Graph-Based Dependency Parsing

Sandra Kübler, Markus Dickinson

Based on slides from Ryan McDonald and Joakim Nivre

Notation

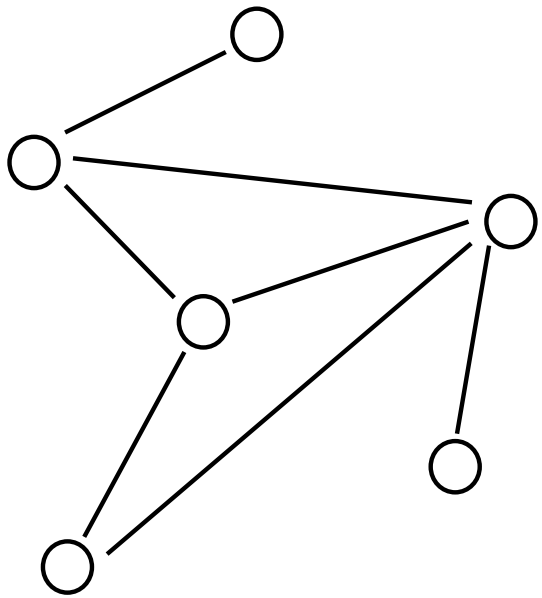
- Sentence $x = w_0, w_1, \ldots, w_n$, with $w_0 = root$
- $L = \{l_1, \ldots, l_{|L|}\}$ set of permissible arc labels
- Let $G = (V, A)$ be a dependency graph for sentence x where:
  - $V = \{0, 1, \ldots, n\}$ is the vertex set
  - A is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from w_i to w_j with label $l_k \in L$
- By the usual definition, G **is a tree**

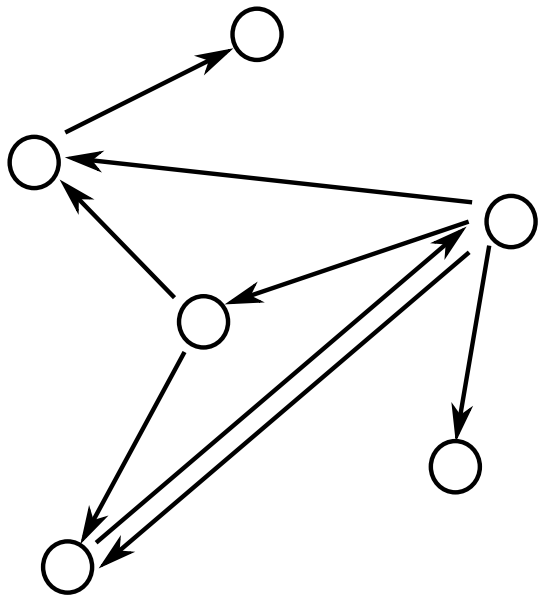
Data-Driven Parsing

- Goal: Learn a good predictor of dependency graphs
- Input: x
- Output: dependency graph/tree G
- Tuesday:
  - Parameterize parsing by transitions
  - Learn to predict transitions given the input and a history
  - Predict new graphs using deterministic parsing algorithm
- Today:
  - Parameterize parsing by dependency arcs
  - Learn to predict entire graphs given the input
  - Predict new graphs using spanning tree algorithms

Some Graph Theory

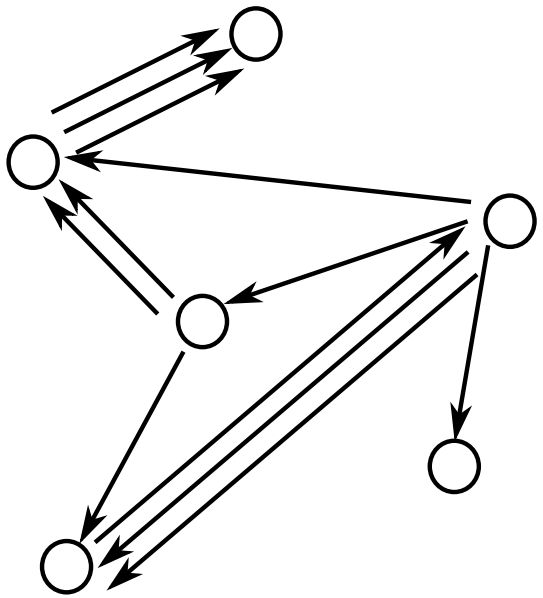
- A graph $G = (V, A)$ is a set of verteces V and arcs $(i, j) \in A$, where $i, j \in V$
- Undirected graphs: $(i, j) \in A \Leftrightarrow (j, i) \in A$
- **Directed graphs (digraphs)**: $(i, j) \in A \not\Rightarrow (j, i) \in A$





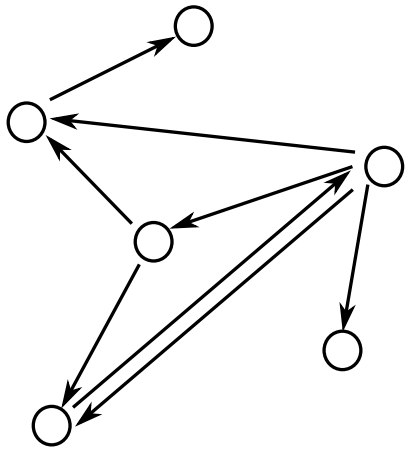
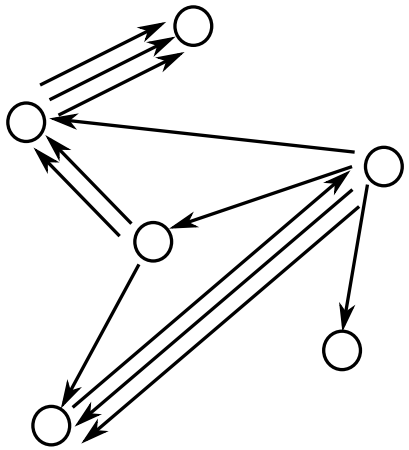
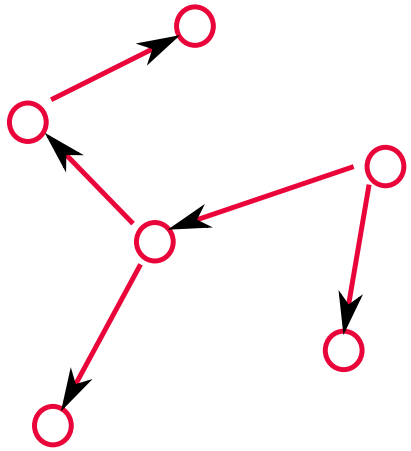
Multi-Digraphs

