Dependency Parsing

Lecture 11

Kilian Evang, Jakub Waszczuk

Heinrich Heine University Düsseldorf

Summer semester 2021

Today

- Some graph theory (slides from the course at ESSLLI 2007 by Ryan McDonald and Joakim Nivre)
- The Chu-Liu-Edmonds Algorithm (slides from the course at ESSLLI 2007 by Ryan McDonald and Joakim Nivre)

Notation Reminder

- 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

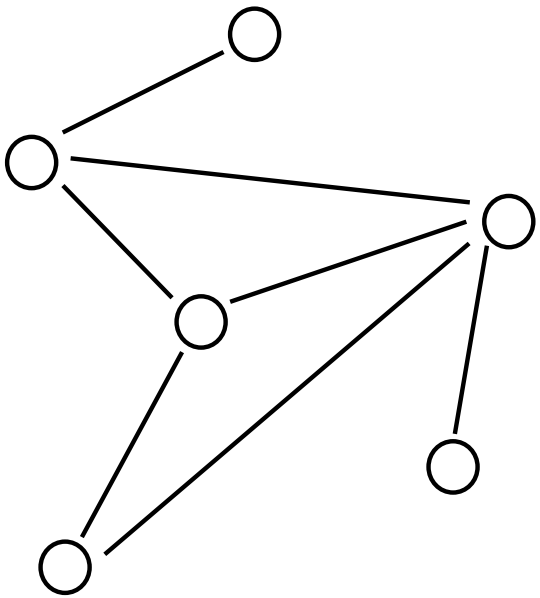
- Goal: Learn a good predictor of dependency graphs
- Input: x
- Output: dependency graph/tree G
- Last lecture:
  - Parameterize parsing by transitions
  - Learn to predict transitions given the input and a history
  - Predict new graphs using deterministic parsing algorithm
- This lecture:
  - Parameterize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

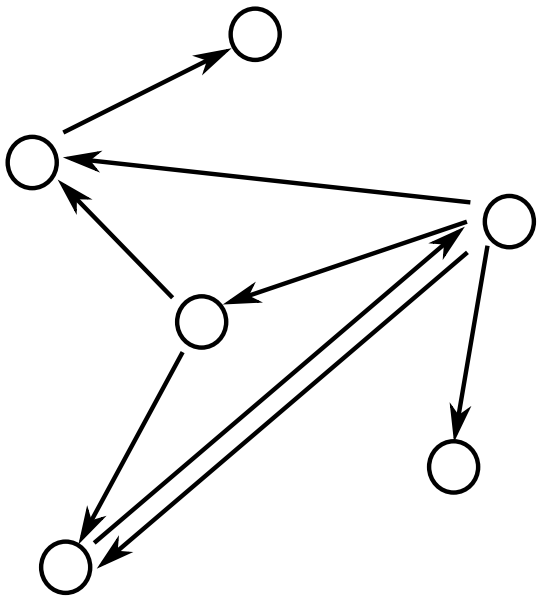
Lecture 4: Outline

- Graph theory refresher
- Arc-factored models (a.k.a. Edge-factored models)
  - Maximum spanning tree formulation
  - Projective and non-projective inference algorithms
  - Partition function and marginal algorithms – Matrix Tree Theorem
- Beyond Arc-factored Models
  - Vertical and horizontal markovization
  - Approximations

Some Graph Theory Reminders

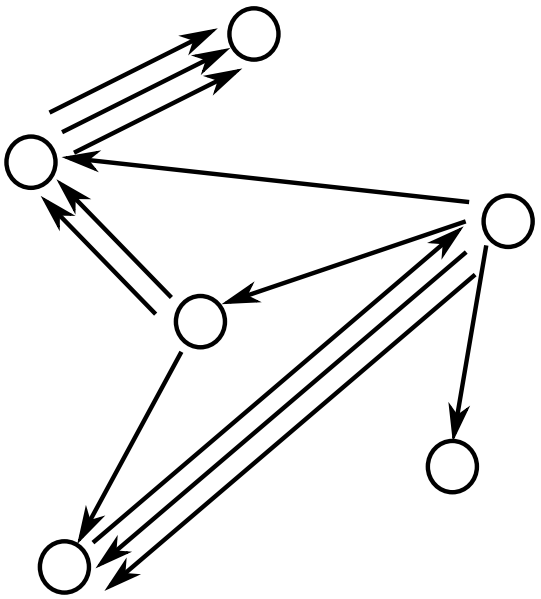
- A graph $G = (V, A)$ is a set of verteces V and arcs $(i, j) \in A$, where $i, j \in V$
- Undirected graphs: $(i, j) \in A \Leftrightarrow (j, i) \in A$
- **Directed graphs (digraphs)**: $(i, j) \in A \nRightarrow (j, i) \in A$





