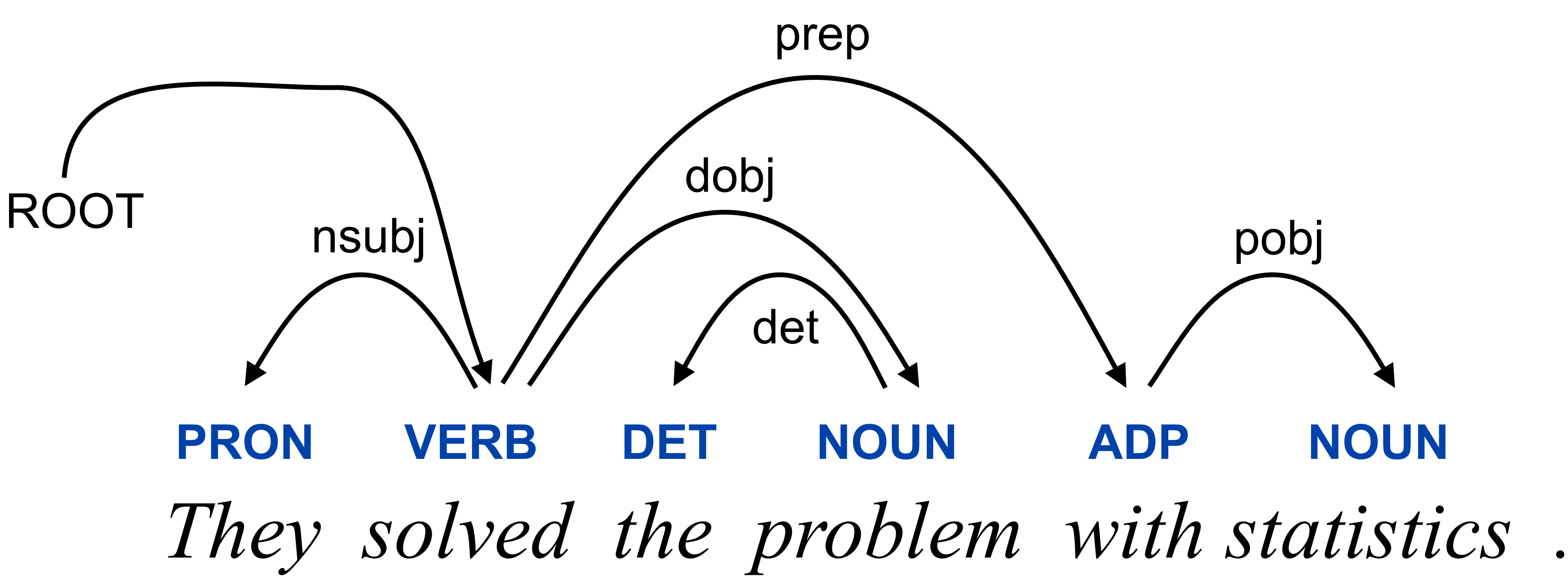
Syntax and Parsing

Slav Petrov – Google Research

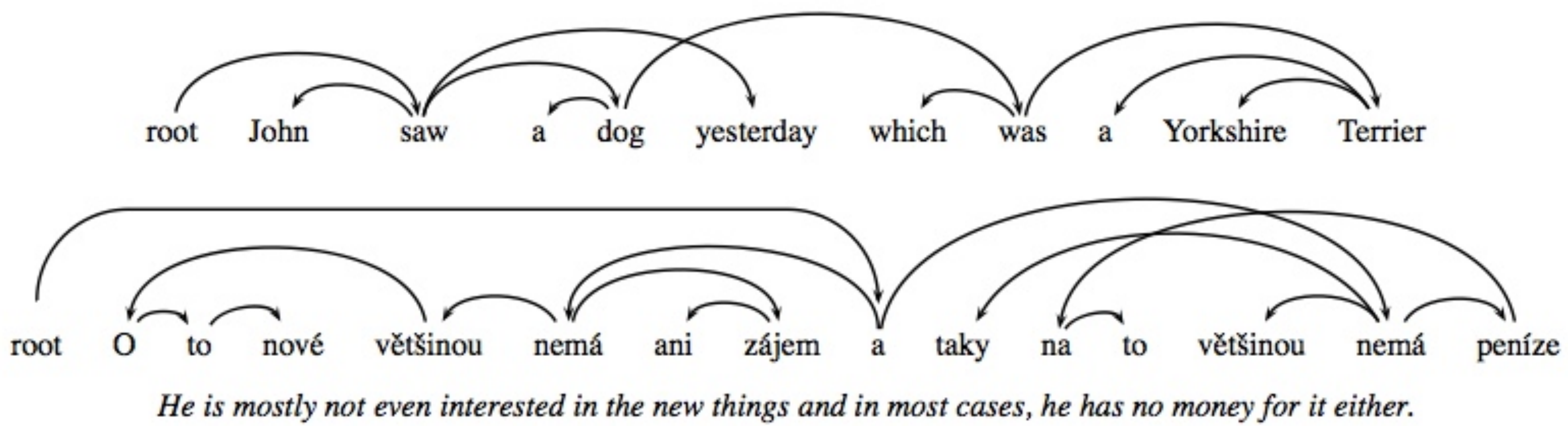
Lisbon Machine Learning School

Dependency Parsing



(Non-)Projectivity

- Crossing Arcs needed to account for non-projective constructions
- Fairly rare in English but can be common in other languages (e.g. Czech):



Formal Conditions

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$
- G is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.
- G is acyclic:
  - If $i \rightarrow j$, then not $j \rightarrow^* i$.
- G obeys the single-head constraint:
  - If $i \rightarrow j$, then not $i' \rightarrow j$, for any $i' \neq i$.
- G is projective:
  - If $i \rightarrow j$, then $i \rightarrow^* i'$, for any i' such that $i < i' < j$ or $j < i' < i$.

Arc-Factored Models

- Assumes that the score / probability / **weight** of a dependency graph factors by its arcs

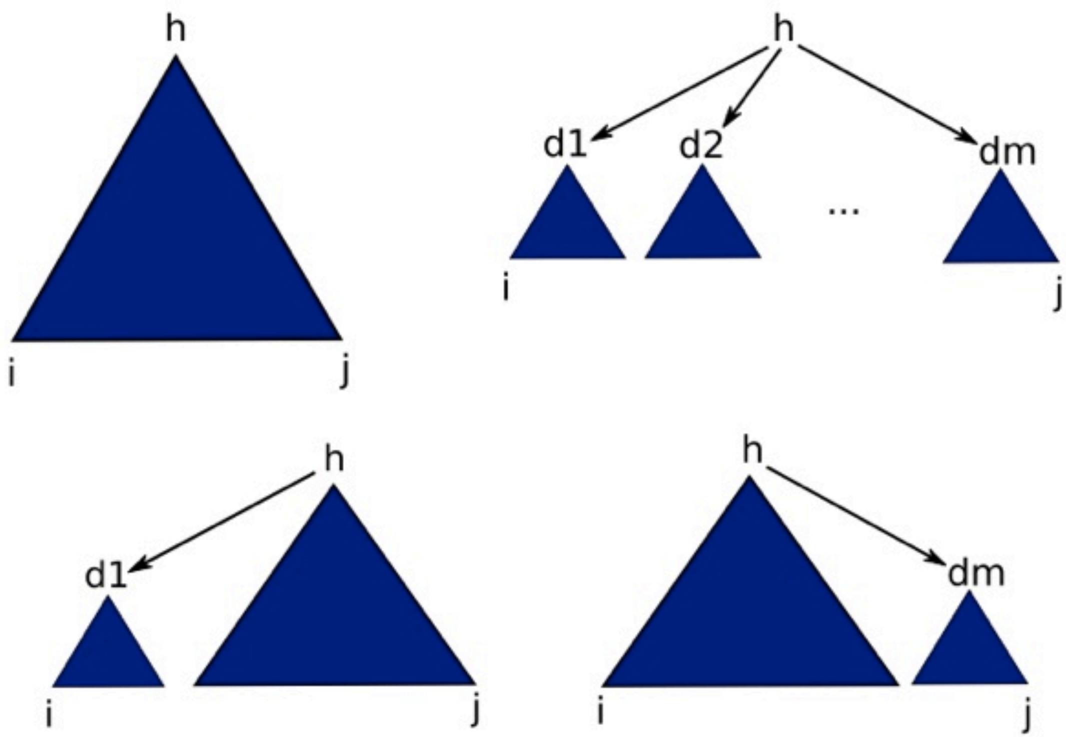
$$w(G) = \prod_{(i,j,k)\in G} w_{ij}^{k}$$

