Probabilistic Context Free Grammars and the CYK Algorithm

Gideon Maillette de Buy Wenniger
Statistical Language Processing and Learning Lab at the Institute for Logic Language and Computation (ILLC), University of Amsterdam, the Netherlands

Contact:
gemdbw - at - gmail - dot -com

Scientific knowledge is organized in levels, not because reduction is impossible but because nature is organized in levels, and the pattern at each level is most clearly discerned by abstracting from the detail to the levels far below. (The pattern of a halftone does not become clearer when we magnify it so the individual spots of ink become visible.) And nature is organized in levels because hierarchic structure systems of Chinese boxes provide the most viable form for any system of even moderate complexity.

Herbert A.Simon (1973). The organization of Complex Systems.

Overview

- Probabilistic Context Free Grammars : Motivation and Formal Definition
- Trees and Treebanks, relation between Trees and Rules
- Parsing: A general description
- Parsing : The CYK Parser

Motivating Example

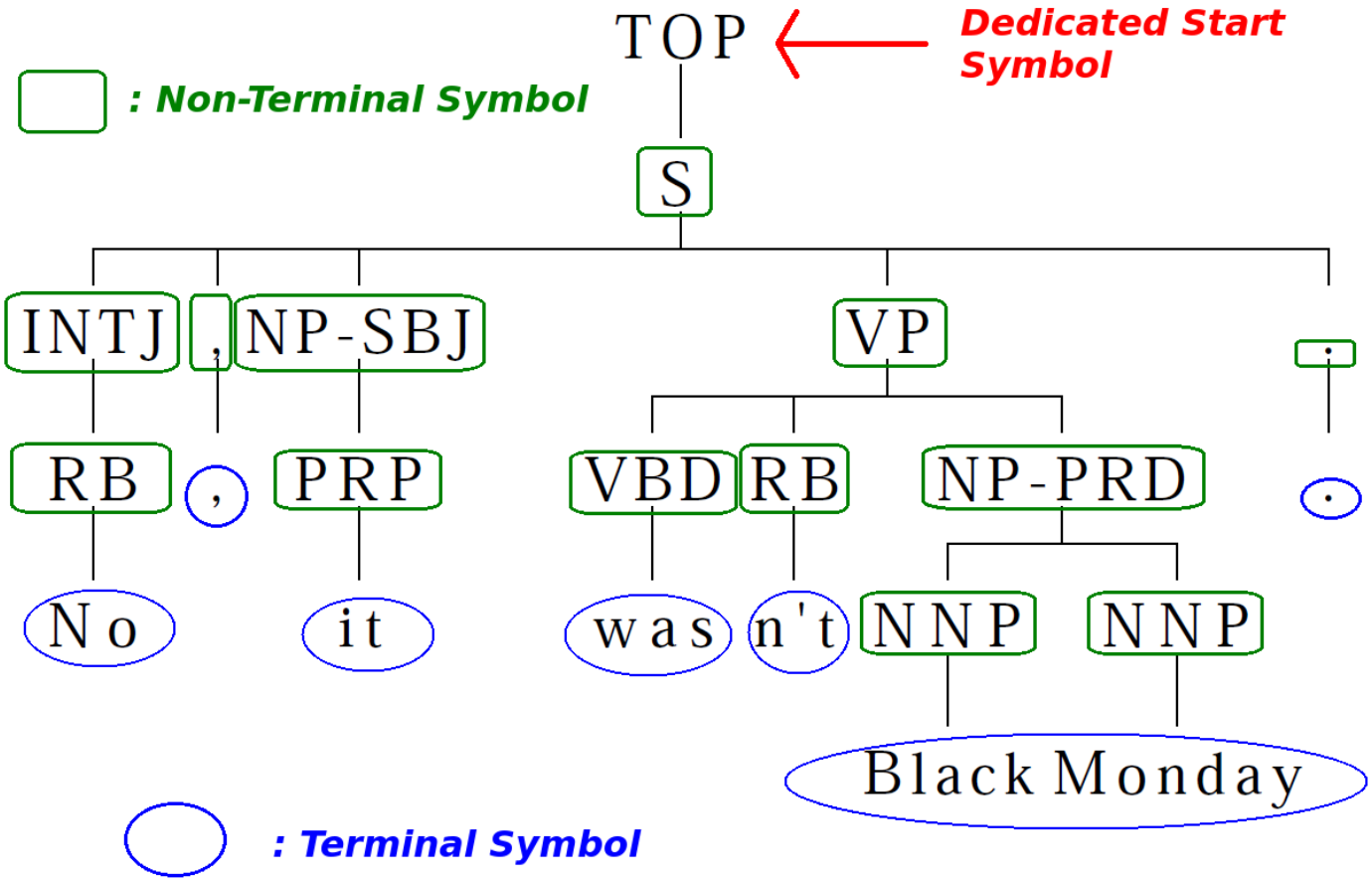


Figure 1: Parse Tree for sentence Wall Street Journal Treebank

Our Goal

- We want to reconstruct the parses of training examples
- We want to assign good parses to new unseen examples
- When dealing with the ambiguity in choosing from a set of possible parses, we want to use probability to choose the most likely one
- We want to build a grammar that captures our training examples but can also generalize to new examples