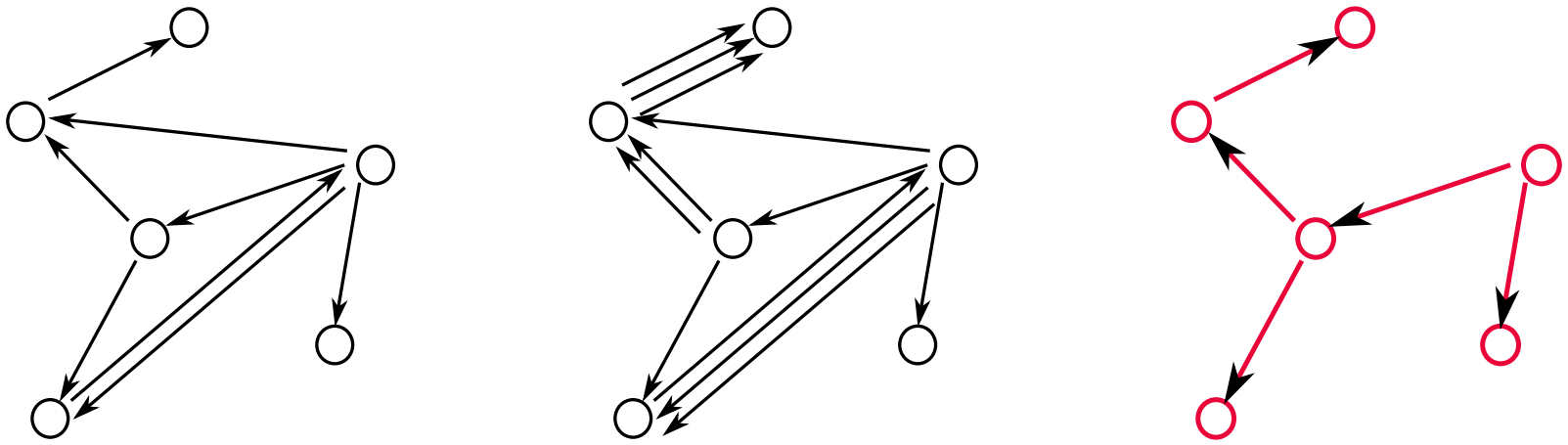
Multi-Digraphs

- A multi-digraph is a digraph where there can be multiple arcs between verteces
- $G = (V, A)$
- $(i, j, k) \in A$ represents the kth 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

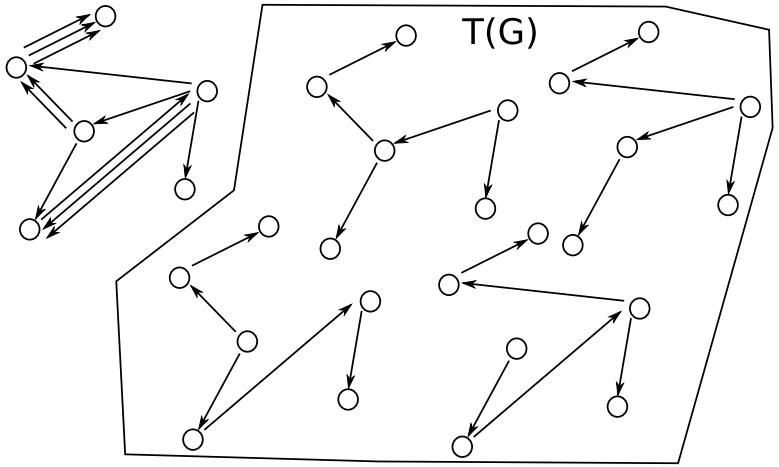
- Assume we have a weight function for each arc in a multi-digraph $G = (V, A)$
- Define $w_{ij}^k \geq 0$ to be the weight of $(i, j, k) \in A$ for a multi-digraph
- Define the weight of directed spanning tree G' of graph G as