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.

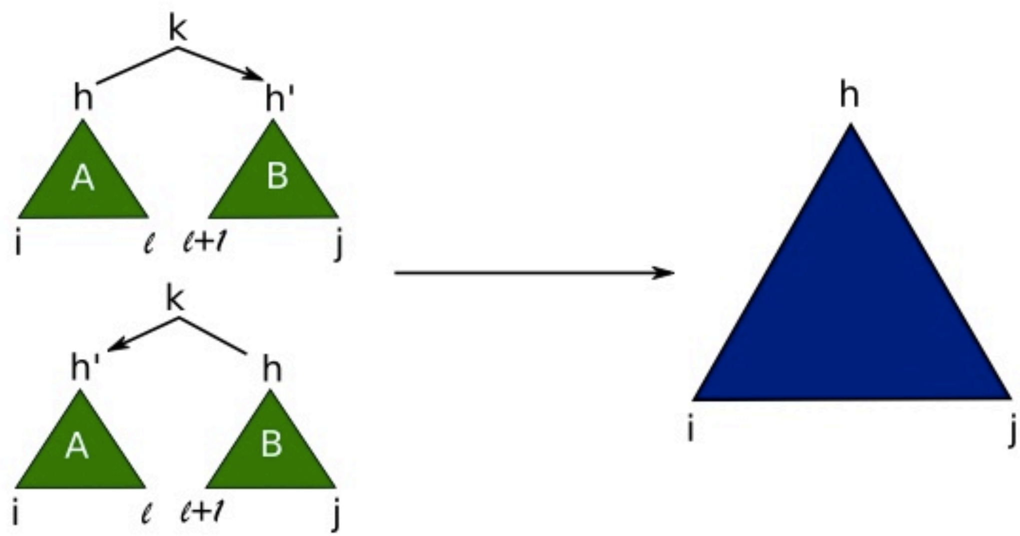
Arc-factored Projective Parsing

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



Arc-factored Projective Parsing

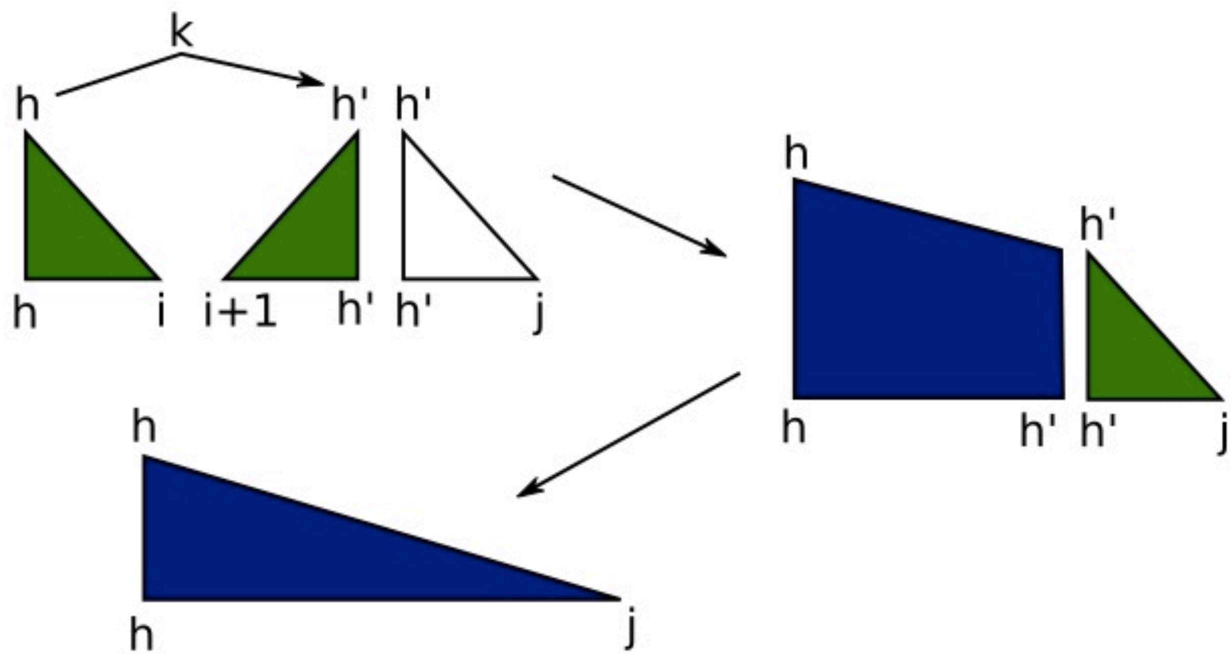
- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY



