BITS Pilani
Pilani Campus

Natural Language Processing
DSECL ZG565

Dr. Chetana Gavankar, Ph.D,

IIT Bombay-Monash University Australia

Associate Professor, BITS Pilani

Chetana.gavankar@pilani.bits-pilani.ac.in

innovate achieve lead

BITS Pilani
Pilani Campus

Session 11 - Dependency Parsing
Date – 24th Feb 2024

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin, Prof. Pawan Goyal and many others who made their course materials freely available online.

Session Content

(Ref: Chapter 15 Jurafsky and Martin)

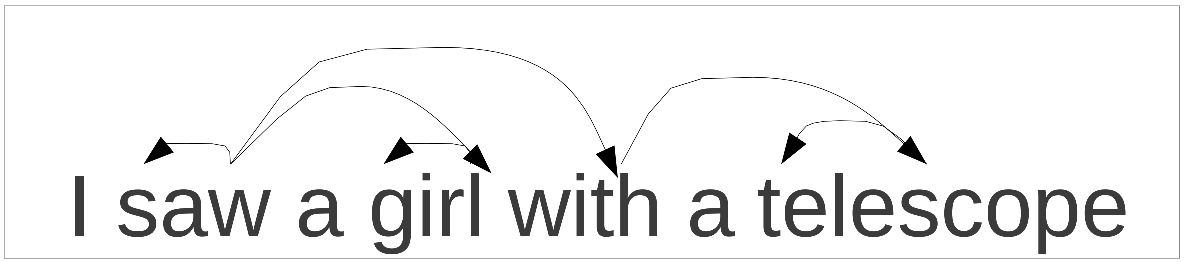
- Motivation
- Dependency structure and Dependency grammar
- Dependency Relation
- Universal Dependencies
- Method of Dependency Parsing
- Graph based dependency Parsing
- Evaluation

Ambiguity is Explosive

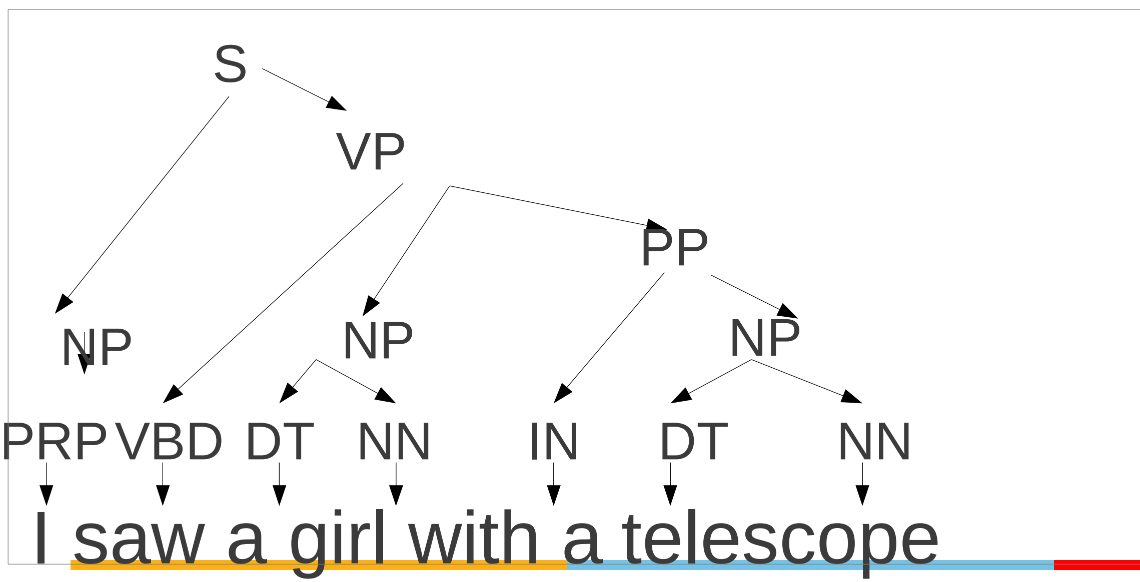
- "I saw the man with the telescope": 2 parses
- "I saw the man on the hill with the telescope.": 5 parses
- "I saw the man on the hill in Texas with the telescope": 14 parses
- "I saw the man on the hill in Texas with the telescope at noon.": 42 parses
- "I saw the man on the hill in Texas with the telescope at noon on Monday" 132 parses

Two Types of Parsing

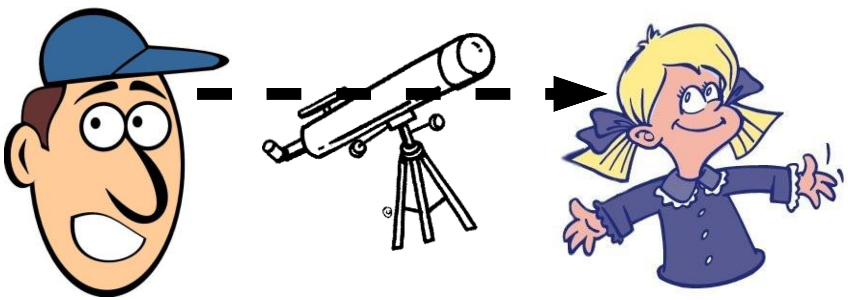
- **Dependency:** focuses on relations between words

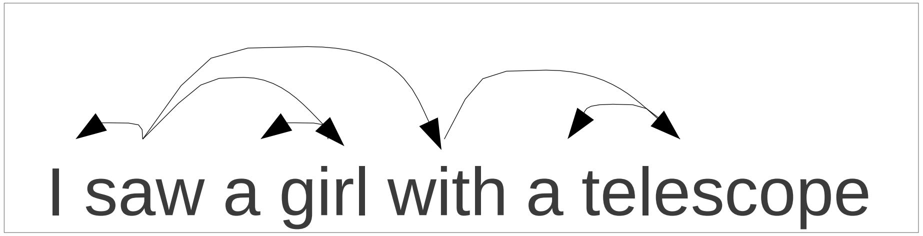


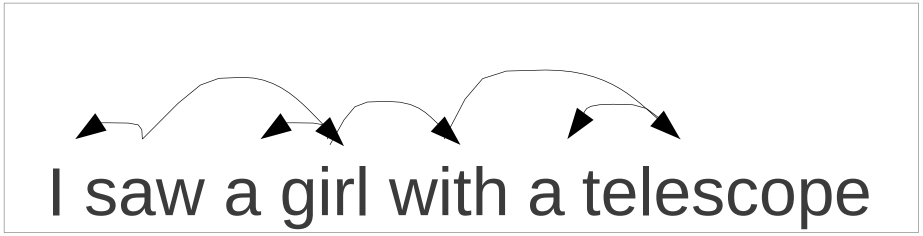
- **Phrase structure:** focuses on identifying phrases and their recursive structure



Dependencies Also Resolve Ambiguity



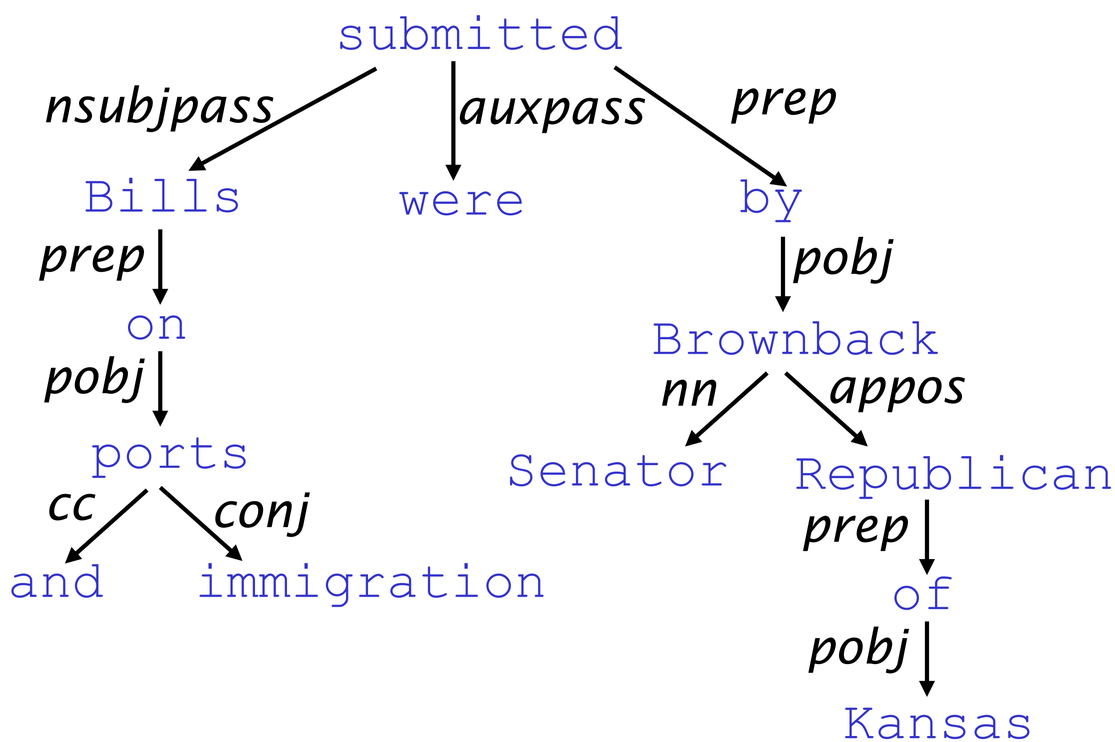




Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)

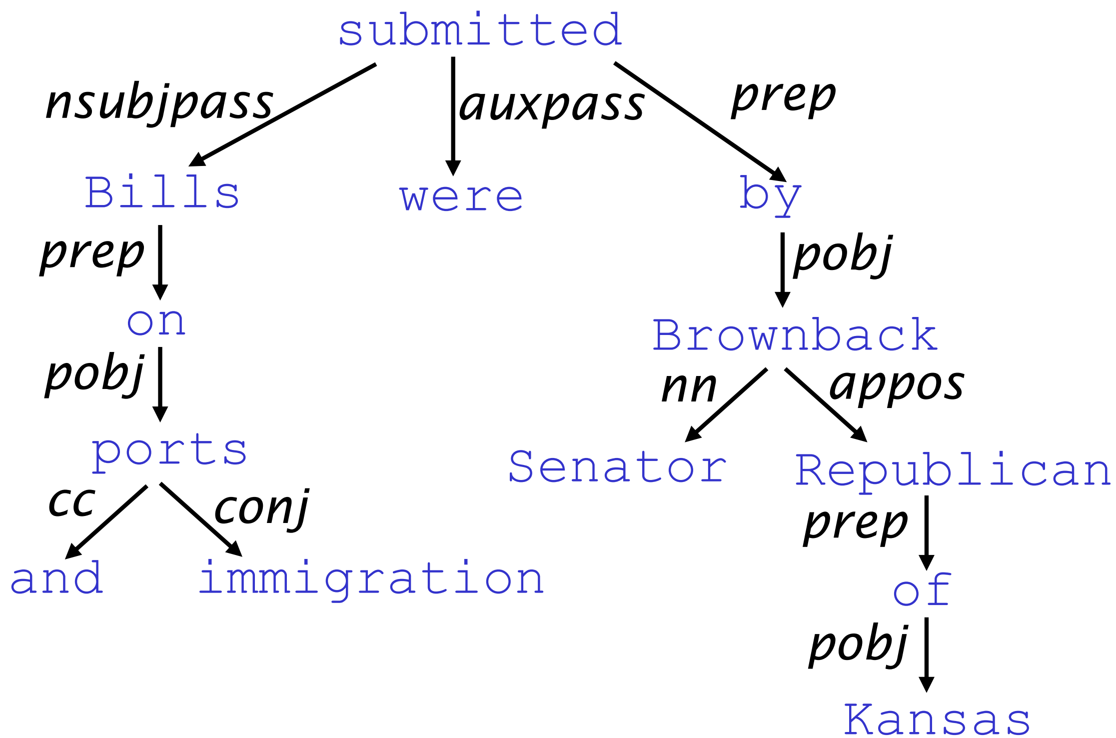


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrow connects a head (governor) with a dependent (modifier)

