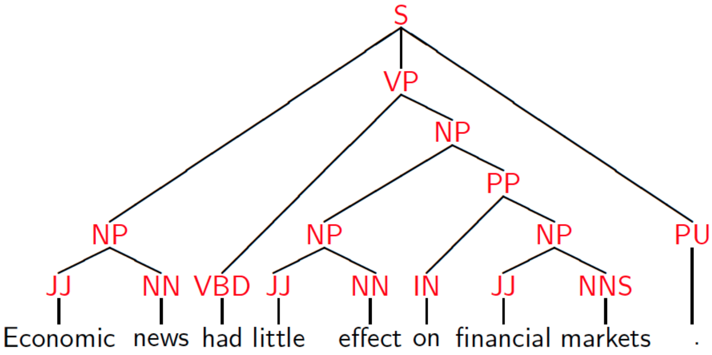
Dependency Grammars and Parsing - Introduction

Niloy Ganguly

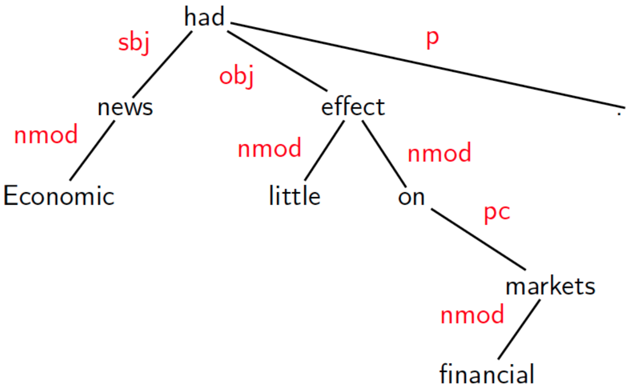
CSE, IIT Kharagpur

Week 6, Lecture 1

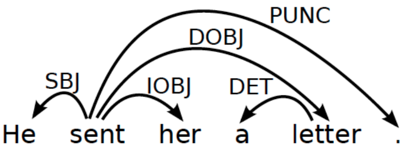
Phrase Structure



Dependency Structure Representation

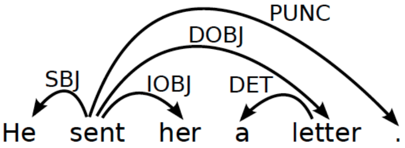


Dependency Structure



- Connects the words in a sentence by putting arrows between the words.
- Arrows show relations between the words and are typed by some grammatical relations.
- Arrows connect a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate).
- Usually dependencies form a tree.

Criteria for Heads and Dependents

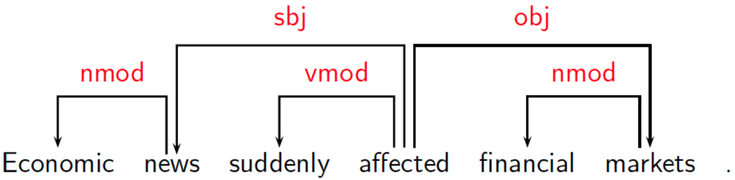


Criteria for a syntactic relation between a head H and a dependent D in a construction C

- H determines the syntactic category of C; H can replace C.
- D specifies H.
- H is obligatory; D may be optional.
- H selects D and determines whether D is obligatory.
- The form of D depends on H (agreement or government).
- The linear position of D is specified with reference to H.

Some Clear Cases

Construction	Head	Dependent
Exocentric	Verb	Subject (sbj)
	Verb	Object (obj)
Endocentric	Verb	Adverbial (vmod)
	Noun	Attribute (nmod)



Comparison

Phrase structures explicitly represent

- Phrases (nonterminal nodes)
- Structural categories (nonterminal labels)

Comparison

Phrase structures explicitly represent

- Phrases (nonterminal nodes)
- Structural categories (nonterminal labels)

Dependency structures explicitly represent

- Head-dependent relations (directed arcs)
- Functional categories (arc labels)

Dependency Graphs

- A dependency structure can be defined as a directed graph G, consisting of
  - a set V of nodes,
  - a set A of arcs (edges),

Dependency Graphs

- 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.

Dependency Graphs

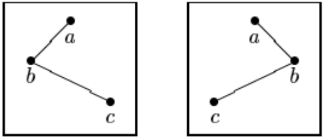
- A dependency structure can be defined as a directed graph G, consisting of
  - a set V of nodes,
  - a set A of arcs (edges),
- Labeled graphs:
  - Nodes in V are labeled with word forms (and annotation).
  - Arcs in A are labeled with dependency types.
- Notational convention:
  - Arc (w_i, d, w_j) 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 Graphs

- 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.