- A multi-digraph is a digraph where there can be multiple arcs between verteces
- $G = (V, A)$
- $(i, j, k) \in A$ represents the k^{th} arc from vertex i to vertex j



Directed Spanning Trees (a.k.a. Arborescence)

- 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

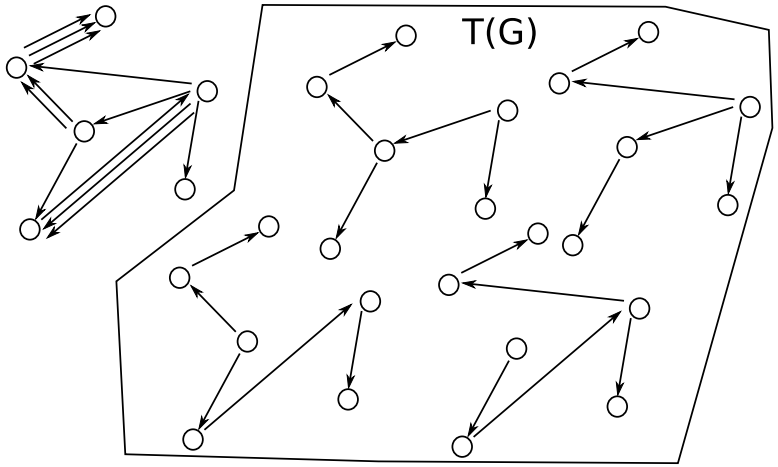
- Assume we have a weight function for each arc in a multi-digraph $G = (V, A)$
- Define $w_{ij}^k \geq 0$ to be the weight of $(i, j, k) \in A$ for a multi-digraph
- Define the weight of directed spanning tree G' of graph G as

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

- Notation: $(i, j, k) \in G = (V, A) \Leftrightarrow$ the arc $(i, j, k) \in A$

Maximum Spanning Trees (MST) of (Multi-)Digraphs

- 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 highest weight

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

- Solutions ... to come.

Arc-Factored Dependency Models

- Remember: Data-driven parsing parameterizes model and then learns parameters from data
- **Arc-factored model**
  - Assumes that the score / probability / **weight** of a dependency graph factors by its arcs

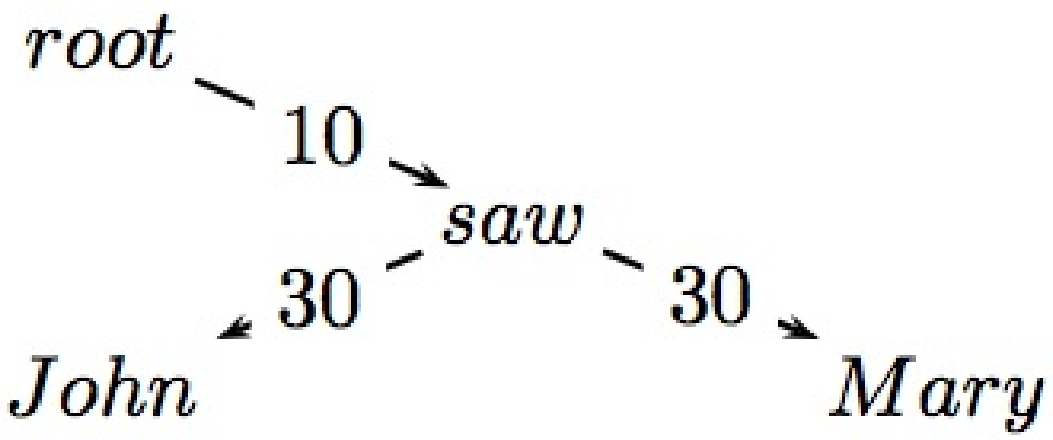
$$w(G) = \prod_{(i,j,k) \in G} w_{ij}^k$$

look familiar?

  - w_{ij}^k is the weight of creating a dependency from word w_i to w_j with label l_k
  - Thus there is an assumption that each dependency decision is independent
    - Strong assumption! Will address this later.

Arc-Factored Dependency Models Example

- Weight of dependency graph is $10 \times 30 \times 30 = 9000$



- In practice arc weights are much smaller

Three Important Problems

1. **Inference** $\equiv$ finding the MST of G_x

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

2. Defining w_{ij}^k and its **feature space**
3. **Learning** w_{ij}^k
   - Can use perceptron-based learning if we solve (1)

Chu-Liu-Edmonds Algorithm

- Finds the MST originating out of a vertex of choice
- Assumes weight of tree is **sum** of arc weights
- No problem, we can use logarithms