Usually, dependencies form a tree (connected, acyclic, single-head)



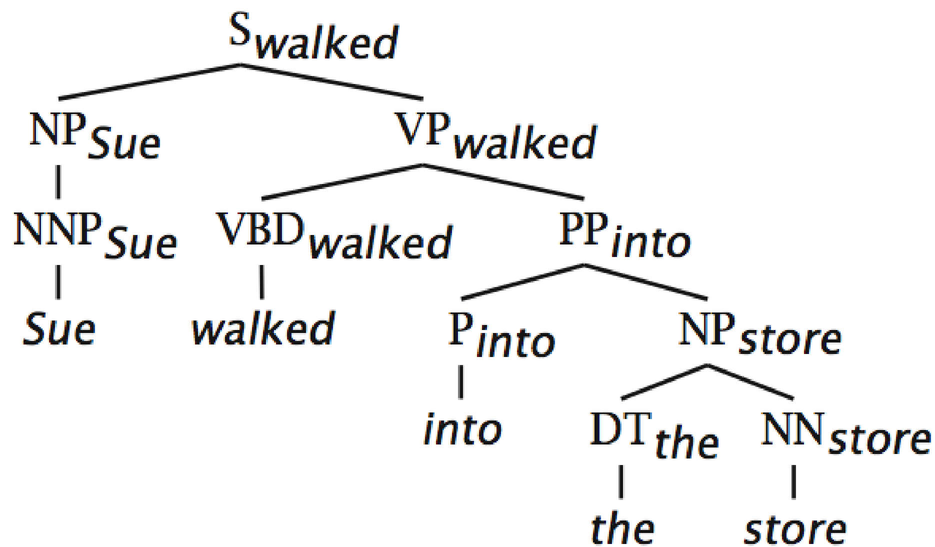
Relation between phrase structure and dependency structure

A dependency grammar has a notion of a head. Officially, CFGs don't.
But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":

- The head of a Noun Phrase is a noun/number/adj/…
- The head of a Verb Phrase is a verb/modal/….

The head rules can be used to extract a dependency parse from a CFG parse

- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., "flat")

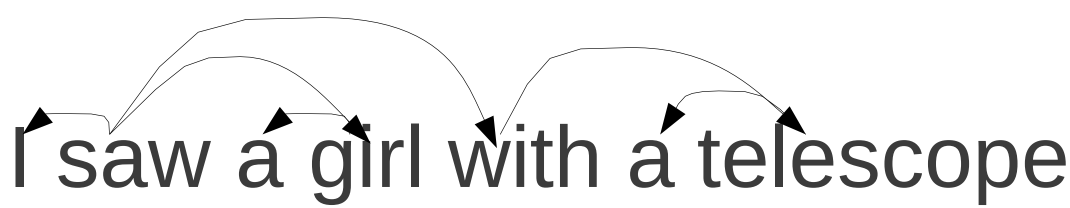


Dependency graph

- A dependency structure can be defined as a directed graph G, consisting of
  - a set V of nodes,
  - a set A of arcs (edges),
- Labeled graphs:
  - Nodes in V are labeled with word forms (and annotation).
  - Arcs in A are labeled with dependency types.
- Notational convention:
  - Arc (w_i, d, w_j) links head w_i to dependent w_j with label d
  - $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
  - $i \rightarrow j \equiv (i, j) \in A$
  - $i \rightarrow^* j \equiv i = j \vee \exists k : i \rightarrow k, k \rightarrow^* j$

Formal conditions on dependency graph

- G is connected:
  - For every node i there is a node j such that $i \rightarrow j$ or $j \rightarrow i$.
- G is acyclic:
  - if $i \rightarrow j$ then not $j \rightarrow^* i$.
- G obeys the single head constraint:
  - if $i \rightarrow j$ then not $k \rightarrow j$, for any $k \neq i$.
- G is projective:
  - if $i \rightarrow j$ then $j \rightarrow^* k$, for any k such that both j and k lie on the same side of i.



Universal dependencies

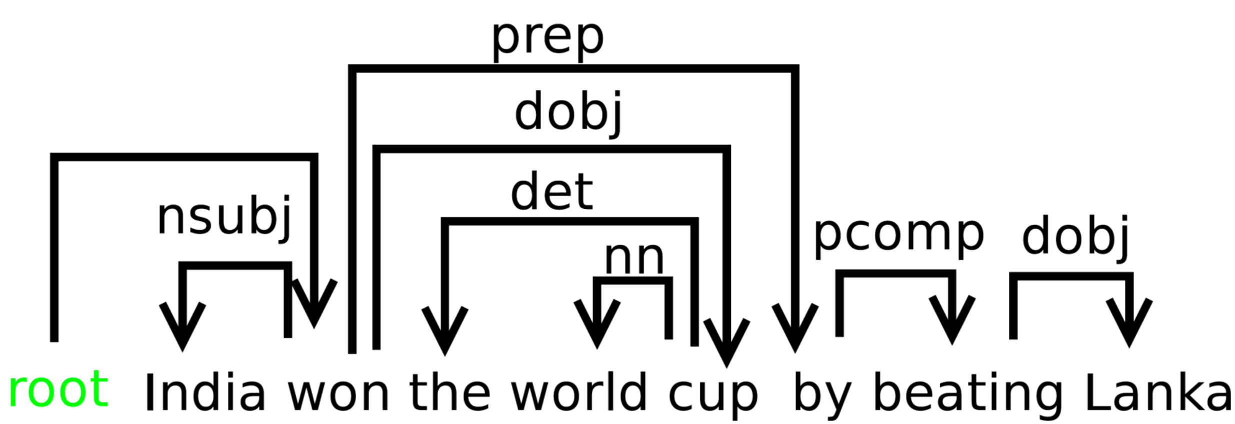
http://universaldependencies.org/

- Annotated treebanks in many languages
- Uniform annotation scheme across all languages:
    - Universal POS tags
    - Universal dependency relations

Dependency Relations

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Example Dependency Parse



Methods of Dependency Parsing

1. **Dynamic programming (like in the CKY algorithm)**

   You can do it similarly to lexicalized PCFG parsing: an $O(n^5)$ algorithm

   Eisner (1996) gives a clever algorithm that reduces the complexity to $O(n^3)$, by producing parse items with heads at the ends rather than in the middle

2. **Graph algorithms**

   You create a Maximum Spanning Tree for a sentence

3. **Constraint Satisfaction**

   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. **Deterministic parsing**

   Greedy choice of attachments guided by machine learning classifiers

   MaltParser (Nivre et al. 2008) – discussed in the next segment

Deterministic parsing

Basic idea

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

Configurations

A parser configuration is a triple $c = (S, B, A)$, where

- S : a stack $[\ldots, w_i]_S$ of partially processed words,
- B : a buffer $[w_j, \ldots]_B$ of remaining input words,
- A : a set of labeled arcs (w_i, d, w_j).

Stack	Buffer	Arcs
[sent, her, a]$_S$	[letter, .]$_B$	He $\xleftarrow{\text{SBJ}}$ sent

Transition based systems for Dependency parsing

A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, *where*

- C is a set of configurations,
- T is a set of transitions, such that $t : C \to C$,
- c_s is an initialization function
- $C_t \subseteq C$ is a set of terminal configurations.

A transition sequence for a sentence x is a set of configurations
$C_{0,m} = (c_o, c_1, \ldots, c_m)$ such that
$c_o = c_s(x)$, $c_m \in C_t$, $c_i = t(c_{i-1})$ for some $t \in T$

Initialization: $([]_S, [w_1, \ldots, w_n]_B, \{\})$
Termination: $(S, []_B, A)$

Arc eager parsing(Malt parser)

Left-Arc(d) $\dfrac{([\ldots, w_i]_S,\ [w_j, \ldots]_B,\ A)}{([\ldots]_S,\ [w_j, \ldots]_B,\ A \cup \{(w_j, d, w_i)\})}$ $\neg\mathrm{HEAD}(w_i)$

Right-Arc(d) $\dfrac{([\ldots, w_i]_S,\ [w_j, \ldots]_B,\ A)}{([\ldots, w_i, w_j]_S,\ [\ldots]_B,\ A \cup \{(w_i, d, w_j)\})}$

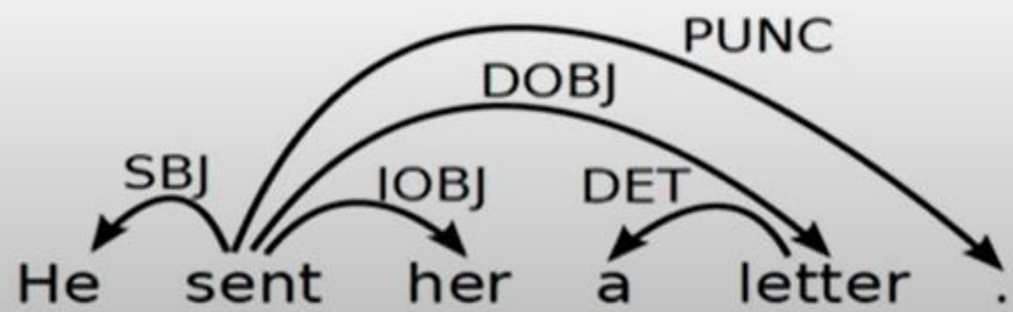
Reduce $\dfrac{([\ldots, w_i]_S,\ B,\ A)}{([\ldots]_S,\ B,\ A)}$ $\mathrm{HEAD}(w_i)$

Shift $\dfrac{([\ldots]_S,\ [w_i, \ldots]_B,\ A)}{([\ldots, w_i]_S,\ [\ldots]_B,\ A)}$

Arc Eager Parsing example

Transitions:

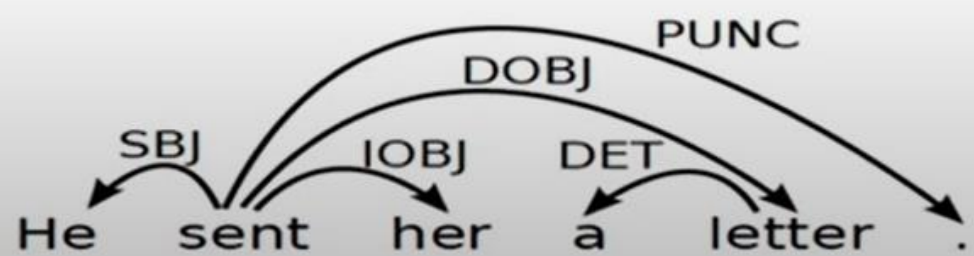
Stack	Buffer	Arcs
$[\ ]_S$	$[\text{He, sent, her, a, letter, .}]_B$	



