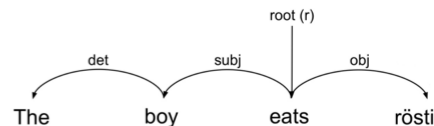


8. Dependency Parsing with MTT

Overview

Construct a tree relating words with syntactic relations

- Relations are directed, i.e., a dependency tree is a directed tree
- Relations are labeled, e.g., noun-modifier, determiner, object, etc.

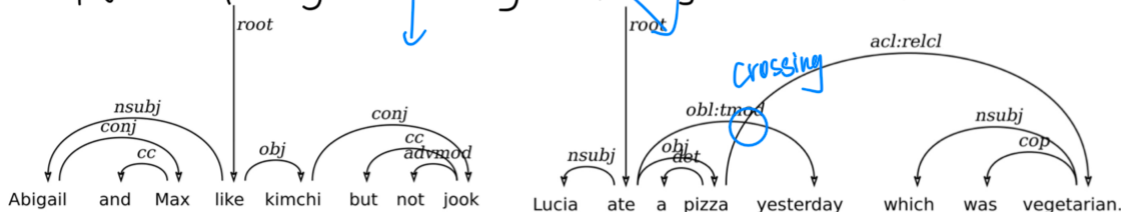


Dependency Trees

A dependency tree (also called dependency graph) for a sentence containing N tokens as a tree with N nodes representing the words (optionally, punctuation) of the original sentence and one special node, the root node.

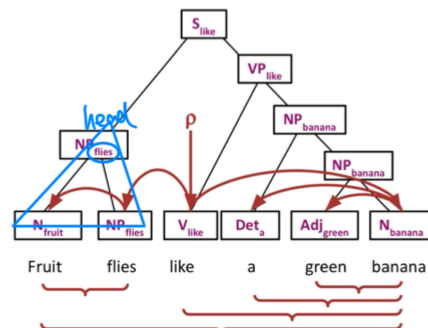
Projective & Non-projective

- Projective dependency trees: no crossing arcs, closely related to constituency.
- Non-projective dependency trees: crossing arcs, closely related to discontinuous constituency.



Relation to Constituency Structure

- Syntactic constituents are always contiguous - any head word chosen will have a path to every node in span.
- All such dependency trees will be projective.
- There is a tight relationship between projective dependency trees and constituency trees.



Dependency relationships produced this way are **always nested within constituents** → we get a **projective dependency tree!**

Probability Distributions over Non-projective trees

$$P(\mathbf{t}|\mathbf{w}) = \frac{1}{Z} \exp \{ \text{score}(\mathbf{t}, \mathbf{w}) \}, \quad Z = \sum_{\mathbf{t} \in T(\mathbf{w})} \exp \{ \text{score}(\mathbf{t}, \mathbf{w}) \}$$

- Compatibility of the parse $\mathbf{t}$ with the sentence $\mathbf{w}$ is specified by scoring function $\text{score}(\mathbf{t}, \mathbf{w})$
- $T(\mathbf{w})$ denotes the set of all admissible parses of sentence $\mathbf{w}$.
- $N = |\mathbf{w}|$, the length of input sentence
- Computing Z requires $O(N^N)$, spanning trees N^{N-2} , root constraint $(N-1)^{N-2}$

Edge-Factored Assumption

- The scoring function, which quantifies how appropriate a parse tree is for the given sentence, decomposes over the edges of the tree.
- An edge-factored scoring function is of the form:

$$\text{score}(\mathbf{t}, \mathbf{w}) = \sum_{i \rightarrow j \in \mathbf{t}} \underbrace{\text{score}(i \rightarrow j, \mathbf{w})}_{\text{edges}} + \underbrace{\text{score}(r, \mathbf{w})}_{\text{root}},$$

where r is the root according to the tree $\mathbf{t}$.

- In terms of probability

$$P(\mathbf{t} | \mathbf{w}) = \frac{1}{Z} \prod_{i \rightarrow j \in \mathbf{t}} \exp \{ \text{score}(i, j, \mathbf{w}) \} \exp \{ \text{score}(r, \mathbf{w}) \}$$

$$Z = \sum_{\mathbf{t} \in \mathcal{T}(\mathbf{w})} \prod_{i \rightarrow j \in \mathbf{t}} \exp \{ \text{score}(i, j, \mathbf{w}) \} \exp \{ \text{score}(r, \mathbf{w}) \}$$

edges root

Kirchhoff's Matrix-Tree Theorem (MTT)

Provides a method of counting number of undirected spanning trees in an undirected graph in $O(N^3)$ time.

$$N_T(G) = |\hat{L}|$$

$L = D - A$ ↗ ↘ remove i -th row and column

$$A_{ij} = \exp \{ \text{score}(i, j, \mathbf{w}) \}$$

$$\rho_j = \exp \{ \text{score}(j, \mathbf{w}) \}$$

Graph Laplacian

$$L_{ij} = \begin{cases} -A_{ij} & , i \neq j \\ \sum_{k \neq i} A_{kj} & , \text{otherwise} \end{cases}$$

Modified Graph Laplacian

$$L_{ij} = \begin{cases} \rho_j & , i=1 \\ -A_{ij} & , i \neq j \\ \sum_{k \neq i} A_{kj} & , \text{otherwise} \end{cases} \quad \text{*Root score added.}$$

Now the probabilities become:

$$P(\mathbf{t} | \mathbf{w}) = \frac{1}{Z} \prod_{i \rightarrow j \in \mathbf{t}} \exp \{ \text{score}(i, j, \mathbf{w}) \} \exp \{ \text{score}(r, \mathbf{w}) \}$$

$$Z = |L| \leftarrow O(N^3)$$

To optimize, we use log-likelihood and use backpropagation to calculate the gradients.

$$L(\theta) = \sum_{i=1}^N \log P(\mathbf{t}^{(i)} | \mathbf{w}^{(i)}) = \sum_{i=1}^N \text{score}(\mathbf{t}^{(i)}, \mathbf{w}^{(i)}) - \log Z(\mathbf{w}^{(i)})$$

Finding the Best Parse of a Sentence

Goal: Find maximum-weight directed spanning tree subject to a root constraint.