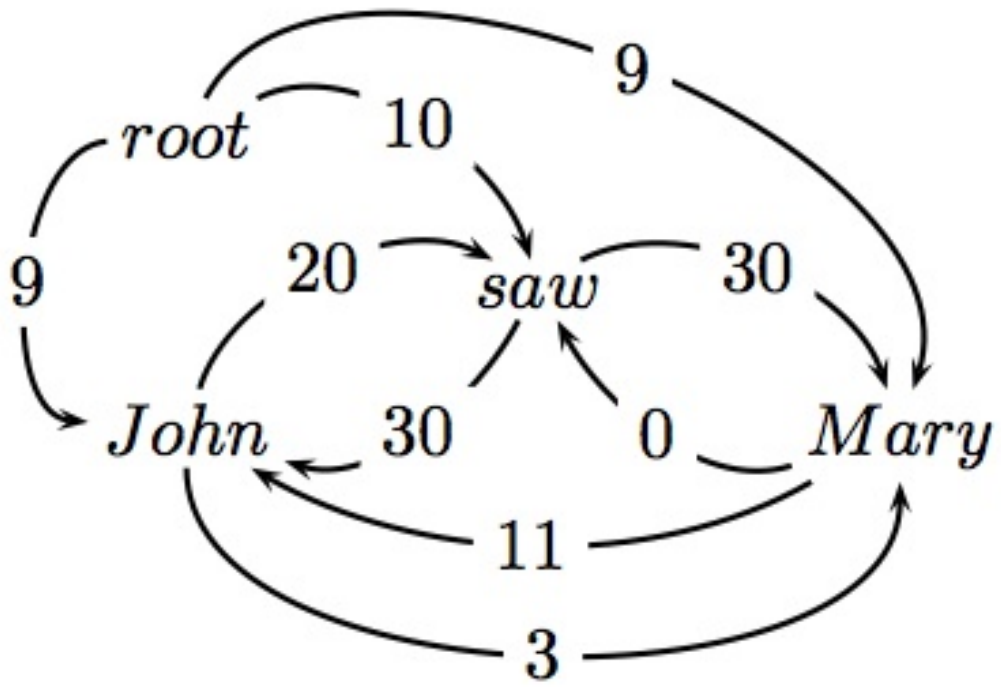
$$
\begin{aligned}
G &= \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k \\
&= \arg\max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} w_{ij}^k \\
&= \arg\max_{G \in T(G_x)} \sum_{(i,j,k) \in G} \log w_{ij}^k
\end{aligned}
$$

So if we let $w_{ij}^k = \log w_{ij}^k$, then we get

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

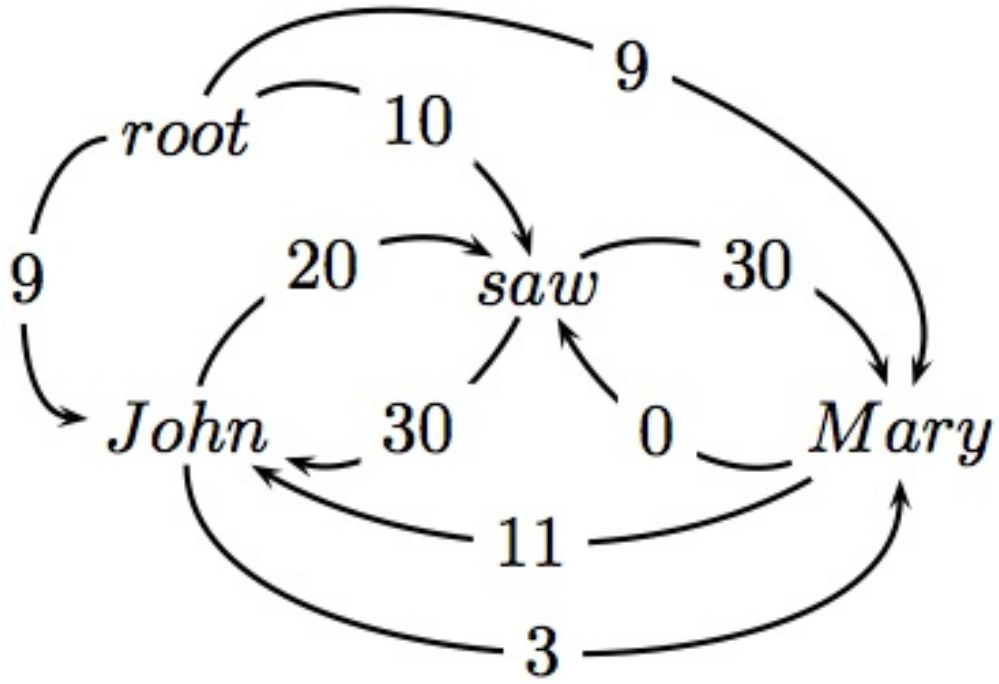
Chu-Liu-Edmonds

- x = root John saw Mary
- Remove all arcs into the root node



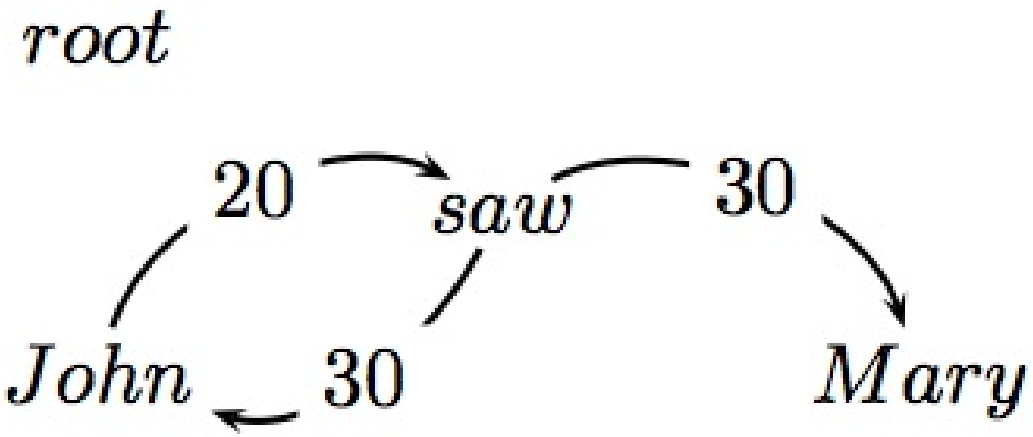
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex



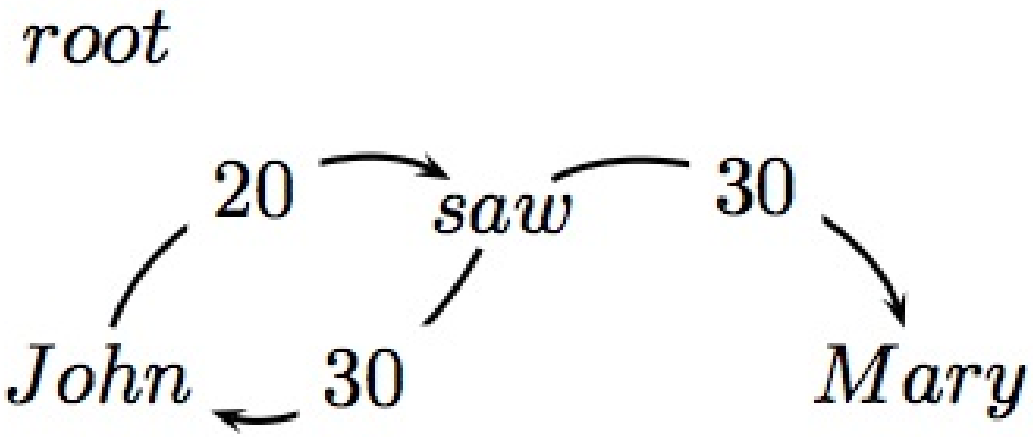
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex



Chu-Liu-Edmonds

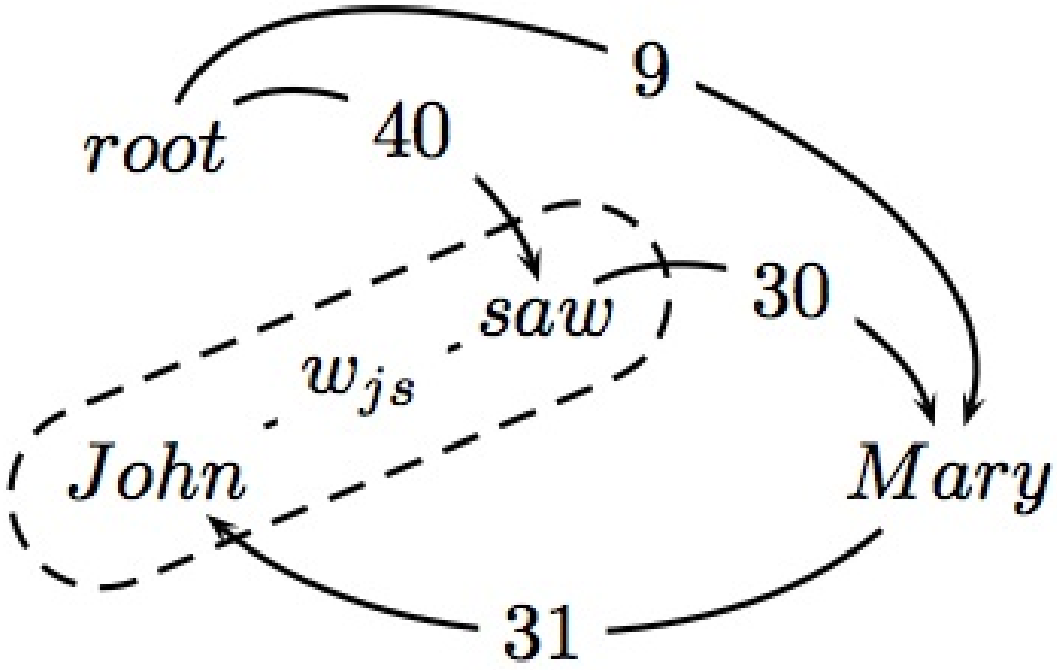
- Find highest scoring incoming arc for each vertex



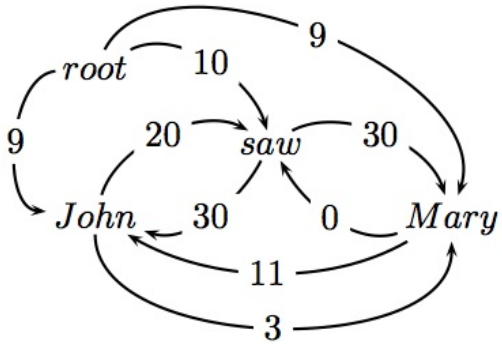
- If this is a tree, then we have found MST!!

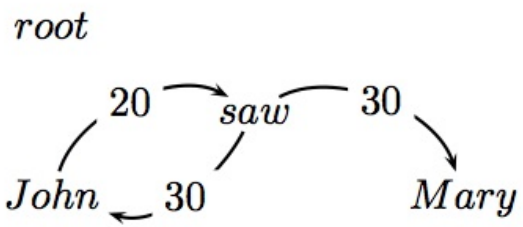
Chu-Liu-Edmonds

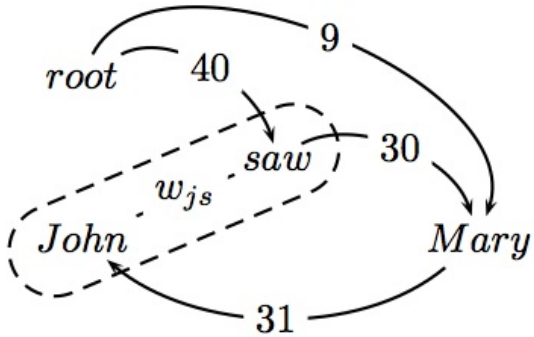
- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle



Chu-Liu-Edmonds

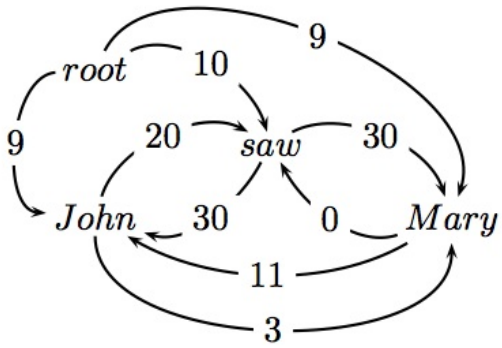


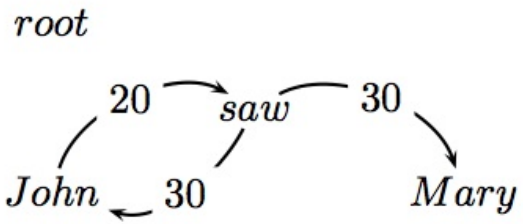


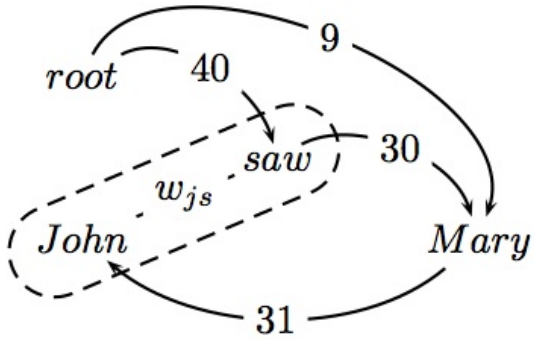


- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John $\rightarrow$ Mary is 3 and saw $\rightarrow$ Mary is 30

Chu-Liu-Edmonds



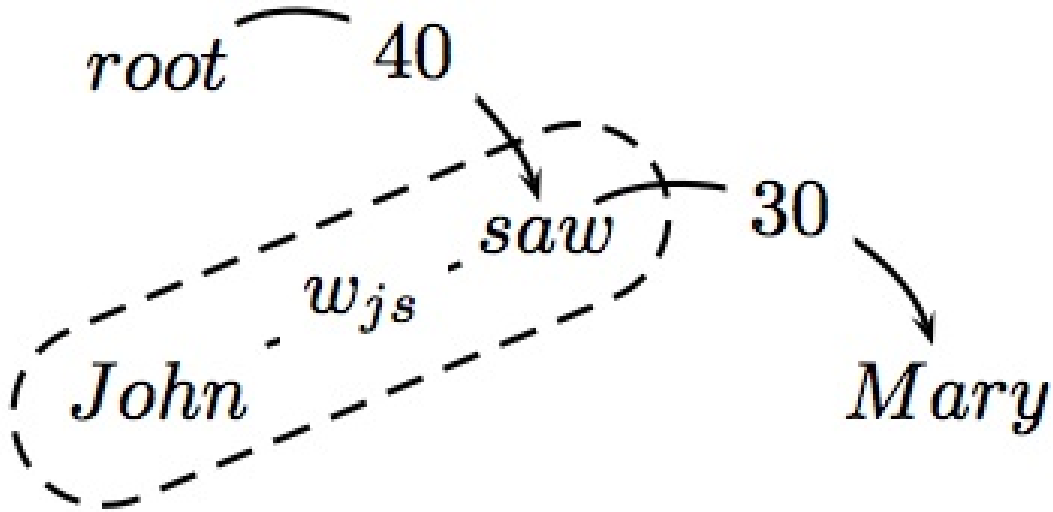




- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root $\rightarrow$ saw $\rightarrow$ John is 40 (**)
  - root $\rightarrow$ John $\rightarrow$ saw is 29

Chu-Liu-Edmonds

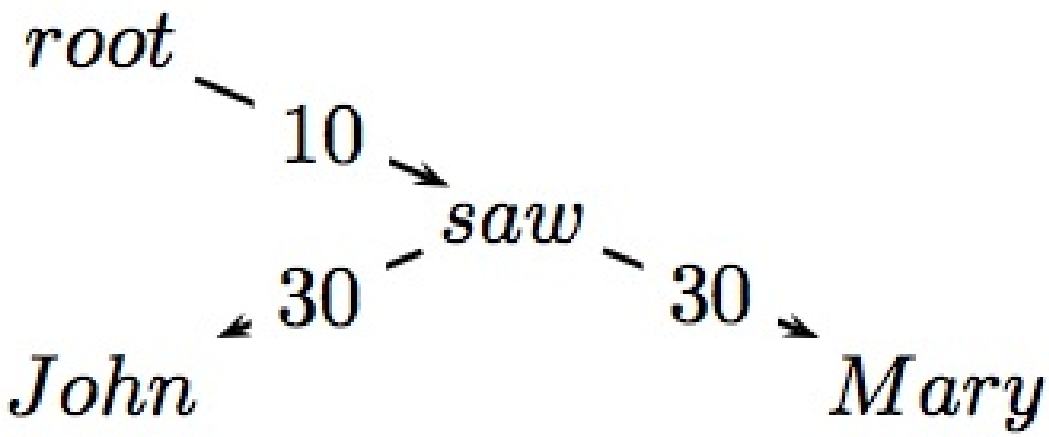
- This is a tree and the MST for the contracted graph!!



- Go back up recursive call and reconstruct final graph

Chu-Liu-Edmonds

- This is the MST!!



Chu-Liu-Edmonds

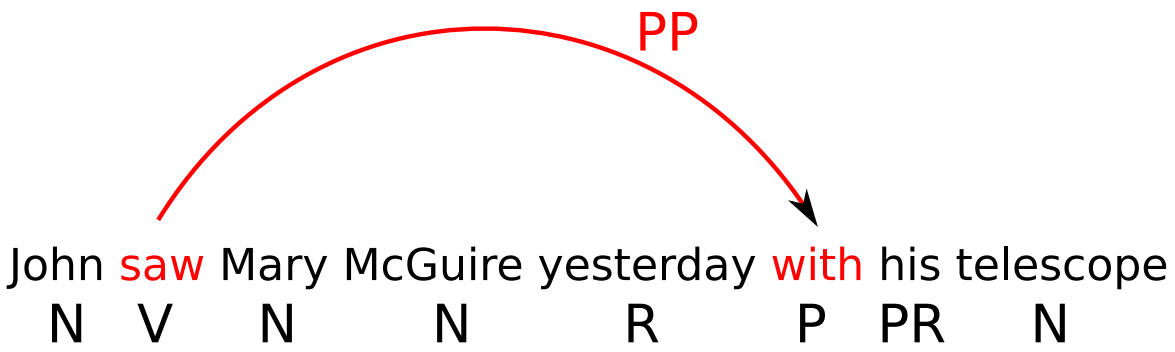
- Naive implementation $O(n^3 + |L|n^2)$
  - Converting G_x to a digraph – $O(|L|n^2)$
  - Finding best arc – $O(n^2)$
  - Contracting cycles – $O(n^2)$
  - At most n recursive calls
- Better algorithms run in $O(|L|n^2)$
- Chu-Liu-Edmonds searches all dependency graphs
  - Both projective and non-projective
  - Thus, it is an exact non-projective search algorithm!!!
- **What about the projective case?**