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

- 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

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

- 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

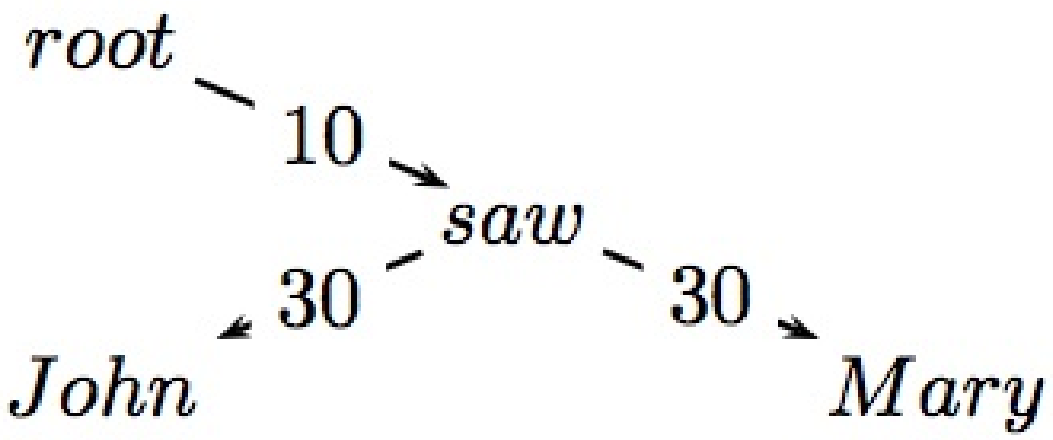
$$w(G) = \prod_{(i,j,k) \in G} w_{ij}^k$$

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

Important Concept G_x

- For input sentence $x = w_0, \ldots, w_n$, define $G_x = (V_x, A_x)$ as:
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid \forall\ i, j \in V_x \text{ and } l_k \in L\}$
- Thus, G_x is complete multi-digraph over vertex set representing words

Theorem

Every valid dependency graph for sentence x is equivalent to a directed spanning tree for G_x that originates out of vertex 0

- Falls out of definitions of tree constrained dependency graphs and spanning trees
  - Both are spanning/connected (contain all words)
  - Both are trees

Three Important Problems

Theorem

Every valid dependency graph for sentence x is equivalent to a directed spanning tree for G_x that originates out of vertex 0

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

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

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

Inference - Getting Rid of Arc Labels

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

- Consider all the arcs between vertexes i and j
- Now, consider the arc (i, j, k) such that,

$$(i, j, k) = \arg\max_k w_{ij}^k$$

Theorem

The highest weighted dependency tree for sentence x must contain the arc (i, j, k) – (assuming no ties)

- Easy proof: if not, sub in (i, j, k) and get higher weighted tree

Inference - Getting Rid of Arc Labels

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

- Thus, we can reduce G_x from a multi-digraph to a simple digraph
- Just remove all arcs that do not satisfy

$$(i, j, k) = \arg\max_k w_{ij}^k$$

- Problem is now equal to the MST problem for digraphs

We will use the **Chu-Liu-Edmonds Algorithm**

[Chu and Liu 1965, Edmonds 1967]

