$ VBD NP PP. However, we can see that the probailities should actually depend on affinity of the verb 'dump' and preposition 'into' on one side, and the noun 'sacks' and the preposition 'of' on other side.

A Solution: Probabilistic Lexicalized CFGs

- use heads of phrases
- expanded set of rules, e.g.:

$$\text{VP(dumped)} \rightarrow \text{VBD(dumped) NP(sacks) PP(into)}$$

- large number of new rules
- sparse data problem
- solution: new independence assumptions
- proposed solutions by Charniak, Collins, etc. around 1999