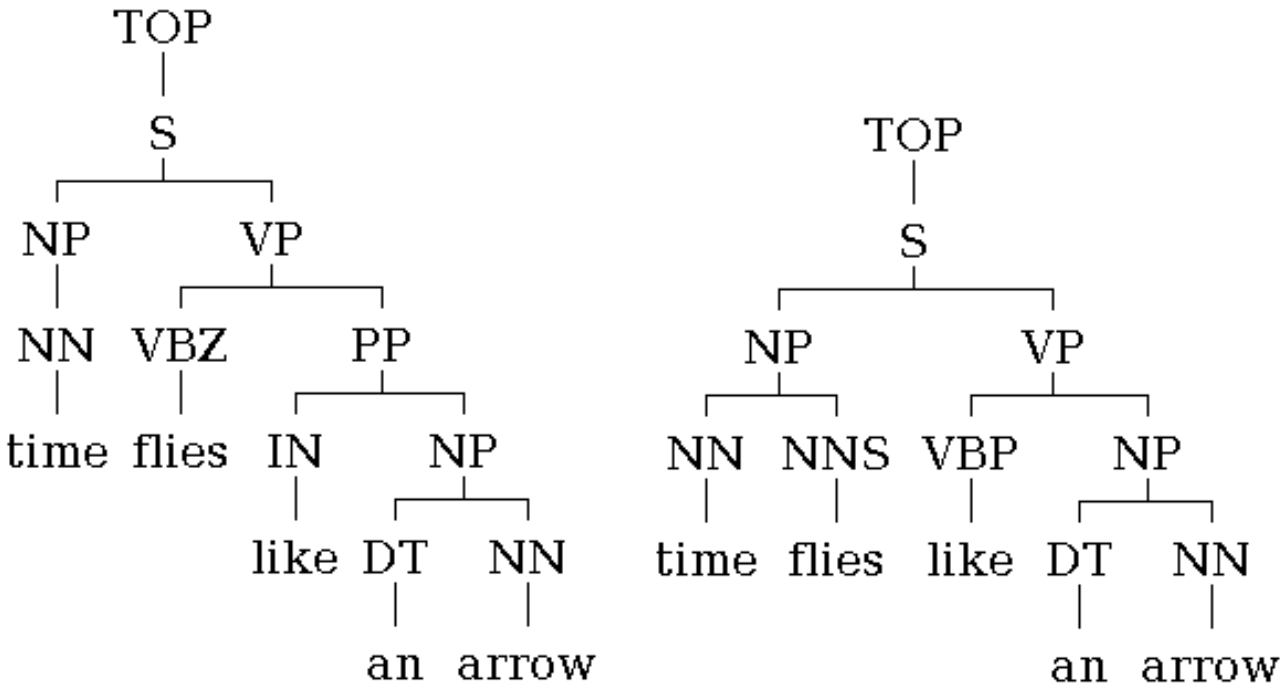
A Context Free Grammar(CFG) example

A very simple CFG that can generate the sentence “The box floats.“

$s - maj \rightarrow s\ fpunc$	$\text{det} \rightarrow \text{the}$
$s \rightarrow np\ vp$	$\text{noun} \rightarrow \text{box}$
$vp \rightarrow verb$	$\text{verb} \rightarrow \text{box}$
$np \rightarrow det\ noun$	$\text{verb} \rightarrow \text{floats}$
	$\text{fpunc} \rightarrow .$

Relation between trees and rules

- There is a strong relations between trees and (Probabilistic) Context Free Grammar rules
- Trees are produced by parsing with rules
- Rules can be extracted from trees by cutting out parts
- Ambiguity expressed by the grammar implies multiple trees and vice versa
- Ambiguity is expressed by (multiple) rules for the same left-hand side, each rule having a probability other than 1