Find Maximum-weight Spanning Tree (MST)

$$\text{argmax}_{\mathbf{t} \in \mathcal{T}} \sum_{i \rightarrow j \in \mathbf{t}} \text{score}(i, j, \mathbf{w})$$

Chu-Liu-Edmonds (CLE) Algorithm $O(N^3)$

Algorithm 7.1

def MST(G):

 if CYCLE IN GREEDY(G):

 return EXPAND(CONSTRAIN(MST(CONTRACT(G , CYCLE))))

 else

```

return CONSTRAIN (GREEDY (G))
end if

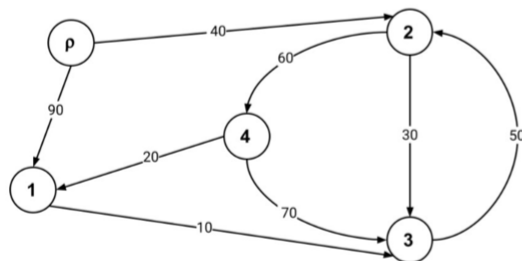
def CONSTRAIN(G):
  if NUMBER OF ROOT EDGES (GREEDY (G)) > 1:
     $e \leftarrow$  ROOT EDGE TO REMOVE (G)
    if CYCLE IN GREEDY (G - e):
      return CONSTRAIN (CONTRACT (G, CYCLE))
    else
      return CONSTRAIN (G - e)
    end if
  else
    return GREEDY (G)
  end if

```

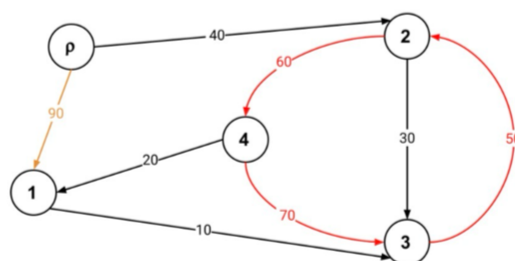


Example:

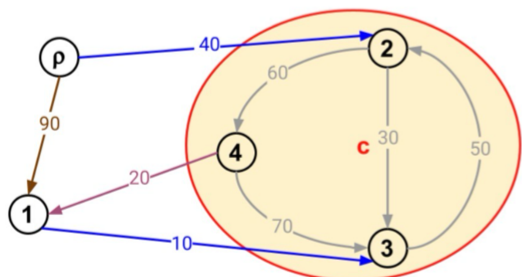
① Constructing the "greedy path"



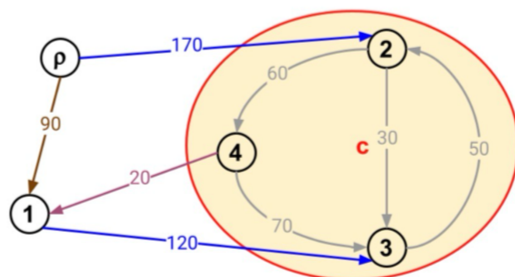
(a) Original Graph.



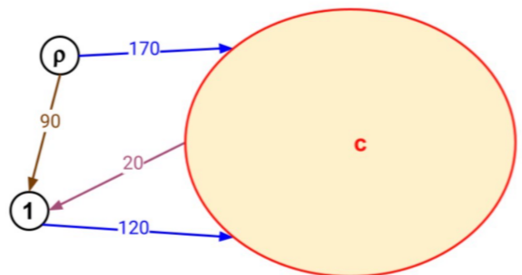
(b) Step 1



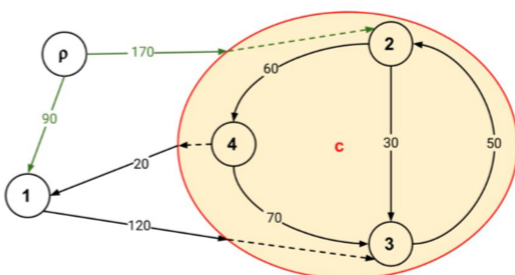
(c) Step 2



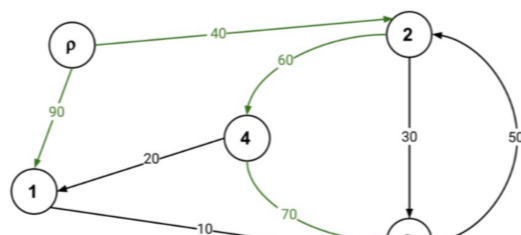
(d) Step 3

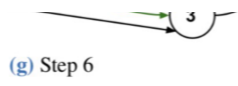


(e) Step 4



(f) Step 5





② Contract