Formal conditions on Dependency Graphs

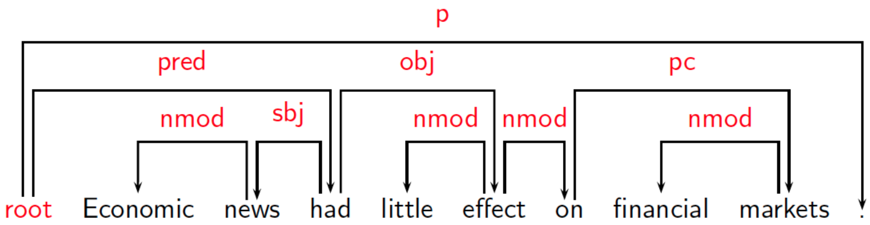
- G is connected:
  - For every node i there is a node j such that $i \rightarrow j$ or $j \rightarrow i$.
- G is acyclic:
  - if $i \rightarrow j$ then not $j \rightarrow^* i$.
- G obeys the single head constraint:
  - if $i \rightarrow j$ then not $k \rightarrow j$, for any $k \neq i$.
- G is projective: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



Formal Conditions: Basic Intuitions

Connectedness, Acyclicity and Single-Head

- **Connectedness:** Syntactic structure is complete.
- **Acyclicity:** Syntactic structure is hierarchical.
- **Single-Head:** Every word has at most one syntactic head.
- **Projectivity:** No crossing of dependencies.



Dependency Parsing

Dependency Parsing

- **Input:** Sentence $x = w_1, \ldots, w_n$
- **Output:** Dependency graph G

Parsing Methods

- Deterministic Parsing
- Maximum Spanning Tree Based
- Constraint Propagation Based

Transition Based Parsing: Formulation

Niloy Ganguly

CSE, IIT Kharagpur

Week 6, Lecture 2

Deterministic Parsing

Basic idea

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

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

Transition System

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 $t : C \rightarrow C$,
- c_s is an initialization function
- $C_t \subseteq C$ is a set of terminal configurations.

Transition System

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 $t \in T$

Transition System

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 $t : C \rightarrow C$,
- 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: $(S, []_B, A)$

Transitions for Arc-Eager Parsing

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

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

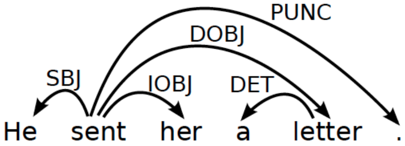
$$\text{Reduce}\ \frac{([\ldots, w_i]_S,\ B,\ A)}{([\ldots]_S,\ B,\ A)}\ \text{HEAD}(w_i)$$

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

Parse Example

Transitions:

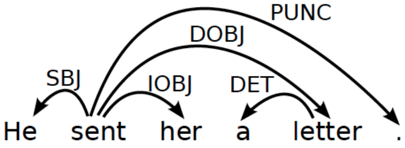
Stack	Buffer	Arcs
$[\ ]_S$	$[\text{He, sent, her, a, letter, .}]_B$	



Parse Example

Transitions: SH

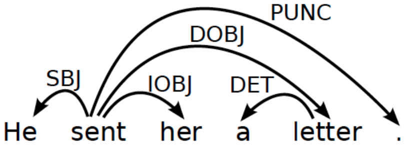
Stack	**Buffer**	**Arcs**
$[\text{He}]_S$	$[\text{sent, her, a, letter, .}]_B$	



Parse Example

Transitions: SH-LA

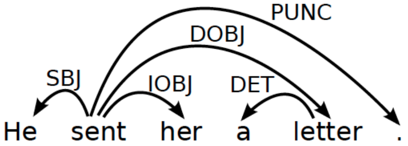
Stack	**Buffer**	**Arcs**
$[\,]_S$	$[\text{sent, her, a, letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent



Parse Example

Transitions: SH-LA-SH

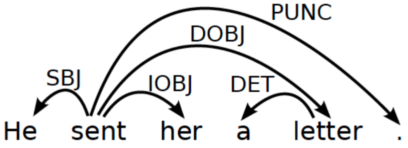
Stack	**Buffer**	**Arcs**
$[\text{sent}]_S$	$[\text{her, a, letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent



Parse Example

Transitions: SH-LA-SH-RA

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

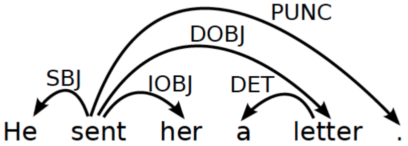


Example

Parse Example

Transitions: SH-LA-SH-RA-SH

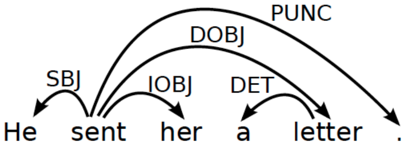
Stack	**Buffer**	**Arcs**
$[\text{sent, her, a}]_S$	$[\text{letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent sent $\xrightarrow{\text{IOBJ}}$ her



Parse Example

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

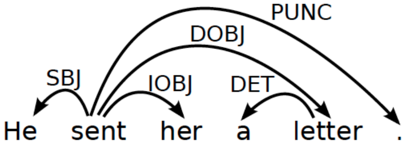
Stack	**Buffer**	**Arcs**
$[\text{sent, her}]_S$	$[\text{letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter



Parse Example

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

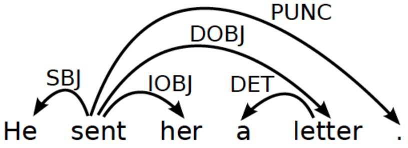
Stack	**Buffer**	**Arcs**
$[\text{sent}]_S$	$[\text{letter}, .]_B$	He $\xleftarrow{\text{SBJ}}$ sent sent $\xrightarrow{\text{IOBJ}}$ her a $\xleftarrow{\text{DET}}$ letter



Parse Example

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

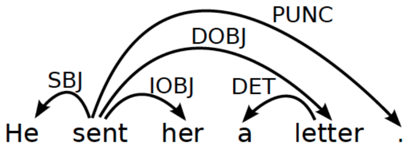
Stack	**Buffer**	**Arcs**
$[\text{sent, letter}]_S$	$[.]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter
		sent $\xrightarrow{\text{DOBJ}}$ letter



Parse Example

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

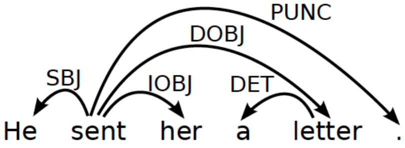
Stack	**Buffer**	**Arcs**
$[\text{sent}]_S$	$[.]_B$	He $\xleftarrow{\text{SBJ}}$ sent sent $\xrightarrow{\text{IOBJ}}$ her a $\xleftarrow{\text{DET}}$ letter sent $\xrightarrow{\text{DOBJ}}$ letter



Parse 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 $\xrightarrow{\text{PUNC}}$.



Transition Based Parsing: Learning

Niloy Ganguly

CSE, IIT Kharagpur

Week 6, Lecture 3

Classifier-Based Parsing

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.

Classifier-Based Parsing

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**.

Classifier-Based Parsing

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)

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)

Deterministic Parsing

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., $o(c) = t$).

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 top$(S_c) \leftarrow$ first(B_c) in T
- **Right-Arc** if top$(S_c) \rightarrow$ first(B_c) in T
- **Reduce** if $\exists w < top(S_c) : w \leftrightarrow$ 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.

MST-based Dependency Parsing

Niloy Ganguly

CSE, IIT Kharagpur

Week 6, Lecture 4

Maximum Spanning Tree Based

Maximum Spanning Tree Based

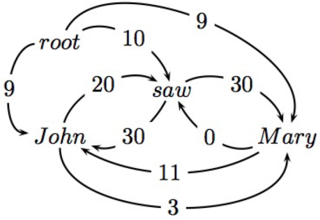
Basic Idea

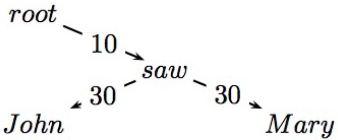
Starting from all possible connections, find the maximum spanning tree.

Maximum Spanning Tree Based

Basic Idea

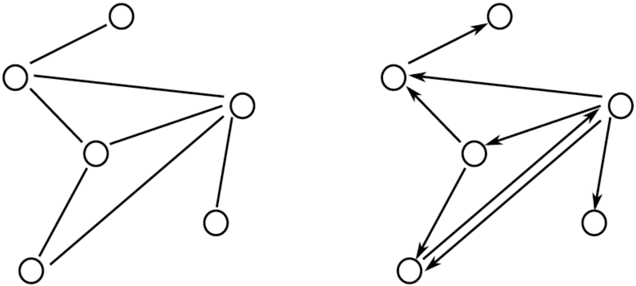
Starting from all possible connections, find the maximum spanning tree.