Arc Eager Parsing example

Transitions: SH-LA

Stack	Buffer	Arcs
$[\ ]_S$	$[\text{sent, her, a, letter, .}]_B$	$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$



Arc Eager Parsing example

Transitions: SH-LA-SH

Stack	**Buffer**	**Arcs**
$[\text{sent}]_S$	$[\text{her, a, letter, .}]_B$	$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

Arc Eager Parsing example

Transitions: SH-LA-SH-RA

Stack	**Buffer**	**Arcs**
$[\text{sent, her}]_S$	$[\text{a, letter, .}]_B$	$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$ $\text{sent} \xrightarrow{\text{IOBJ}} \text{her}$

Arc Eager Parsing example

Transitions: SH-LA-SH-RA-SH

Stack

$[\text{sent, her, a}]_S$

Buffer

$[\text{letter, .}]_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

sent $\xrightarrow{\text{IOBJ}}$ her

Arc Eager Parsing example

Transitions: SH-LA-SH-RA-SH-LA

Stack

$[\text{sent, her}]_S$

Buffer

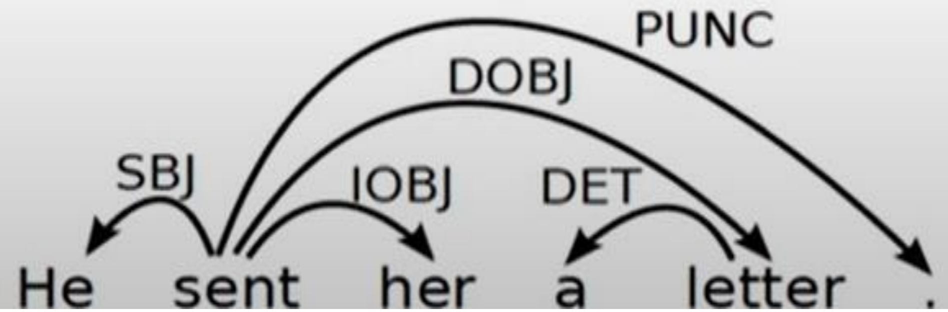
$[\text{letter, .}]_B$

Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

$\text{sent} \xrightarrow{\text{IOBJ}} \text{her}$

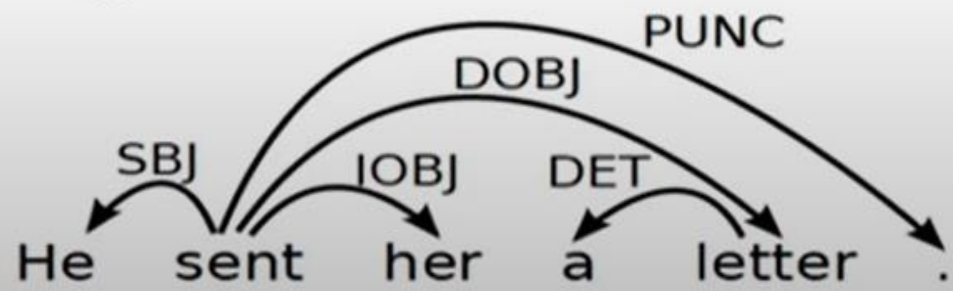
$\text{a} \xleftarrow{\text{DET}} \text{letter}$



Arc Eager Parsing example

Transitions: SH-LA-SH-RA-SH-LA-RE

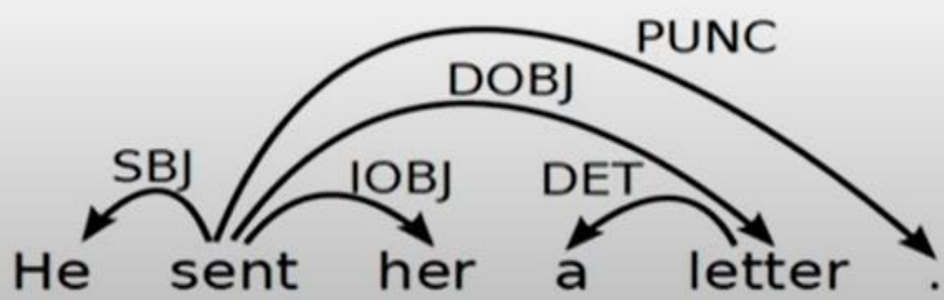
Stack	**Buffer**	**Arcs**
$[\text{sent}]_S$	$[\text{letter}, .]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter



Arc Eager Parsing example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA

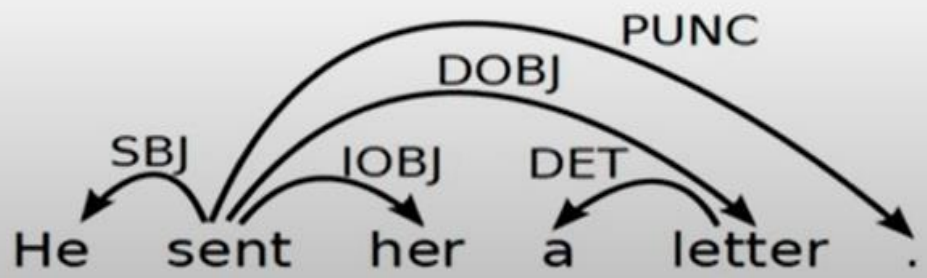
Stack	Buffer	Arcs
$[\text{sent, letter}]_S$	$[.]_B$	$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$
		$\text{sent} \xrightarrow{\text{IOBJ}} \text{her}$
		$\text{a} \xleftarrow{\text{DET}} \text{letter}$
		$\text{sent} \xrightarrow{\text{DOBJ}} \text{letter}$



Arc Eager Parsing example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE

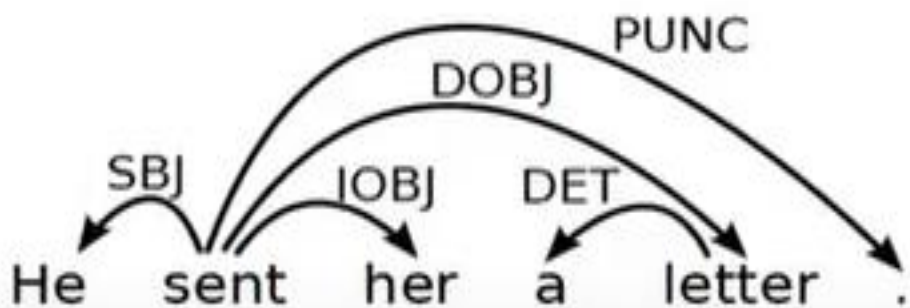
Stack	**Buffer**	**Arcs**
$[\text{sent}]_S$	$[.]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter
		sent $\xrightarrow{\text{DOBJ}}$ letter



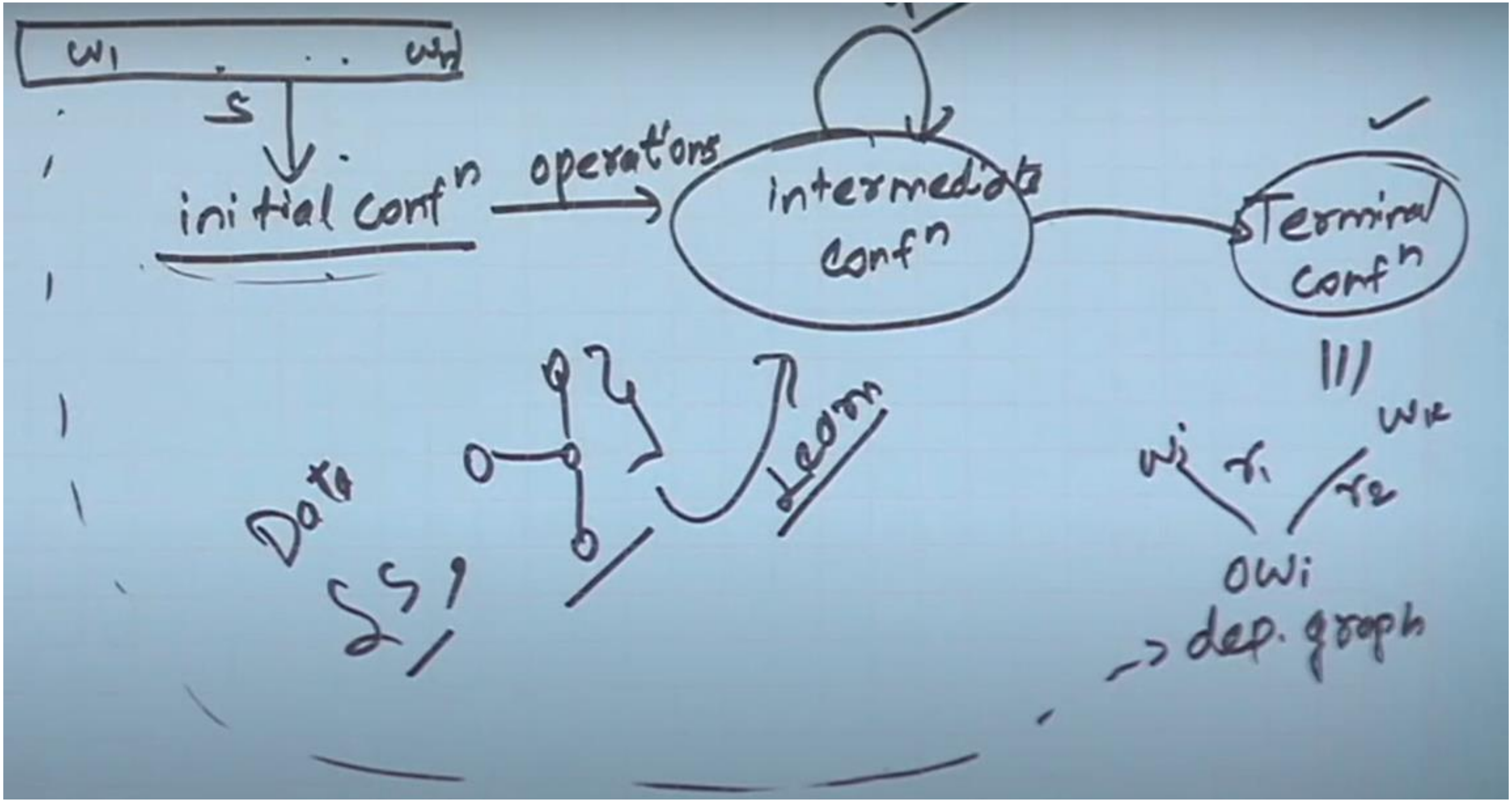
Arc Eager Parsing example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