One more Example

- x = root If shit happens, you deserve it. (Catholicism)

head	dep.	weight	head	dep.	weight
root	shit	9	happens	it	17
root	happens	21	you	shit	13
root	you	11	you	happens	11
root	deserve	30	you	deserve	35
root	it	12	deserve	If	15
If	happens	6	deserve	shit	32
If	deserve	7	deserve	happens	29
shit	happens	13	deserve	you	33
shit	deserve	12	deserve	it	22
happens	If	20	it	shit	4
happens	shit	22	it	happens	15
happens	you	27	it	deserve	3
happens	deserve	34			

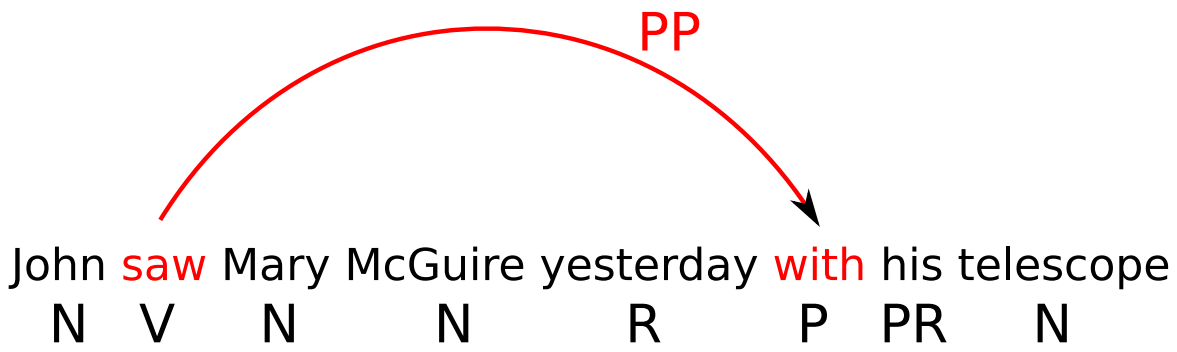
Arc Features: $\mathbf{f}(i, j, k)$



- Features from McDonald et al.
  - Identities of the words w_i and w_j and the label l_k

head=saw & dependent=with

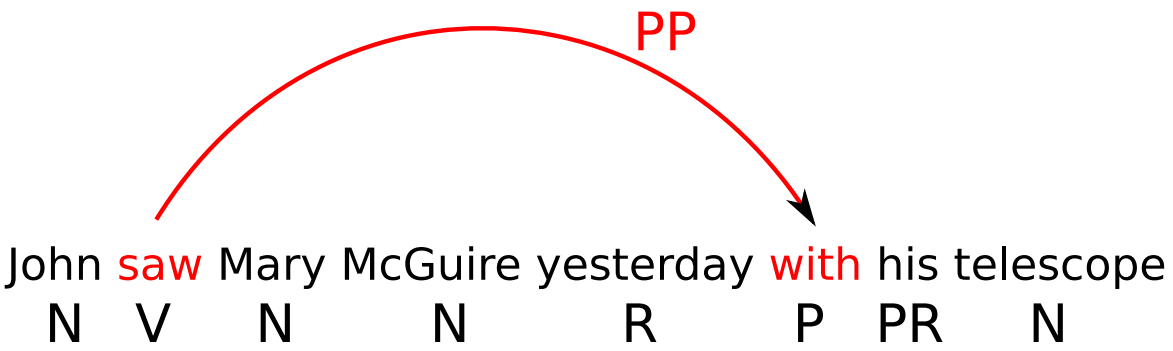
Arc Features: $\mathbf{f}(i, j, k)$



- Features from McDonald et al.
  - Part-of-speech tags of the words w_i and w_j and the label l_k

head-pos=Verb & dependent-pos=Preposition

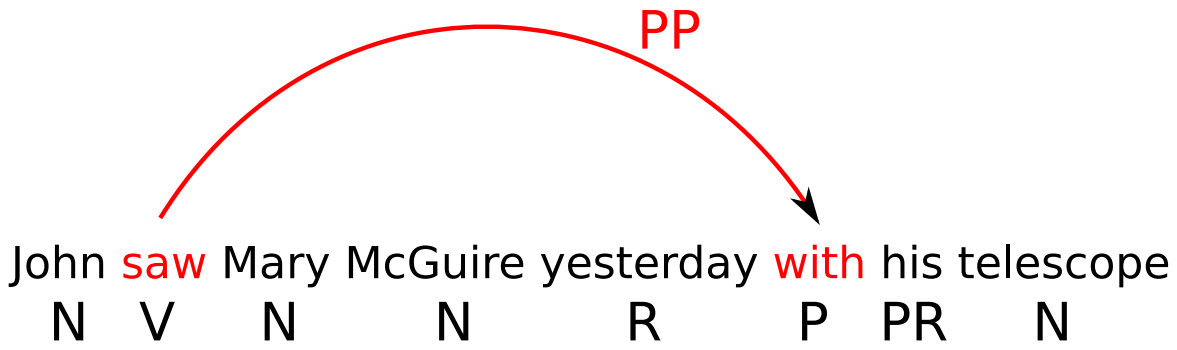
Arc Features: $\mathbf{f}(i, j, k)$



- Features from McDonald et al.
  - 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
...