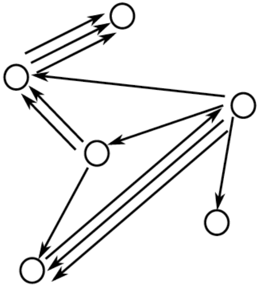
Some Graph Theory Reminders

- A graph $G = (V, A)$ is a set of vertices 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-Digraphs

- 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.

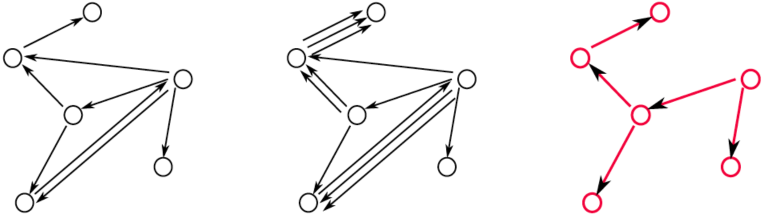


Directed Spanning Trees

- 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)

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



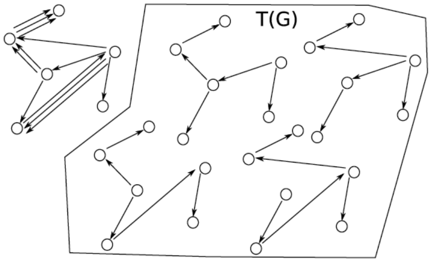
Weighted Directed Spanning Trees

- 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

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

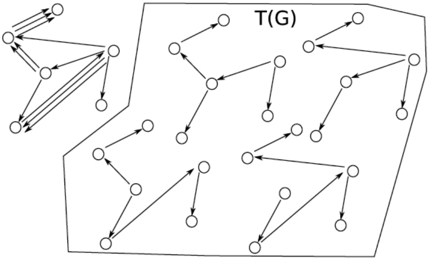
Maximum Spanning Trees (MST)

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



Maximum Spanning Trees (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

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

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

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

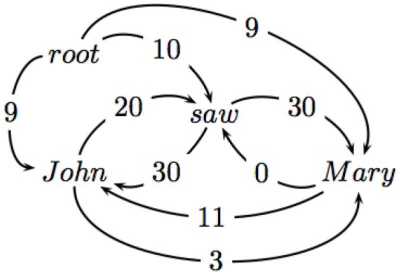
Chu-Liu-Edmonds Algorithm

- 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 Algorithm

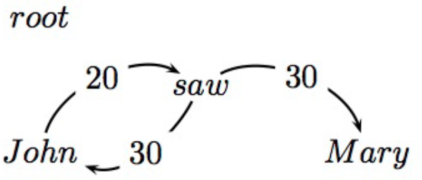
x = John saw Mary

- Build the directed graph



Chu-Liu-Edmonds Algorithm

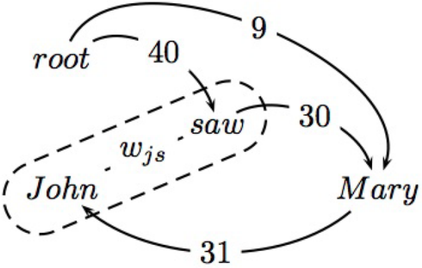
- Find the highest scoring incoming arc for each vertex



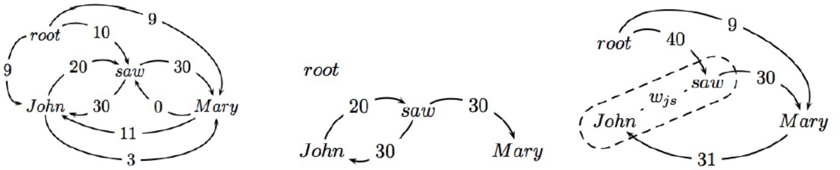
- If this is a tree, then we have found MST.