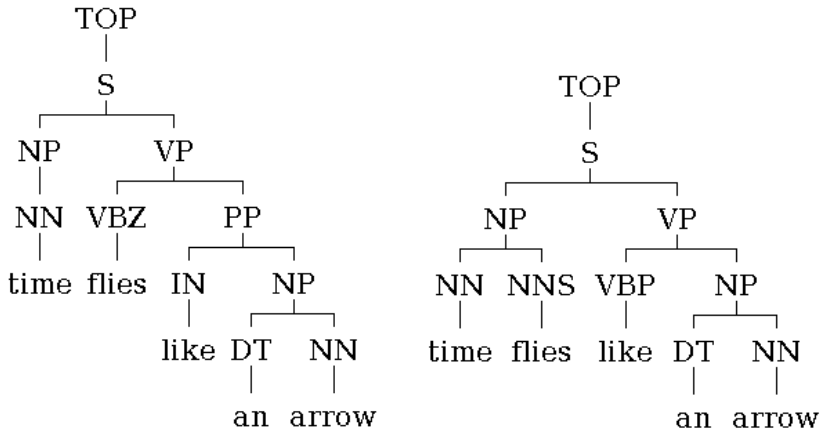


(a) Usual meaning (b) Unusual meaning

Figure 2: Two parses of an ambiguous sentence

- What PCFG rules may we extract from these two trees?

Extracted PCFG



(a) Usual meaning (b) Unusual meaning

TOP → S : 1	PP → IN NP : 1
S → NP VP : 1	NN → time : 2/4
NP → NN : 1/4	NN → arrow : 2/4
NP → NN NNS : 1/4	NNS → flies : 1
NP → DT NN : 2/4	VBP → like : 1
VP → VBZ PP : 1/2	IN → like : 1
VP → VBP NP : 1/2	DT → an : 1
PP → IN NP : 1	VBZ → flies : 1

Grammar extraction: Implementation remarks

Example sentence from treebank:
(TOP (S (NP (NNP Ms.) (NNP Haag)) (S@ (VP (VBZ plays) (NP (NNP Elianti))) (. .))))

- Brackets indicate the beginning "(" and end ")" of subtrees
- Data is binarized/normalized, but still unaries (one child subtrees) at three places:
  - At the top
  - Above the terminals (i.e. "word tags")
  - one level above the word tags
- You can count brackets, use recursive functions or any mechanism you think is best for extracting subtrees.
- You may want to split the problem in 2 steps:
  - Extract "a bag of" subtrees, then
  - produce the rules with their probabilities based on extracted subtrees

Introduction Probabilistic Context Free Grammars: Summary

- In the above we gave informal descriptions of Treebanks and Context Free Grammars
- Our main goal was to give you an intuitive idea of what these things are and help you on your way for the first assignment.
- Context Free Grammars have formal definitions, and these will be covered in the coming lectures.

Formal Definitions Context Free Grammars and Probabilistic Context Free Grammars

- In the next two slides we give the formal definitions for your reference.

Formal Definition CFG

- A Context Free Grammar (CFG) is a four tuple $< W, N, N_1, R >$
- W : Set of terminal symbols (i.e. words)
- N: Set of non-terminal symbols $N_1, ..., N_n$ (i.e. labels)
- $N_1 \in N$: distinguished starting symbol
- R : set of rules Each has form $N_i \rightarrow C_j$, with C_j a string of terminals and non-terminals.

Formal Definition PCFG

Plain Context Free Grammars (CFGs) don't capture how likely different trees are.
$\Rightarrow$ Extend CFGs into probabilistic context free grammars (PCFGs) by adding a probability to every rule:

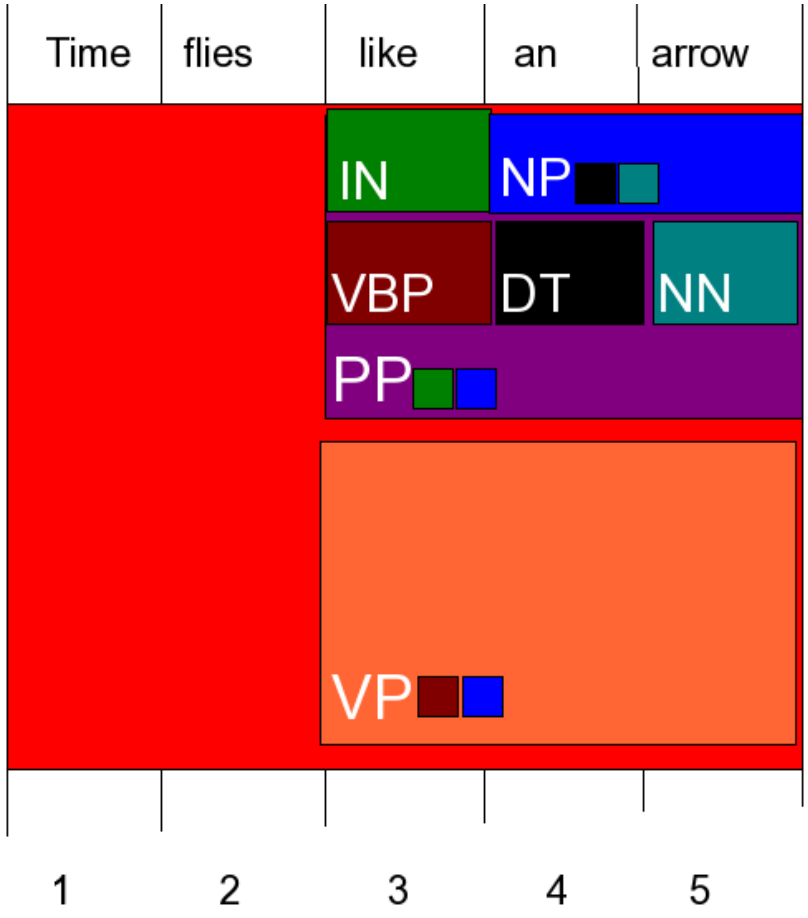
- A Probabilistic Context Free Grammar (PCFG) is a five tuple $< W, N, N_1, R, P >$
- W : Set of terminal symbols (i.e. words)
- N: Set of non-terminal symbols $N_1, ..., N_n$ (i.e. labels)
- $N_1 \in N$: distinguished starting symbol
- R : set of rules Each has form $N_i \rightarrow C_j$, with C_j a string of terminals and non-terminals. Each rule has probability $P(N_i \rightarrow C_j)$
- P : a (probability) function assigning probabilities in the range [0,1] to all rules such that $\forall X \in N \left[\sum_{\beta \in V^*} P(X \rightarrow \beta)\right] = 1$

CKY Algorithm

- A bottom-up chart parsing algorithm
- In simplest form PCFG is required to be in Chomsky Normal form i.e. only binary rules
- One of the most efficient parsing algorithms : $O(N^3)$ complexity
- Uses Dynamic programming: Nodes are iteratively build up for Span Lengths of increasing size

CYK Datastructures

- We want to know what rules are applicable were
- Rules generate a constituent covering a span s , from two constituents that cover subspans partitioning s



PP → IN NP

VP → VBP NP

(c) Chart

TOP → S : 1	PP → IN NP : 1
S → NP VP : 1	NN → time : 2/4
NP → NN : 1/4	NN → arrow : 2/4
NP → NN NNS : 1/4	NNS → flies : 1
NP → DT NN : 2/4	VBP → like : 1
VP → VBZ PP : 1/2	IN → like : 1
VP → VBP NP : 1/2	DT → an : 1
PP → IN NP : 1	VBZ → flies : 1

(d) PCFG

CKY Algorithm - Initialization

Data: An English String $S = a_1 \cdots a_n$, A PCFG G
Result: An initialized Chart *Chart* containing the derivations of the words (SpanLength = 1 derivations)
;
for $i \leftarrow 1$ **to** *SpanLength* **do**
  for *Rule* $r_j \epsilon R$ **do**
    if $(r_j == (N_j \rightarrow a_i))$ **then**
      //Add a node to the Inferred nodes table ;
      Add *Node*(N_j) to *Chart*[i][i].Nodes
      //Add inference of the node from the word a_i *to added node*;
      Chart[i][i].Nodes.get(N_j).Add(*Inferrence*(a_i, null))
    end
  end
end

Algorithm 1: CYK Algorithm initialization

Intermezzo: What is an Inference?

- *Derivation* is used to define one among many different trees that can be found to cover a sentence
- How to distinguish different ways to infer a symbol for a span?
- Programmers want to name important (data) structures!
- I coined the word "*Inference*" to distinguish this
- *Inference* : the part of derivation that denotes a specific way to derive/infer a certain symbol for a certain span

Disclaimer : not to be confused with the concept of (logical) "Inference" in logic!