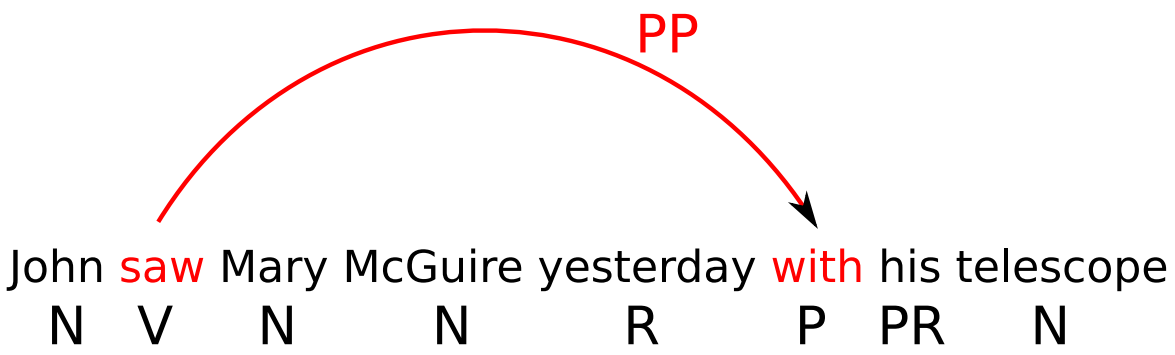
Arc Features: $\mathbf{f}(i, j, k)$



- Features from McDonald et al.
  - Number of words between w_i and w_j, and their orientation

arc-distance=3
arc-direction=right

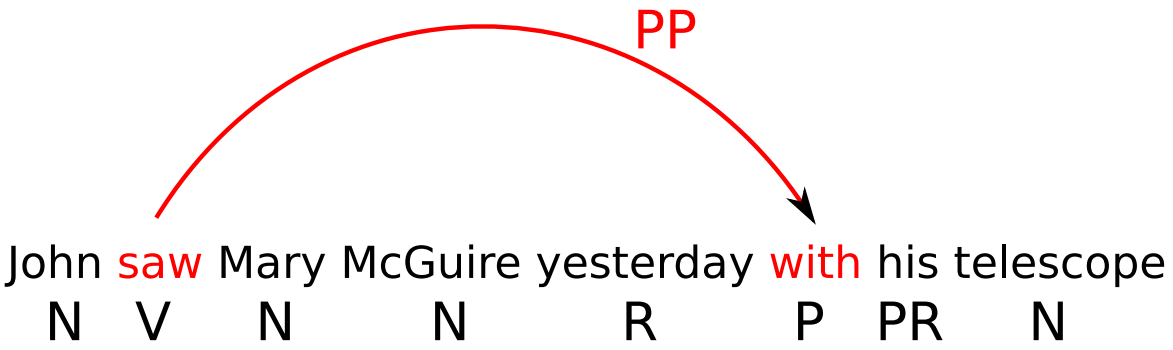
Arc Features: $\mathbf{f}(i, j, k)$



- Label features

arc-label=PP

Arc Features: $\mathbf{f}(i, j, k)$



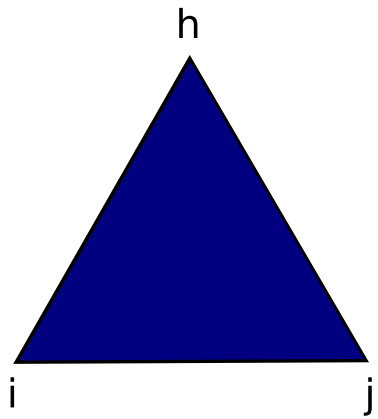
- Combos of the above

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

Arc-factored Projective Parsing

- Projective dependency structures are nested
- Can use CFG like parsing algorithms – chart parsing
- Each chart item (triangle) represents the weight of the best tree rooted at word h spanning all the words from i to j
  - Analog in CFG parsing: items represent best tree rooted at non-terminal NT spanning words i to j
- **Goal**: Find chart item rooted at 0 spanning 0 to n