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

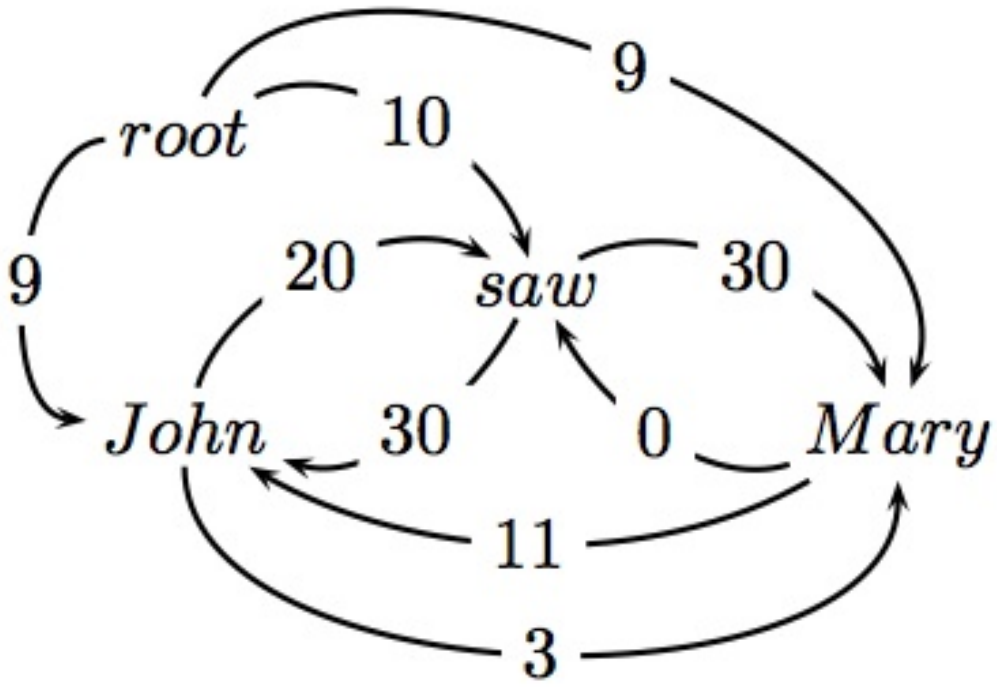
$$
\begin{aligned}
G &= \underset{G \in T(G_x)}{\arg\max} \prod_{(i,j,k) \in G} w_{ij}^k \\
&= \underset{G \in T(G_x)}{\arg\max} \log \prod_{(i,j,k) \in G} w_{ij}^k \\
&= \underset{G \in T(G_x)}{\arg\max} \sum_{(i,j,k) \in G} \log w_{ij}^k
\end{aligned}
$$