CKY Algorithm - Main Loop

Data: An English String S, A PCFG G, An initialized Chart *Chart*
Result: A chart containing all possible parses of S under G

for $i \leftarrow 2$ **to** *SpanLength* **do**
  Loop over all the span start positions;
  for $j \leftarrow 1$ **to** n - $i + 1$ **do**
    Loop over all split points of the chart entry;
    for $k \leftarrow 1$ **to** i - *1* **do**
      for *Rule* $r_j = N_j \rightarrow N_\alpha N_\beta \epsilon R$ **do**
        if $Node(N_\alpha, \_)\epsilon(Chart[j, j+k-1) \wedge$
        $Node(N_\beta, \_)\epsilon(Chart[j+k, j+i-1)$ **then**
          if $NOT(N_j\epsilon$ *Chart[i][i].Nodes)* **then**
            Add $Node(N_j)$ to *Chart*[i][i].Nodes
          Chart[i][i].Nodes.get(N_j).Add(*Inferrence*(N_α, N_β,)
        end
      end
    end
  end
end

General Outline

The CYK Algorithm is based on a datastructure called **Chart** containing **Chart Entries** or just **Entries**. There is one entry for every subspan of the sentence of length n. This implies there are $((n+1) \times n)/2$ **Chart Entries** in total. For convenience one can just use an two-dimensional $n \times n$ array in the implementation. Elements of the array then indicate start and end position of the span. Thus an element in a chart at [i,j] contains the **Chart Entry** spanning nodes i - j (inclusive).
Given our Chart, the algorithm builds inferences of **nodes** (corresponding to non-terminals from the PCFG) for the different chart entries in a bottom up manner. It starts with chart entries of length 1, and then goes on with those of length 2 up till the last one of length n.

General Outline - continued

Sometimes a good start for getting simple explanations is Wikipedia. The problem with the Wikipedia algorithm description however, is that it does not tell us how to remember how inferred **Nodes / Non-terminals** came about. This is essential if we want to build trees, without it we can not really go beyond saying : some parse does / does not exist for this sentence under this grammar. Thus we need to keep track of **Inferences** as well as **Nodes**. We have now tree main datastructures, that are combined as follows: A **Chart Entry** contains a dictionary of inferred **Nodes**, every inferred **Node** contains a list of **Inferences**, which enumerates the possible ways to infer the node from one or two different parts that correspond to the right hand side of some rule in the PCFG.

CYK implementation overview

- A CYK parser has as its main datastructure a chart, which is a 2-dimensional table/array with $((n+1) \times n)$ chart entries. The entries correspond to spans from i - j in the sentence (with $j \geq i$)
  Check: Is this the case in your implementation?
- In addition to **chart entries** there are two more important concepts in CYK parsing. The first is **nodes**. Nodes correspond to Non-terminals that could be inferred with rules. The second is **inferences**. Inferences describe how a Node can be inferred.
  Check: Do you have **chart entries**, **nodes** and **Inferences** represented in some way in your implementation?

CYK implementation overview -continued

- During parsing every chart entry is visited once, and then its possible **inferences** are determined from the possible pairs of sub-chart entries that *partition*(i.e. split it into two parts without overlap) its span .
  Is this the case for your implementation?
- For every partition we must check for all combinations of Inferred node $n_\alpha \epsilon InferredNodes(LeftChartEntry)$ and Inferred node $n_\beta \epsilon InferredNodes(LeftChartEntry)$ wether these two nodes can be combined to match a rule in the grammar $n_\gamma \rightarrow n_\alpha n_\beta$. If this is the case, we add the node n_γ with a list of all such inferences to the chart.

CYK implementation overview -continued

- Every **chart entry** thus has to have a Dictionary/HashTable with **nodes** as its keys. Every key points to a list of **Inferences**. Every Inference consist of a pair of pointers to nodes in the left and right sub-entries that were combined to form that inference.
  Check 1: Do you have such a Dictionary/Hashtable in your ChartEntry Datastrucure?
  Check 2: Do you represent **Inferences** in a way that they give pointers to parts from which they are build, i.e. in the form of pointers to **nodes** in sub-entries that partition the entry containing the node having the inferrence?

Complexity

- The structures and procedures described before guarantee that for every chart entry we have to check at most:
  n **splits into different left and right sub-entries** $\times$ n_1 **nonterminals that are inferred in the left sub-entry** $\times$ n_2 **nonterminals that are inferred in the right sub-entry**
  (Since n_1 and n_2 can be seen as factors/constants this has $O(SentenceLength^1)$ complexity)
- The previous is the case for the chart entries with spanlength > 1. For the special case of chart entries that have spanlength $= 1$, we have to just check all the pre-terminal rules that derive the word/terminal presented at the position of that chart entry. (Which is done as an initialization step). (Thus this has $O(SentenceLength^1)$ complexity)

Complexity - continued

- Now we have to do this for all $((n+1) \times n)/2$ chart entries, meaning a "multiplication" by another $O(n^2)$ and therefore **the total parsing complexity is** $O(SentenceLength^3)$ for the CYK algorithm. If you somehow get exponential time performance, something must be wrong in the way you set up your chart or find the inferrences for the different nodes for the different chart entries.