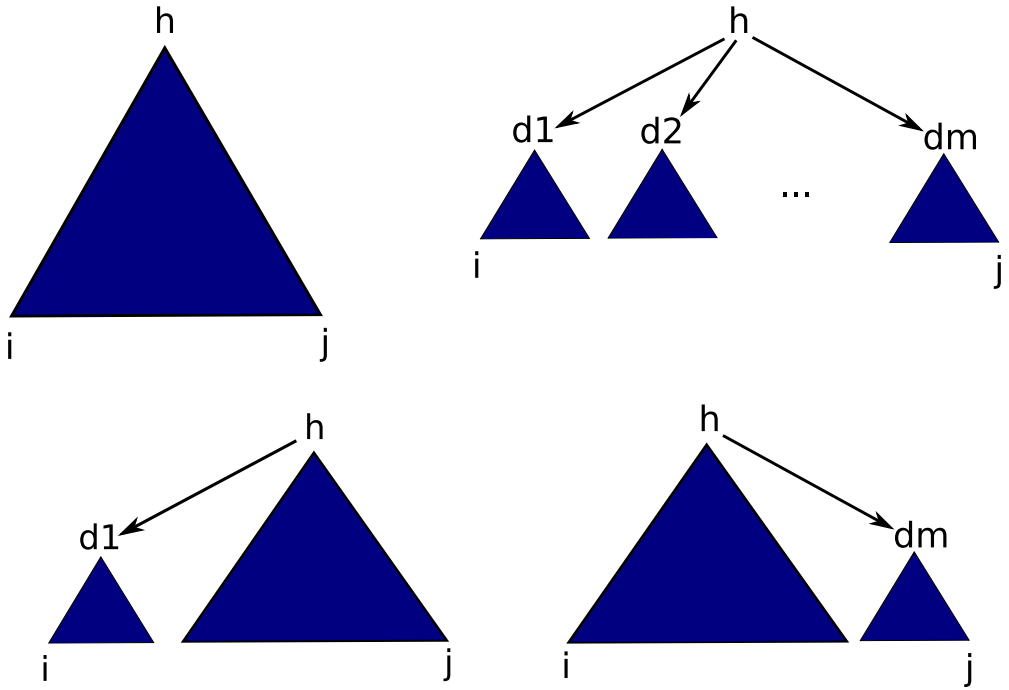


Base case

Length 1, $h = i = j$, has weight 1

Arc-factored Projective Parsing

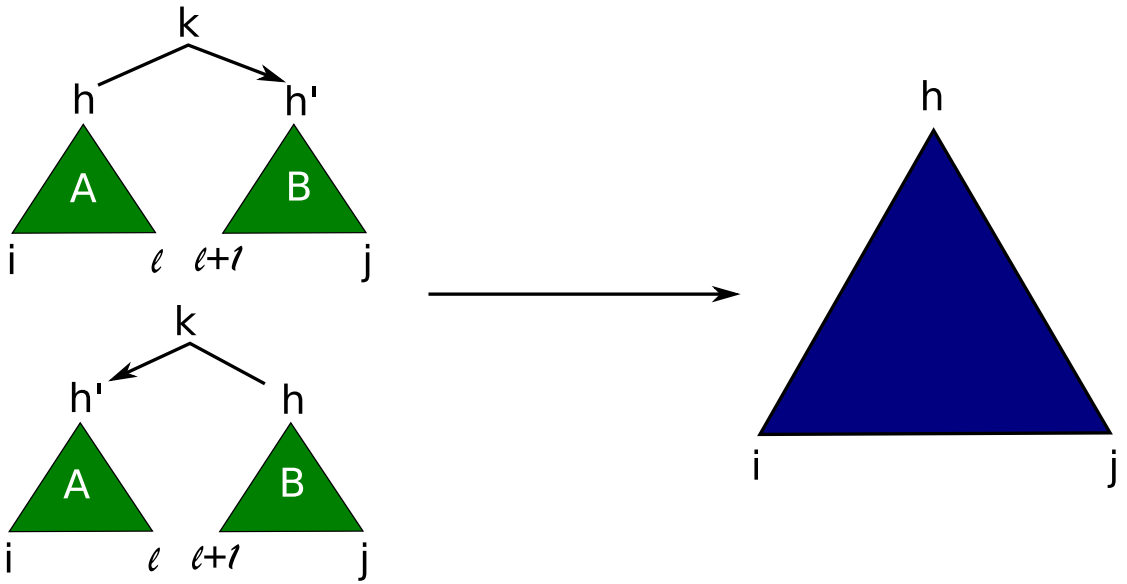
- All projective graphs can be written as the combination of two smaller adjacent graphs



- Inductive hypothesis – algorithm has calculated score of smaller items correctly (just like CKY)

Arc-factored Projective Parsing

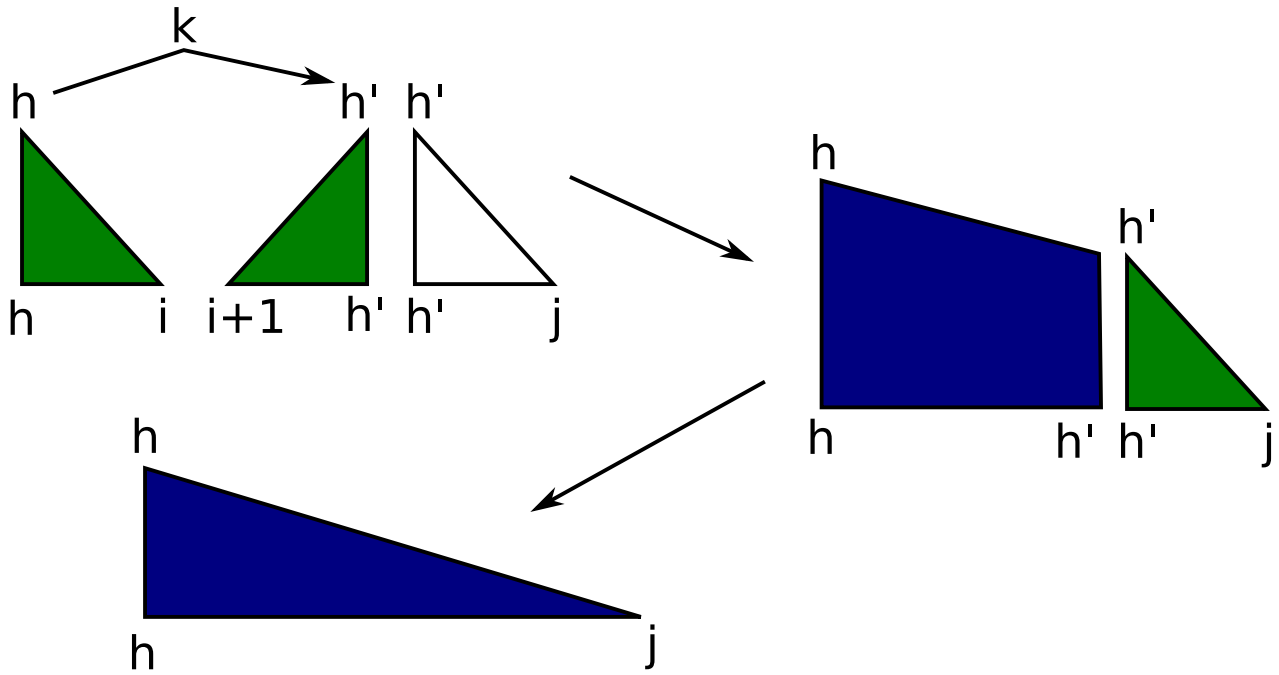
- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY



- Weight of new item: $\max_{l,j,k} \; w(A) \times w(B) \times w^k_{hh'}$
- Algorithm runs in $O(|L|n^5)$
- Use back-pointers to extract best parse (like CKY)

Arc-factored Projective Parsing

- $O(|L|n^5)$ is not that good
- Eisner showed how this can be reduced to $O(|L|n^3)$
  - Key: split items so that sub-roots are always on periphery



Inference in Arc-Factored Models

- Non-projective case
  - $O(|L|n^2)$ with the Chu-Liu-Edmonds MST algorithm
- Projective case
  - $O(|L|n^3)$ with the Eisner algorithm