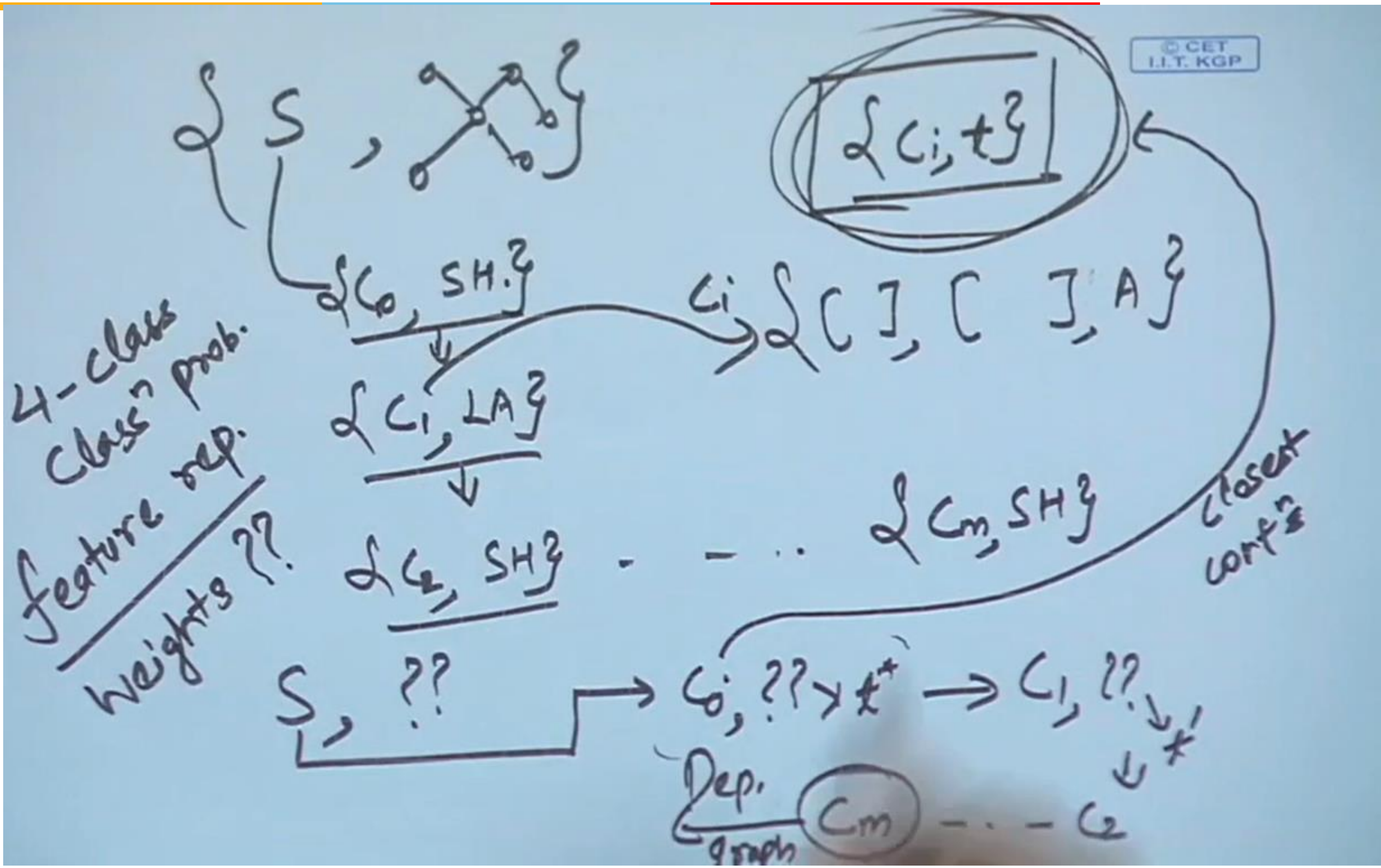
Stack	Buffer	Arcs
$[\text{sent}, .]_S$	$[\ ]_B$	He $\xleftarrow{\text{SBJ}}$ sent sent $\xrightarrow{\text{IOBJ}}$ her a $\xleftarrow{\text{DET}}$ letter sent $\xrightarrow{\text{DOBJ}}$ letter sent $\xrightarrow{\text{PUNC}}$



Learning weights



Deriving dependency graph for a new sentence



Classifier for learning the transmission

Data-driven deterministic parsing:

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

Learning Problem

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

Three issues

- How to represent configurations by feature vectors?
- How to derive training data from treebanks?
- How to learn classifiers?

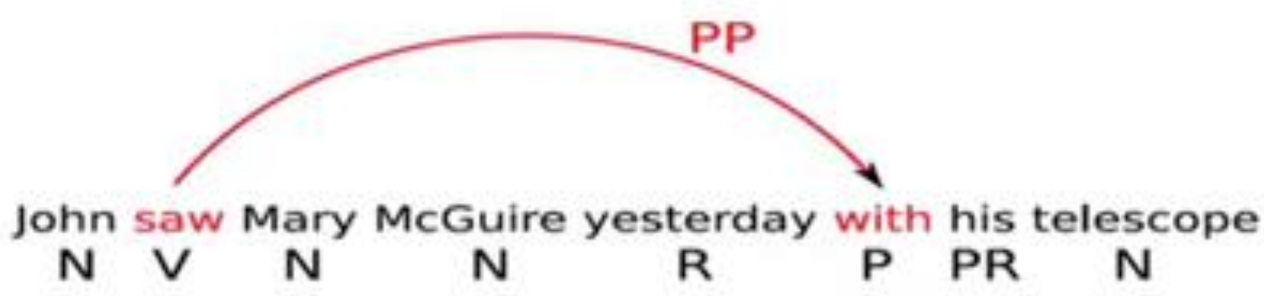
Feature Models

A feature representation $f(c)$ of a configuration c is a vector of simple features $f_i(c)$.

Typical Features

- Nodes:
  - Target nodes (top of S, head of B)
  - Linear context (neighbors in S and B)
  - Structural context (parents, children, siblings in G)
- Attributes:
  - Word form (and lemma)
  - Part-of-speech (and morpho-syntactic features)
  - Dependency type (if labeled)
  - Distance (between target tokens)

Feature Models



Features

Part-of-speech of words surrounding and between w_i and w_j

inbetween-pos = Noun
inbetween-pos = Adverb
dependent-pos-right = Pronoun
head-pos-left=Noun

To guide the parser, a linear classifier can be used:

$$t^* = \arg\max_t w.f(c,t)$$

Weight vector w learned from treebank data.

Using classifier at run-time

PARSE($w_1, \ldots, w_n$)
$c \leftarrow ([]_S, [w_1, \ldots, w_n]_B, \{\})$
while $B_c \neq []$
  $t^* \leftarrow \arg\max_t w.f(c,t)$
  $c \leftarrow t^*(c)$
return $T = (\{w_1, \ldots, w_n\}, A_c)$

Training Data

- Training instances have the form $(f(c), t)$, where
  - $f(c)$ is a feature representation of a configuration c,
  - t is the correct transition out of c (i.e., $o(c) = t$).
- Given a dependency treebank, we can sample the oracle function o as follows:
  - For each sentence x with gold standard dependency graph G_x, construct a transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ such that

    $$c_0 = c_s(x),$$
    $$G_{c_m} = G_x$$
  - For each configuration $c_i (i < m)$, we construct a training instance $(f(c_i), t_i)$, where $t_i(c_i) = c_{i+1}$.

Standard Oracle for Arc Eager parsing

$o(c,T) =$

- **Left-Arc** if $\text{top}(S_c) \leftarrow \text{first}(B_c)$ in T
- **Right-Arc** if $\text{top}(S_c) \rightarrow \text{first}(B_c)$ in T
- **Reduce** if $\exists w < top(S_c) : w \leftrightarrow \text{first}(B_c)$ in T
- **Shift** otherwise

Online learning with an oracle

LEARN($\{T_1, \ldots, T_N\}$)
    $w \leftarrow 0.0$
    for i in $1..K$
      for j in $1..N$
        $c \leftarrow ([]_S, [w_1, \ldots, w_{n_j}]_B, \{\})$
        while $B_c \neq []$
          $t^* \leftarrow \arg\max_t w.f(c,t)$
          $t_o \leftarrow o(c, T_i)$
          if $t^* \neq t_o$
            $w \leftarrow w + f(c, t_o) - f(c, t^*)$
          $c \leftarrow t_o(c)$
    return w

Oracle $o(c, T_i)$ returns the optimal transition of c and T_i

Example

Consider the sentence, 'John saw Mary'.

- Draw a dependency graph for this sentence.
- Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are 'Nouns', while the POS tag of *saw* is 'Verb'. Assume that your features correspond to the following conditions:
  - The stack is empty
  - Top of stack is Noun and Top of buffer is Verb
  - Top of stack is Verb and Top of buffer is Noun

  Initialize the weights of all your features to *5.0*, except that in all of the above cases, you give a weight of *5.5* to *Left-Arc*. Define your feature vector and the initial weight vector.
- Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

Example

F(c,t)=[(c0,LA),(c1,LA),(c2,LA)|(c0,RA),(c1,RA),(c2,RA) |(c0,RE),(c1,RE),(c2,RE)|(c0,SH),(c1,SH),(c2,SH)]

W =[5.5,5.5,5.5|5.0,5,0,5.0|5.0,5.0,5.0|......]

So for the given conditions

F(c,LA) = [1,0,0|0,0,0|......]

F(c,RA) =[0,0,0|1,0,0|......]

t*=argmax(w*f(c,t))

t*=LA

as per oracle /optimal transition t^0 =SH

So we need to update the weights

To update the weights

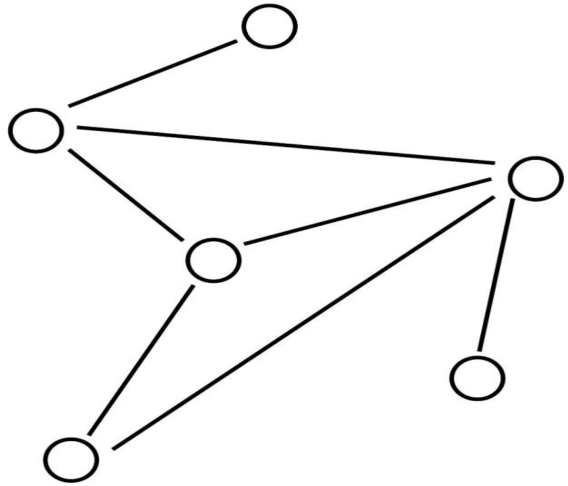
W=W+f(c,t^0)-f(c,t^*)

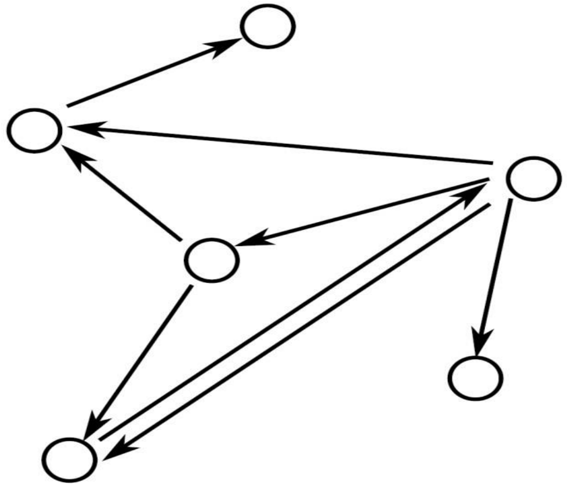
W=[5.5,5.0 5.5,5.5,),5.0..........]+[0,0,0|0,0,0|.........1,0,0]-[1,0,0|0,0,0|.....]