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

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

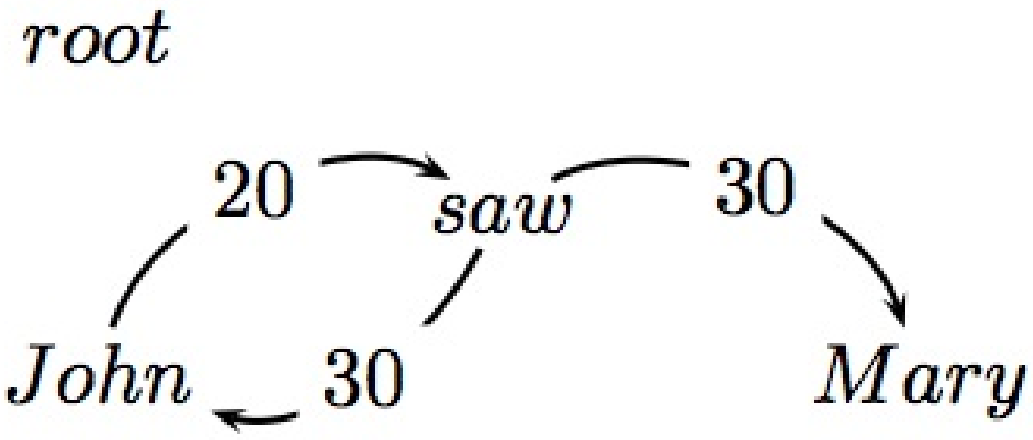
Chu-Liu-Edmonds

- x = root John saw Mary



Chu-Liu-Edmonds

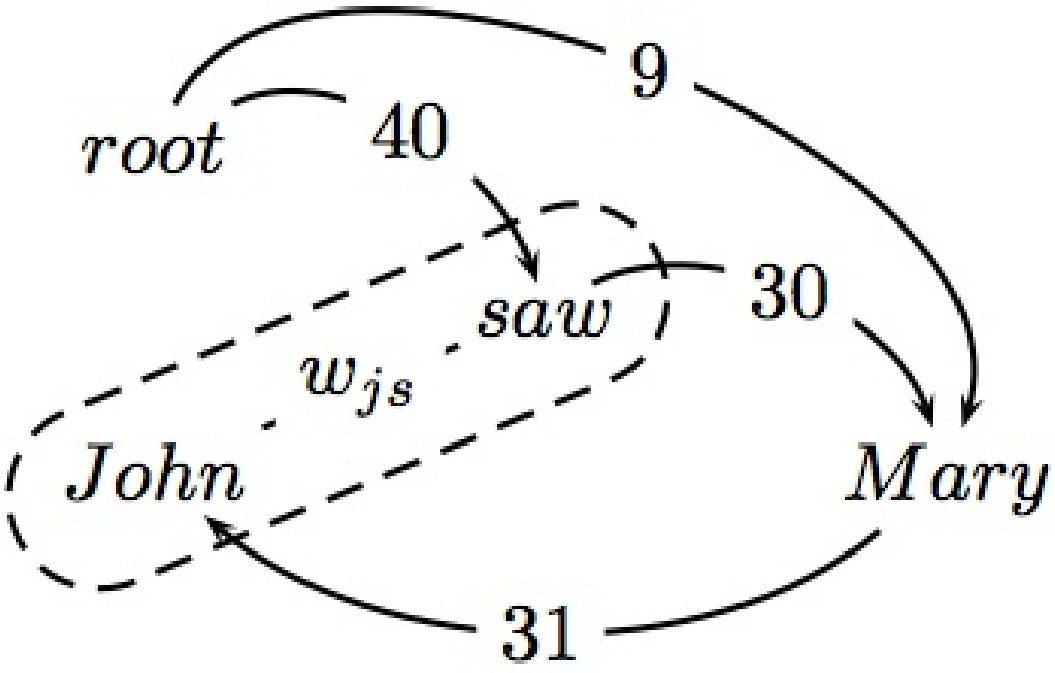
- Find highest scoring incoming arc for each vertex



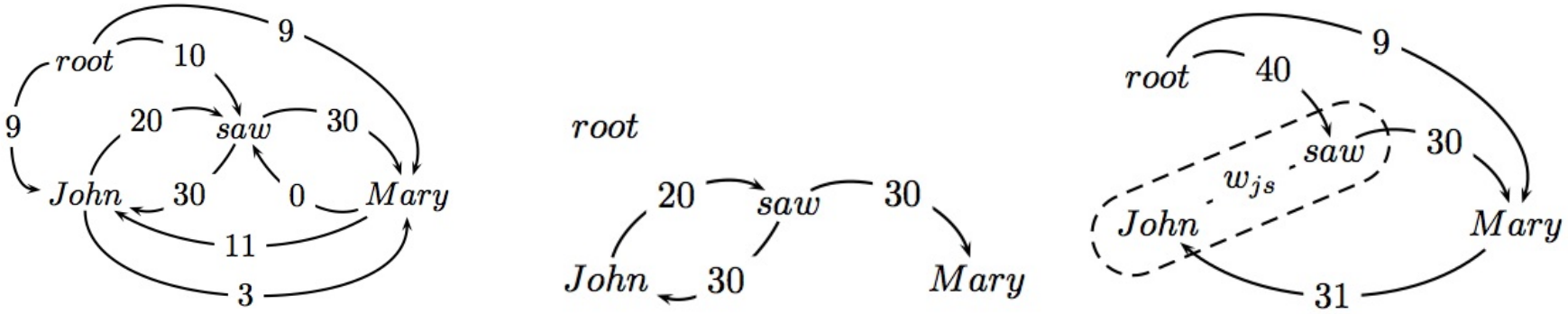
- If this is a tree, then we have found MST!!