Chu-Liu-Edmonds Algorithm

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



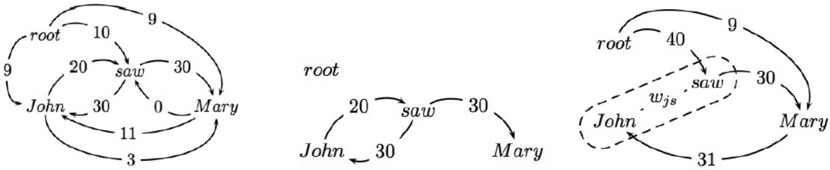
Chu-Liu-Edmonds Algorithm



Outgoing arc weights

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

Chu-Liu-Edmonds Algorithm

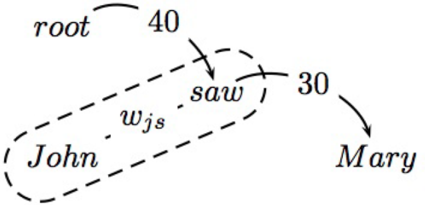


Incoming arc weights

- Equal to the weight of best spanning tree that includes head of incoming arc and all nodes in cycle
- root → saw → John is 40
- root → John → saw is 29

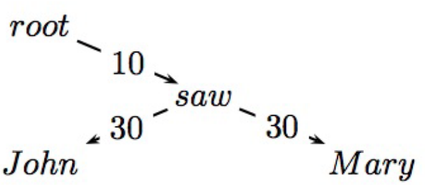
Chu-Liu-Edmonds Algorithm

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 Algorithm



- 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*.

MST-based Dependency Parsing: Learning

Niloy Ganguly

CSE, IIT Kharagpur

Week 6, Lecture 5

Arc weights as linear classifiers

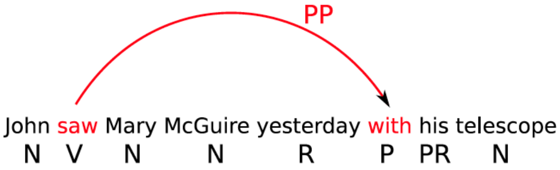
$$w_{ij}{}^{k} = w.f(i,j,k)$$

Arc weights as 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 $f(i,j,k)$

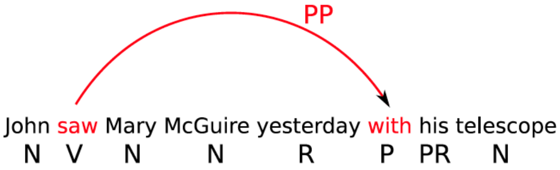


Features

Identities of the words w_i and w_j for a label l_k

head = saw & dependent=with

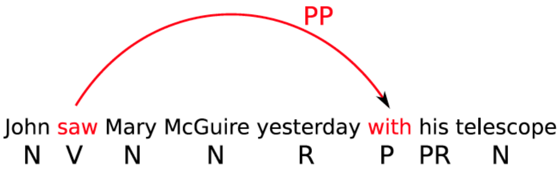
Arc Features $f(i,j,k)$



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 $f(i,j,k)$

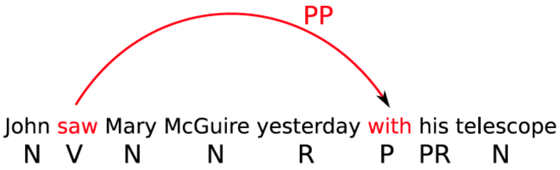


Features

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 $f(i,j,k)$



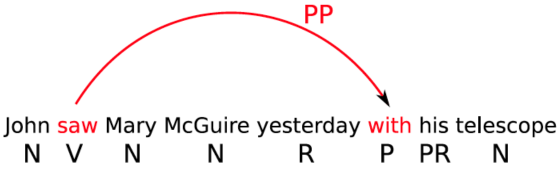
Features

Number of words between w_i and w_j, and their orientation

arc-distance = 3

arc-direction = right

Arc Features $f(i,j,k)$



Features

- Combinations
  head-pos=Verb & dependent-pos=Preposition & arc-label=PP
  head-pos=Verb & dependent=with & arc-distance=3
- No limit : any feature over arc (i,j,k) or input x

Learning the parameters

- Re-write the inference problem

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

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

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

Learning the parameters

- Re-write the inference problem

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

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

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

- Which can be plugged into learning algorithms

Inference-based Learning

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

1. $w^{(0)} = 0; i = 0$
2. for $n : 1..N$
3. for $t : 1..|T|$
4. Let $G' = argmax_{G'} w^{(i)}.f(G')$
5. if $G' \neq G_t$
6. $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7. $i = i + 1$
8. 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.