New vector=[4.5,5.5,5.5|5.0...... ....6.0,5.0,5.0]

Graph

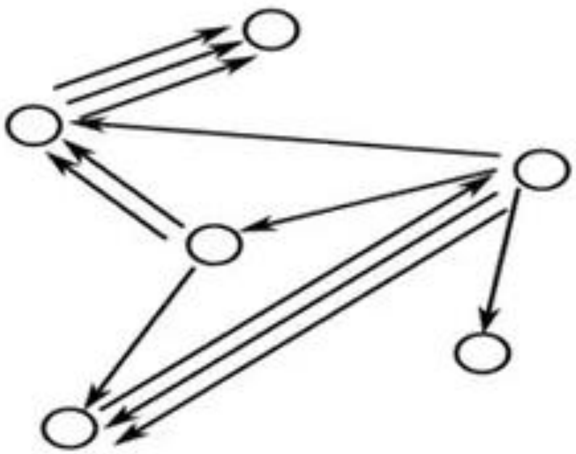
- A graph $G = (V, A)$ is a set of verteces V and arcs $(i, j) \in A$, where $i, j \in V$
- Undirected graphs: $(i, j) \in A \Leftrightarrow (j, i) \in A$
- **Directed graphs (digraphs)**: $(i, j) \in A \not\Rightarrow (j, i) \in A$





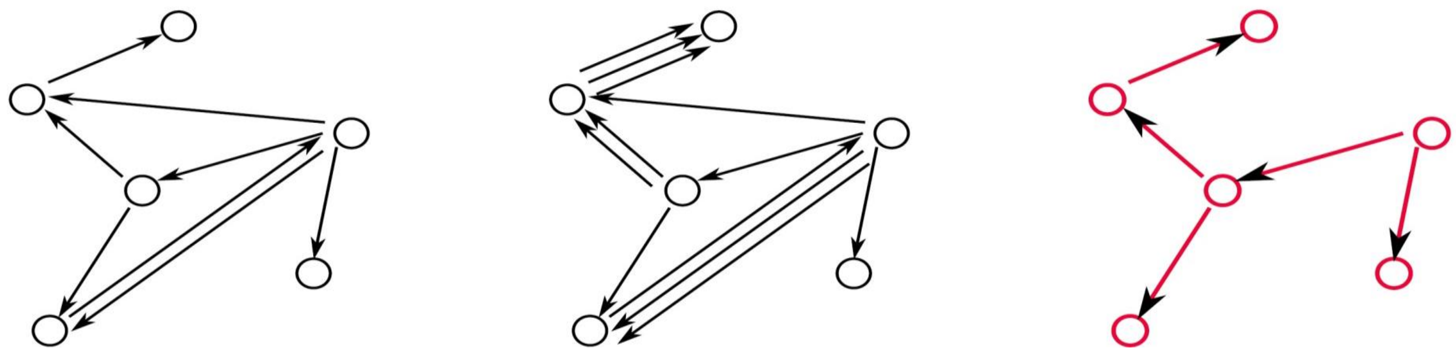
Multi Digraph

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i, j, k) \in A$ represents the k^{th} arc from vertex i to vertex j.



Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - G' is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs



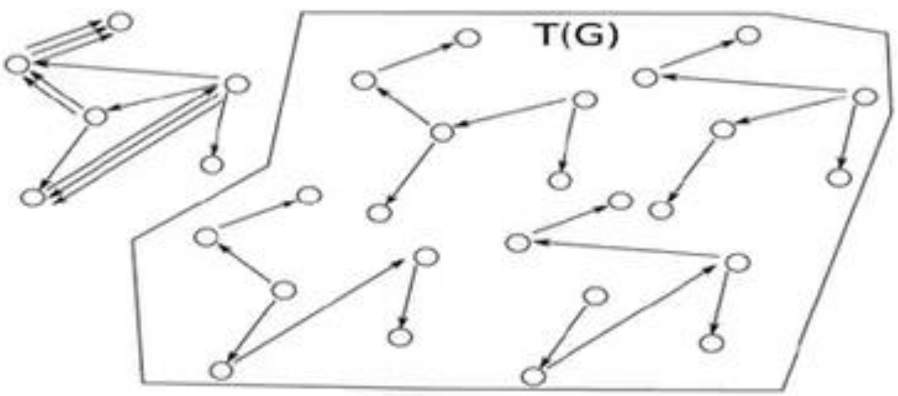
Weighted Spanning tree

- Assume we have a weight function for each arc in a multi-digraph $G = (V, A)$.
- Define $w_{ij}^k \geq 0$ to be the weight of $(i, j, k) \in A$ for a multi-digraph
- Define the weight of directed spanning tree G' of graph G as

$$w(G') = \sum_{(i,j,k) \in G'} w_{ij}^k$$

MST

Let $T(G)$ be the set of all spanning trees for graph G



The MST problem

Find the spanning tree G' of the graph G that has the highest weight

$$G' = \underset{G' \in T(G)}{\arg\max}\, w(G') = \underset{G' \in T(G)}{\arg\max} \sum_{(i,j,k) \in G'} w_{ij}{}^{k}$$

Finding MST

Directed Graph

For each sentence x, define the directed graph $G_x = (V_x, E_x)$ given by

$$V_x = \{x_0 = root, x_1, \ldots, x_n\}$$

$$E_x = \{(i,j) : i \neq j, (i,j) \in [0:n] \times [1:n]\}$$

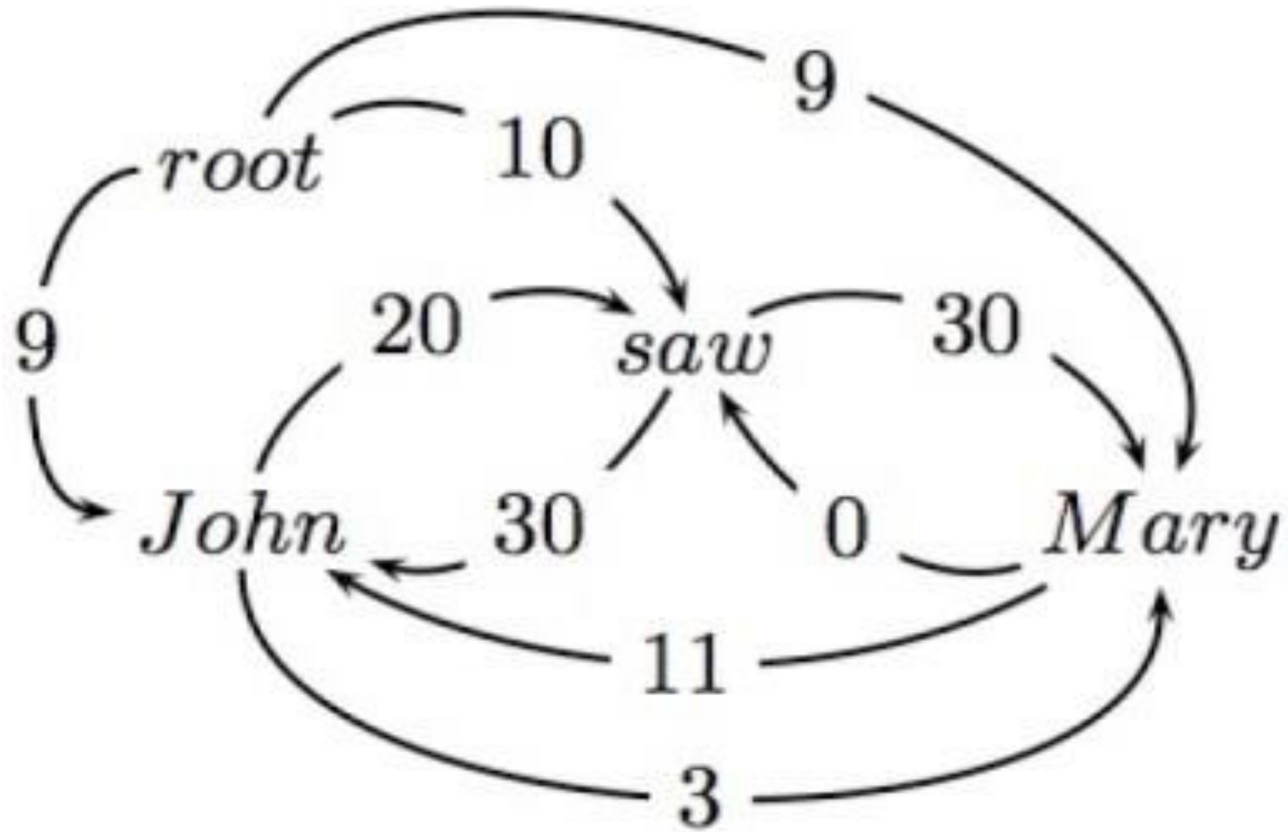
G_x is a graph with

- the sentence words and the dummy root symbol as vertices and
- a directed edge between every pair of distinct words and
- a directed edge from the root symbol to every word

Chu-Liu-Edmonds algorithm

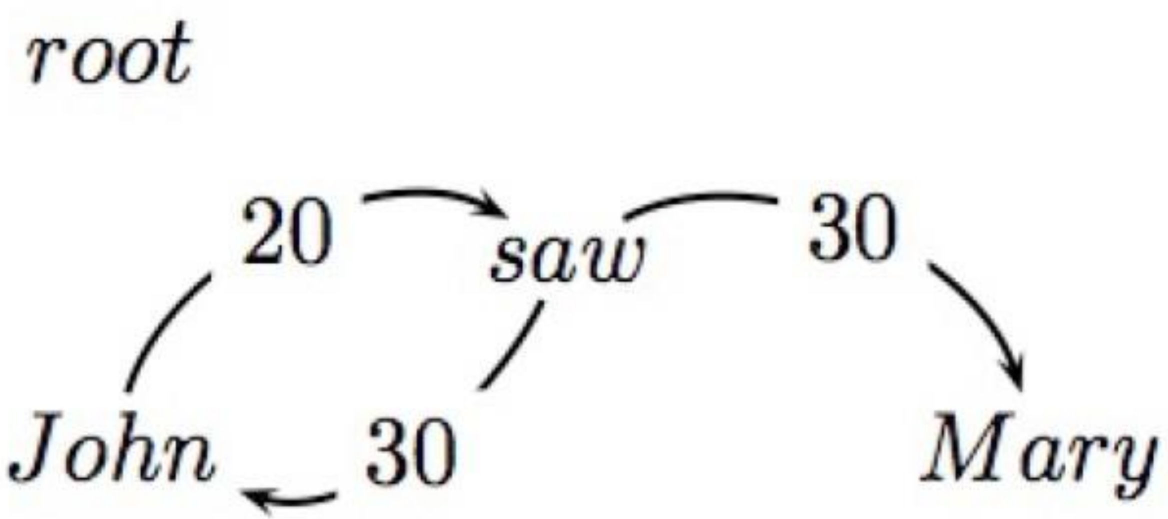
- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - Identify the cycle and contract it into a single vertex.
  - Recalculate edge weights going into and out of the cycle.