Chu-Liu-Edmonds

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

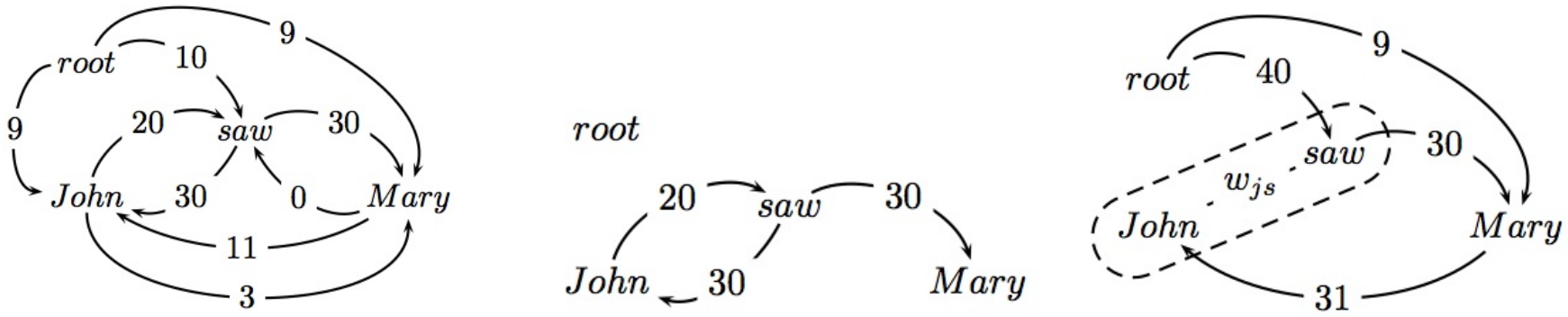


Chu-Liu-Edmonds



- 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

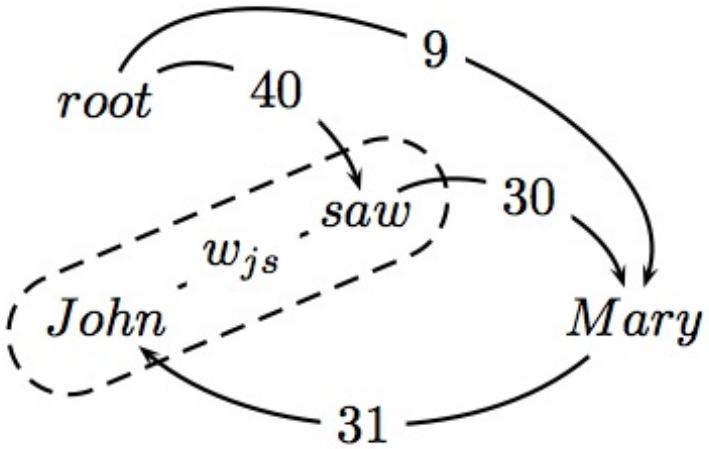


- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29

Chu-Liu-Edmonds

Theorem

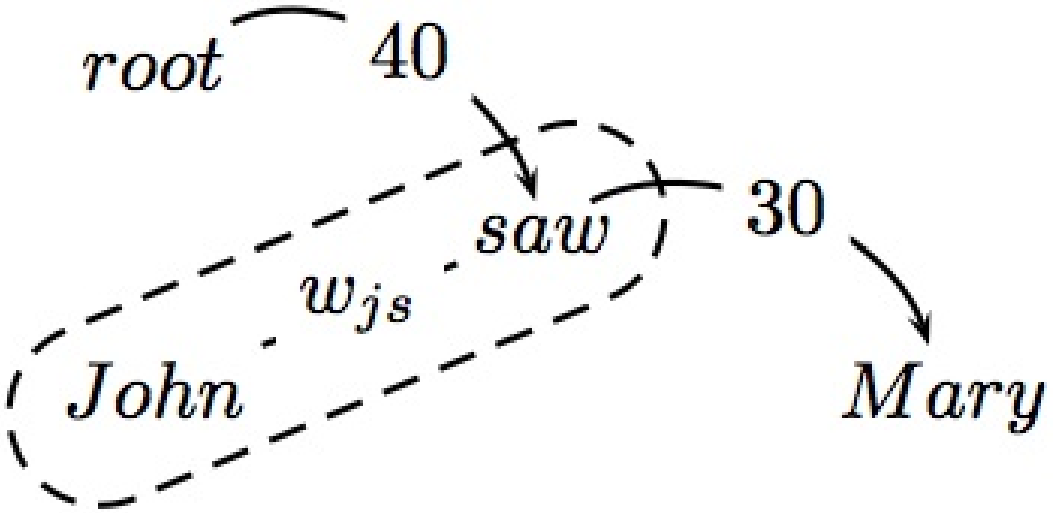
The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph



- Therefore, recursively call algorithm on new graph

Chu-Liu-Edmonds

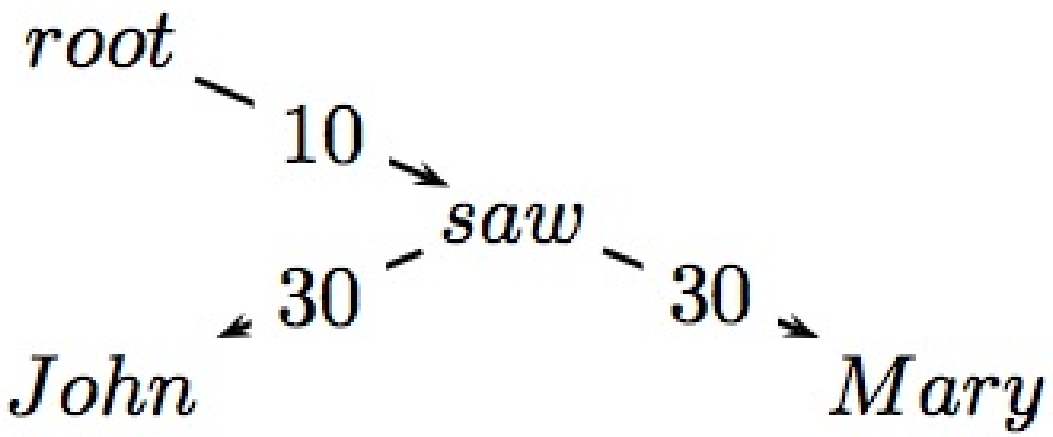
- This is a tree and the MST for the contracted graph!!



- Go back up recursive call and reconstruct final graph

Chu-Liu-Edmonds

- This is the MST!!



Chu-Liu-Edmonds Code

Chu-Liu-Edmonds(G_x, w)

1. Let $M = \{(i^*, j) : j \in V_x, i^* = \arg\max_{i'} w_{ij}\}$
2. Let $G_M = (V_x, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $< G_C, c, ma >=$ contract(G, C, w)
6. Let $G =$ Chu-Liu-Edmonds(G_C, w)
7. Find vertex $i \in C$ such that $(i', c) \in G$ and $ma(i', c) = i$
8. Find arc $(i'', i) \in C$
9. Find all arc $(c, i''') \in G$
10. $G = G \cup \{(ma(c, i'''), i''')\}_{\forall (c, i''') \in G} \cup C \cup \{(i', i)\} - \{(i'', i)\}$
11. Remove all vertices and arcs in G containing c
12. return G

- Reminder: $w_{ij} = \arg\max_k w_{ij}^k$

Chu-Liu-Edmonds Code (II)

contract$(G = (V, A), C, w)$

1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $i \in V - C : \exists_{i' \in C}(i', i) \in A$
   Add arc (c, i) to G_C with
   $ma(c, i) = \arg\max_{i' \in C} score(i', i)$
   $i' = ma(c, i)$
   $score(c, i) = score(i', i)$
4. For $i \in V - C : \exists_{i' \in C}(i, i') \in A$
   Add edge (i, c) to G_C with
   $ma(i, c) = \arg\max_{i' \in C} [score(i, i') - score(a(i'), i')]$
   $i' = ma(i, c)$
   $score(i, c) = [score(i, i') - score(a(i'), i') + score(C)]$
   where $a(v)$ is the predecessor of v in C
   and $score(C) = \sum_{v \in C} score(a(v), v)$
5. return $< G_C, c, ma >$

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 $O(|L|n^2)$ [Tarjan 1977]
- 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?**