Chu-Liu-Edmonds Example



Chu-Liu-Edmonds Example

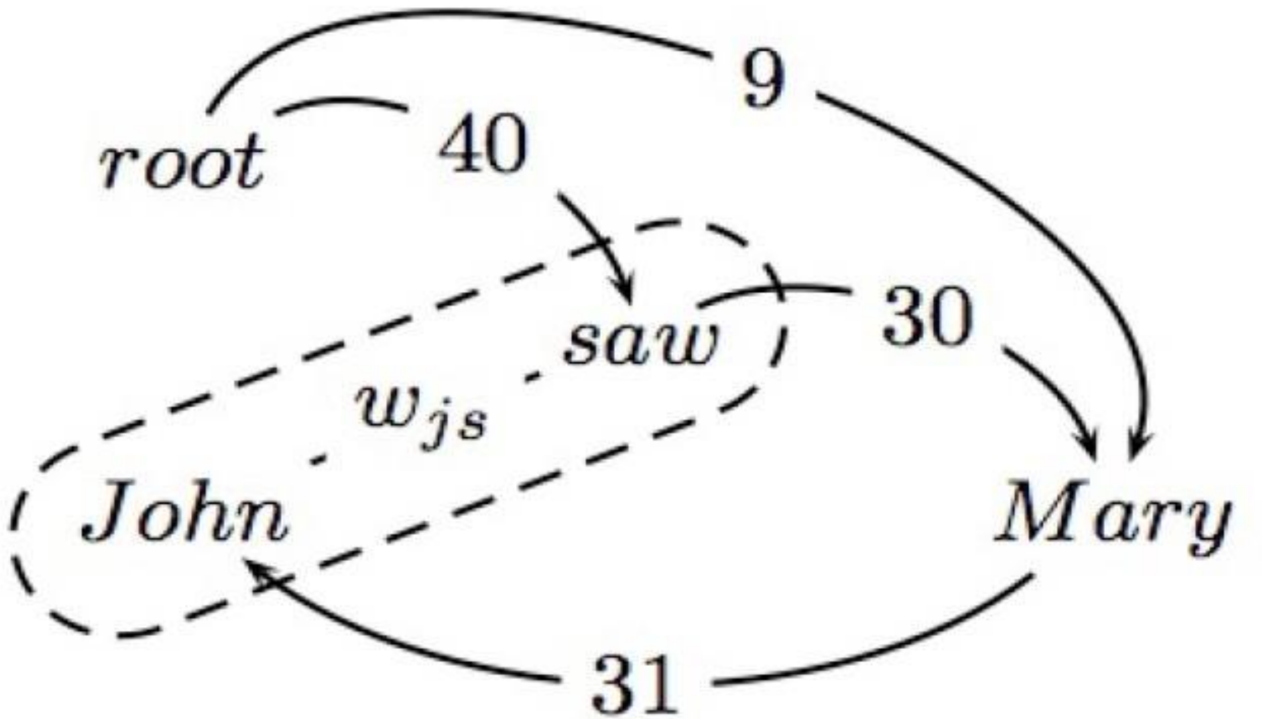
- Find highest scoring incoming arc for each vertex



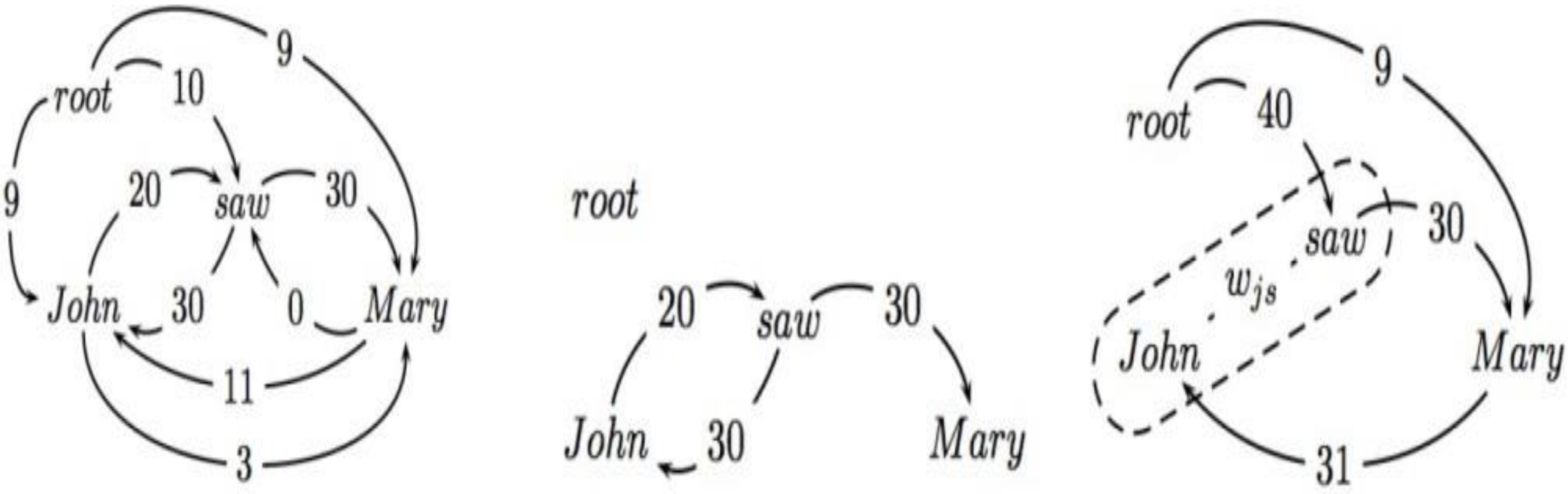
- If this is a tree, then we have found MST!!

Chu-Liu-Edmonds Example

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle

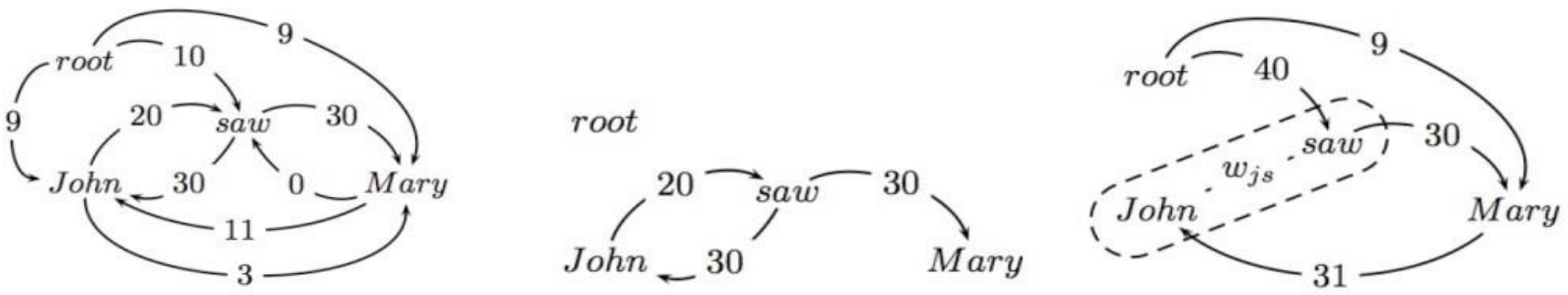


Chu-Liu-Edmonds Example



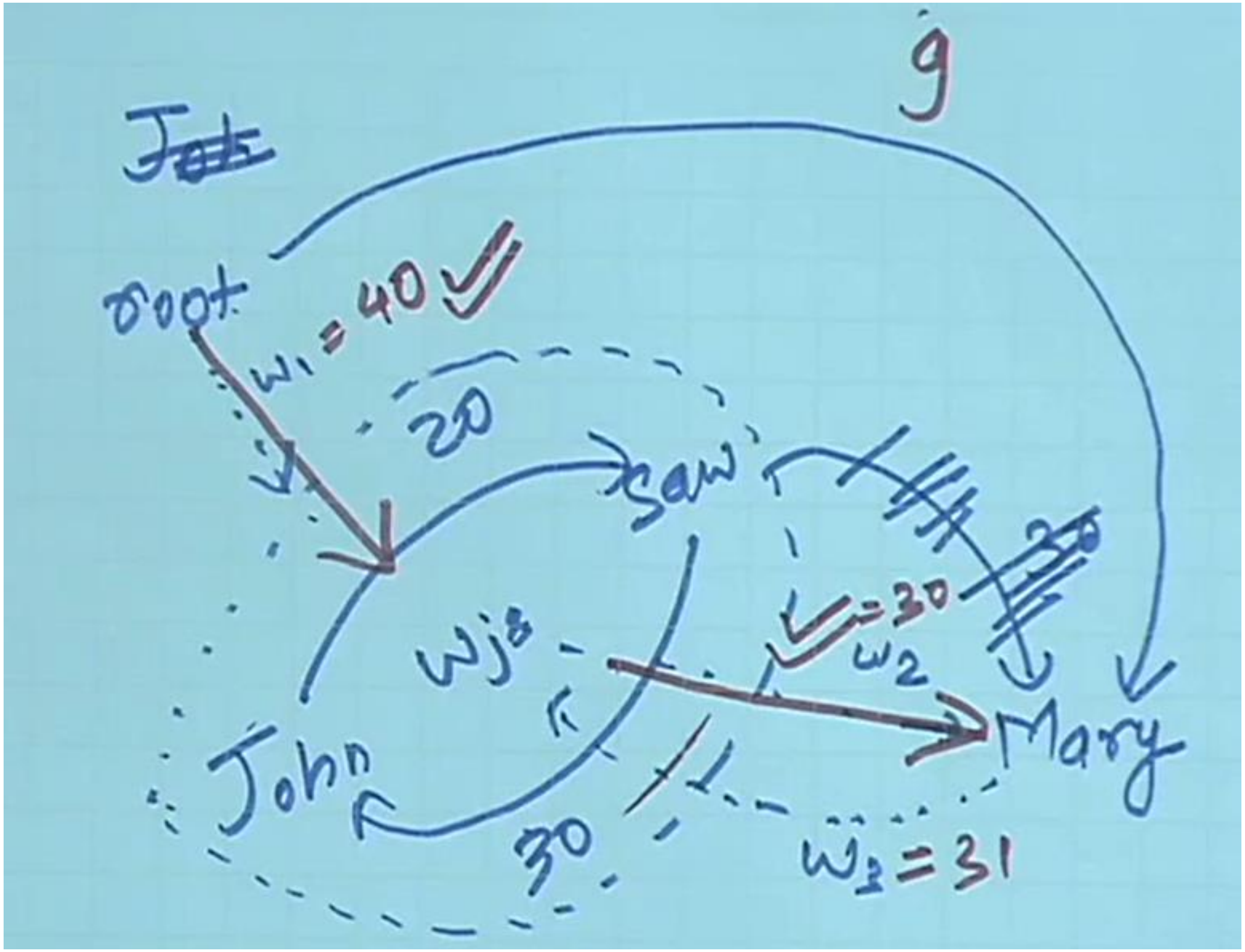
- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John → Mary is 3 and saw → Mary is 30

Chu-Liu-Edmonds Example



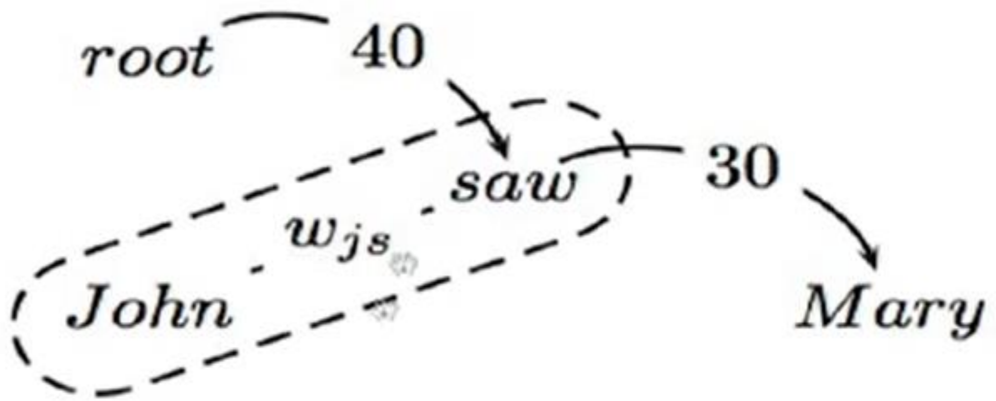
- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root $\longrightarrow$ saw $\longrightarrow$ John is 40 (**)
  - root $\longrightarrow$ John $\longrightarrow$ saw is 29

Chu-Liu-Edmonds Example



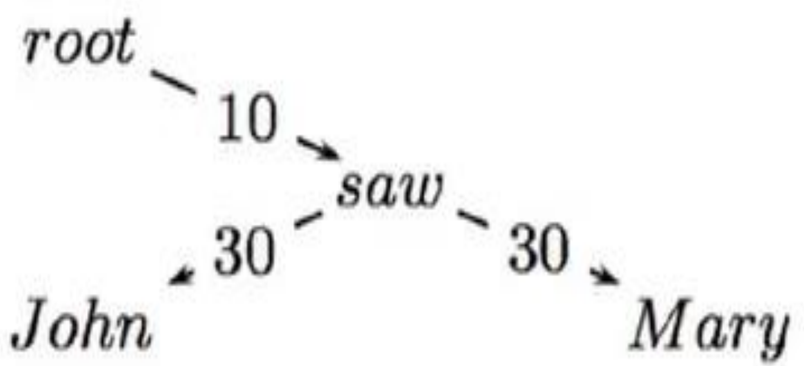
Chu-Liu-Edmonds Example

Calling the algorithm again on the contracted graph:



- This is a tree and the MST for the contracted graph
- Go back up the recursive call and reconstruct final graph

Chu-Liu-Edmonds Example



- The edge from w_{js} to *Mary* was from *saw*
- The edge from *root* to w_{js} represented a tree from *root* to *saw* to *John*.

Linear classifiers

$$w_{ij}^{k} = w.f(i,j,k)$$

- Arc weights are a linear combination of features of the arc $f(i,j,k)$ and a corresponding weight vector w
- What arc features?

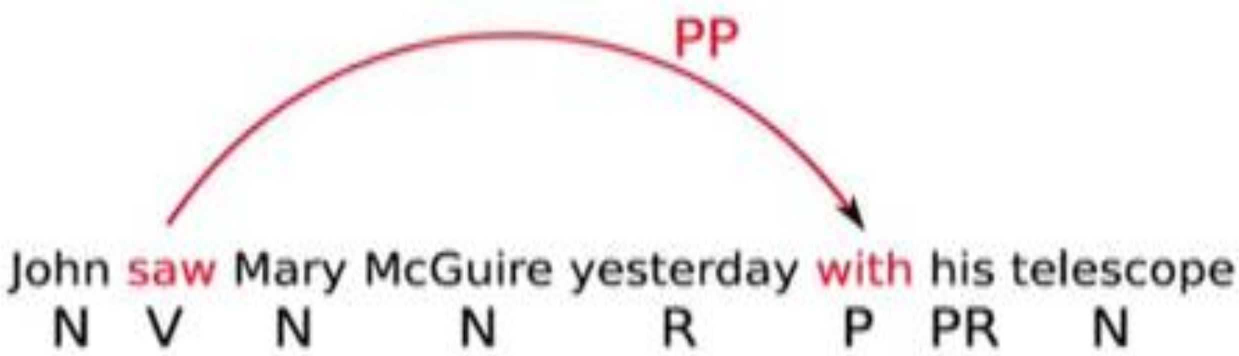
Arc features

Features

Identities of the words w_i and w_j for a label l_k

head = saw & dependent=with

Arc features



Features

Part-of-speech tags of the words w_i and w_j for a label l_k

head-pos = Verb & dependent-pos=Preposition

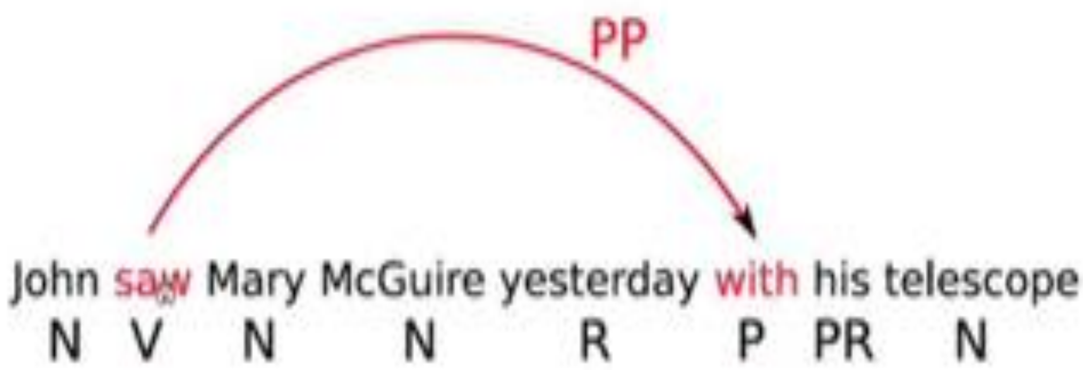
Arc features

Features

Part-of-speech of words surrounding and between w_i and w_j

inbetween-pos = Noun
inbetween-pos = Adverb
dependent-pos-right = Pronoun
head-pos-left=Noun

Arc features



Features

Number of words between w_i and w_j, and their orientation

arc-distance = 3

arc-direction = right

Learning the parameters

- Re-write the inference problem

$$G = \arg\max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w_{ij}^{k}$$

$$= \arg\max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k)$$

$$= \arg\max_{G \in T(G_x)} w \cdot f(G)$$

Inference based learning

Training data: $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

$w^{(0)} = 0; i = 0$

for n : $1..N$

  for t : $1..|T|$

    Let $G' = argmax_{G'} w^{(i)}.f(G')$

    if $G' \neq G_t$

      $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$

      $i = i + 1$

return w^i

EXAMPLE

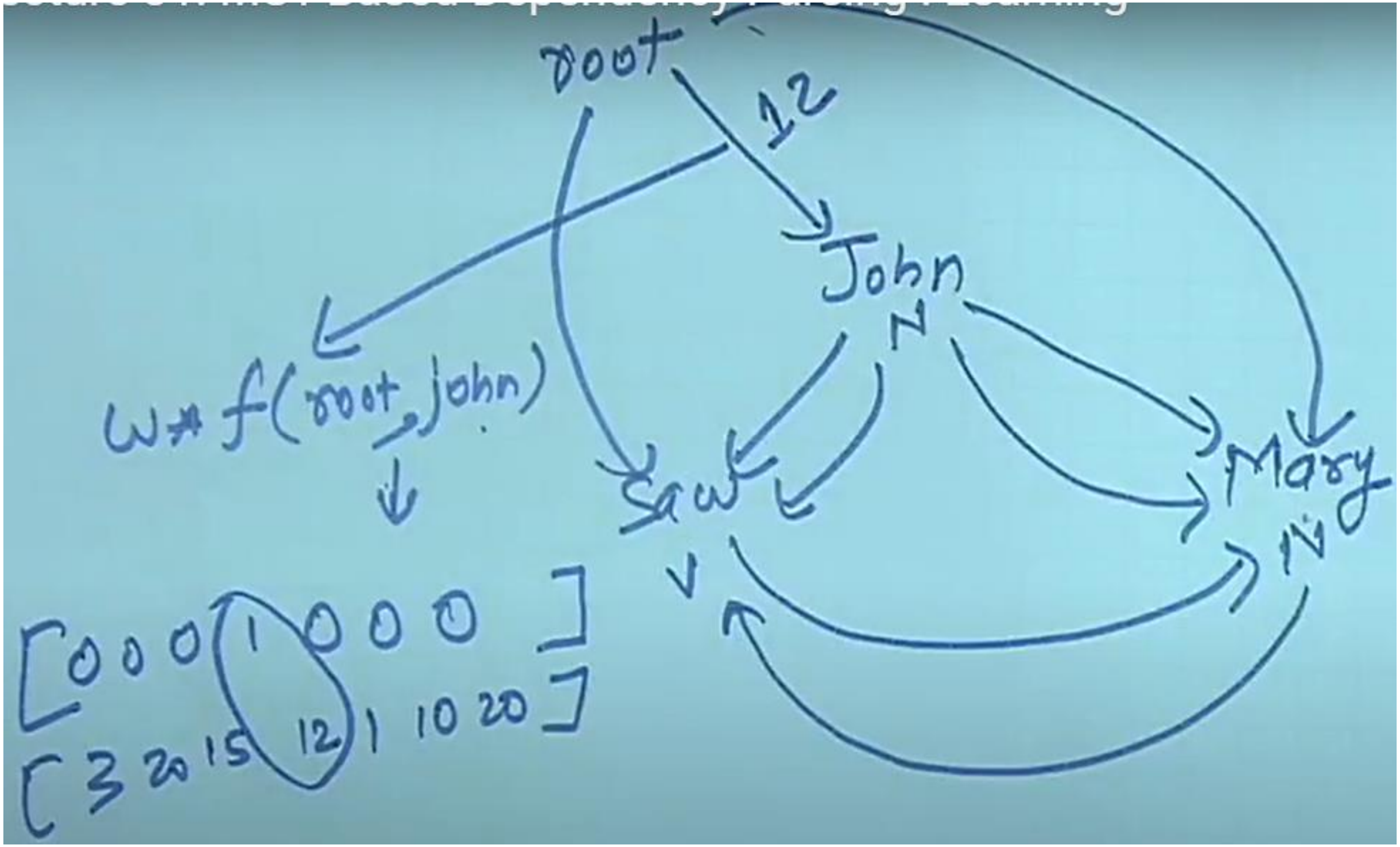
Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word w_i to w_j, your features may be simplified to depend only on words w_i and w_j and not on the relation label.

Below is the set of features

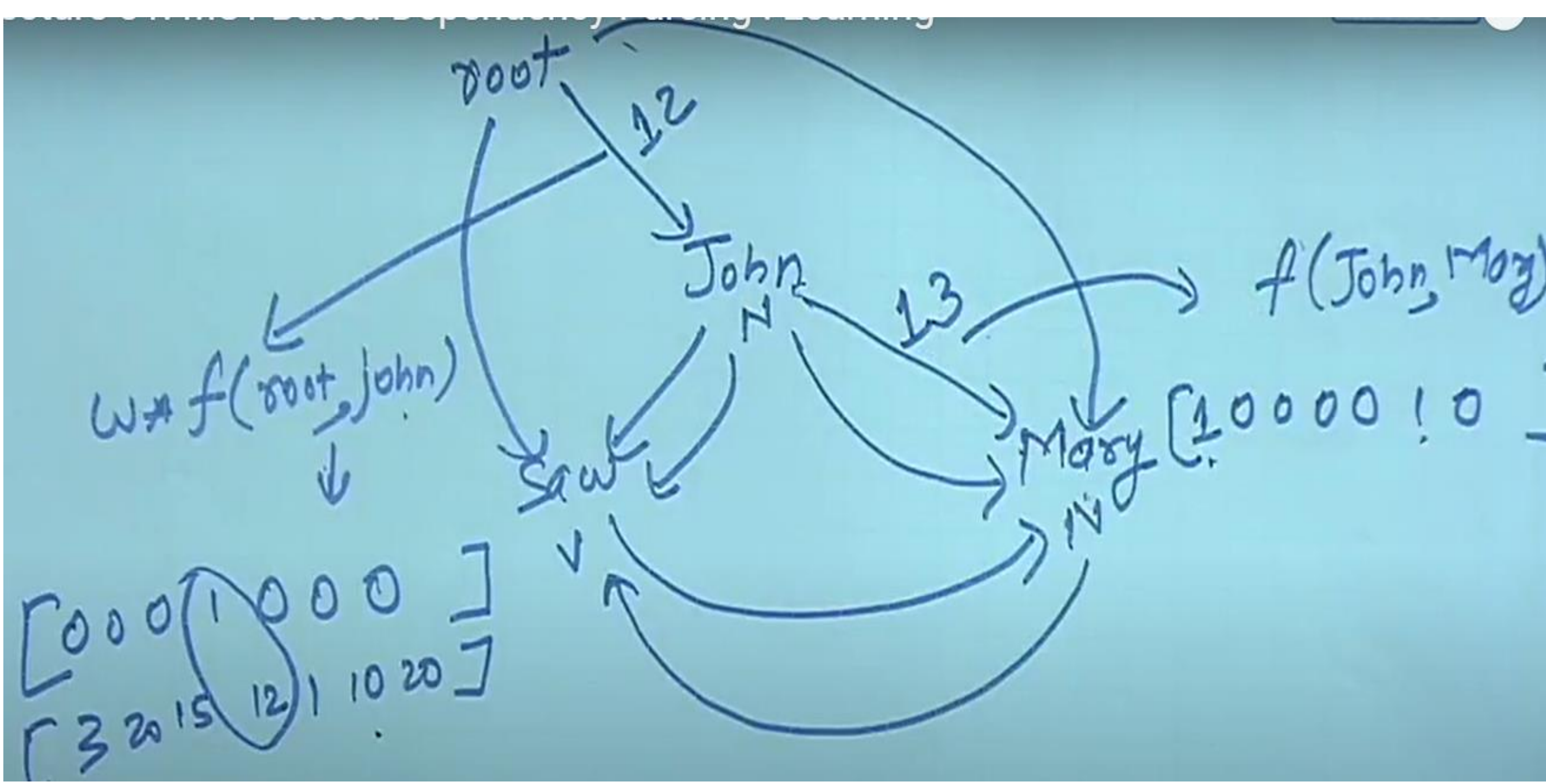
- f_1: pos(w_i) = Noun and pos(w_j) = Noun
- f_2: pos(w_i) = Verb and pos(w_j) = Noun
- f_3: w_i = Root and pos(w_j) = Verb
- f_4: w_i = Root and pos(w_j) = Noun
- f_5: w_i = Root and w_j occurs at the end of sentence
- f_6: w_i occurs before w_j in the sentence
- f_7: pos(w_i) = Noun and pos(w_j) = Verb

The feature weights before the start of the iteration are: $\{3, 20, 15, 12, 1, 10, 20\}$. Determine the weights after an iteration over this example.

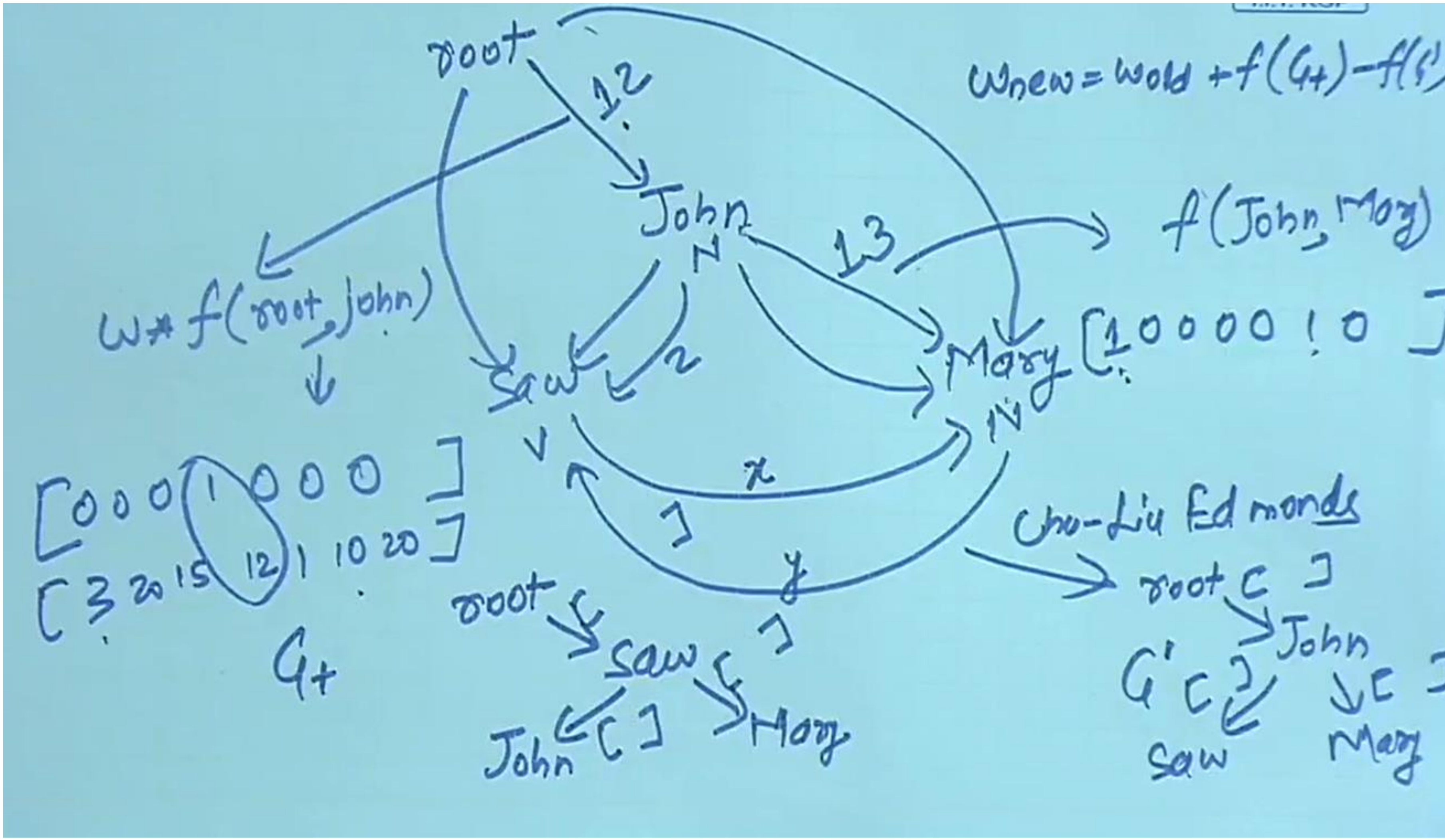
EXAMPLE



EXAMPLE

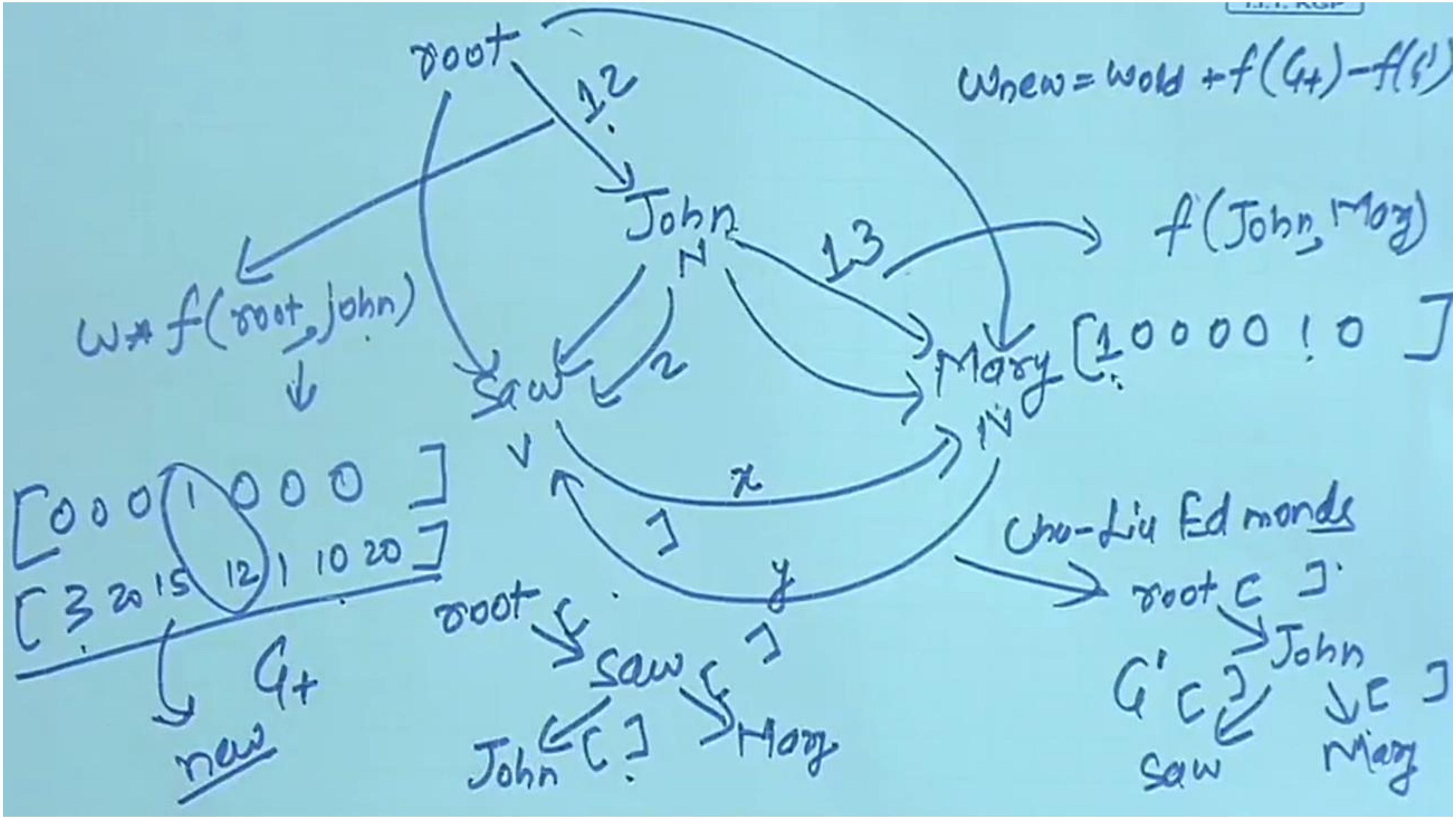

root
12
John
13
f(John, Mary)
w * f(root, john)
Saw
Mary
[1 0 0 0 0 1 0]
[0 0 0 1 0 0 0]
[3 20 15 12 1 10 20]

EXAMPLE



root
John
Saw
Mary
12
13
2
x
y
wnew = wold + f(G+) − f(G')
f(John, Mary)
w*f(root, John)
Chu-Liu Edmonds
G+
G'

EXAMPLE


root
12
John
N
13
Mary
N
Saw
V
2
x
y
Wnew = Wold + f(G+) - f(G')
f(John, Mary)
[1 0 0 0 0 1 0]
w*f(root, john)
[0 0 0 1 0 0 0]
[3 20 15 12 1 10 20]
G+
new
root
saw
John
Mary
Chu-Liu Edmonds
root
John
G'
saw
Mary

Extra Reading

- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition].
- https://www.youtube.com/watch?v=egBq3gi_4No
- https://www.youtube.com/watch?v=R1wL7sA_hHM&list=PLzJaFd3A7DZutMK8fFxZx_mhmFQgzijGE&index=28
- https://www.researchgate.net/publication/328731166_Weighted_Machine_Learning
- https://nptel.ac.in/courses/106105158

- https://realpython.com/natural-language-processing-spacy-python/

Thank you for your time!!