- 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)

Eisner Algorithm

- $O(|L|n^5)$ is not that good
- [Eisner 1996] showed how this can be reduced to $O(|L|n^3)$
  - Key: split items so that sub-roots are always on periphery



Eisner Algorithm PseudoCode

Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$

for $k : 1..n$

  for $s : 1..n$

    $t = s + k$

    if $t > n$ then break

    % First: create incomplete items

    $C[s][t][\leftarrow][0] = \max_{s \leq r < t} \ (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s))$

    $C[s][t][\rightarrow][0] = \max_{s \leq r < t} \ (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$

    % Second: create complete items

    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} \ (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$

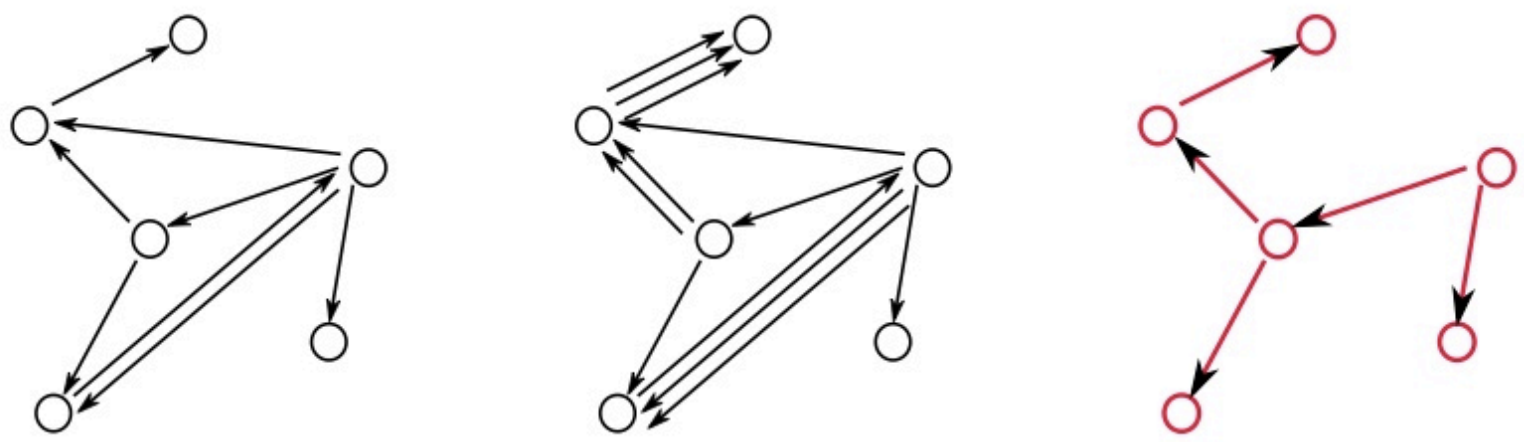
    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} \ (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

  end for

end for

Maximum Spanning Trees (MSTs)

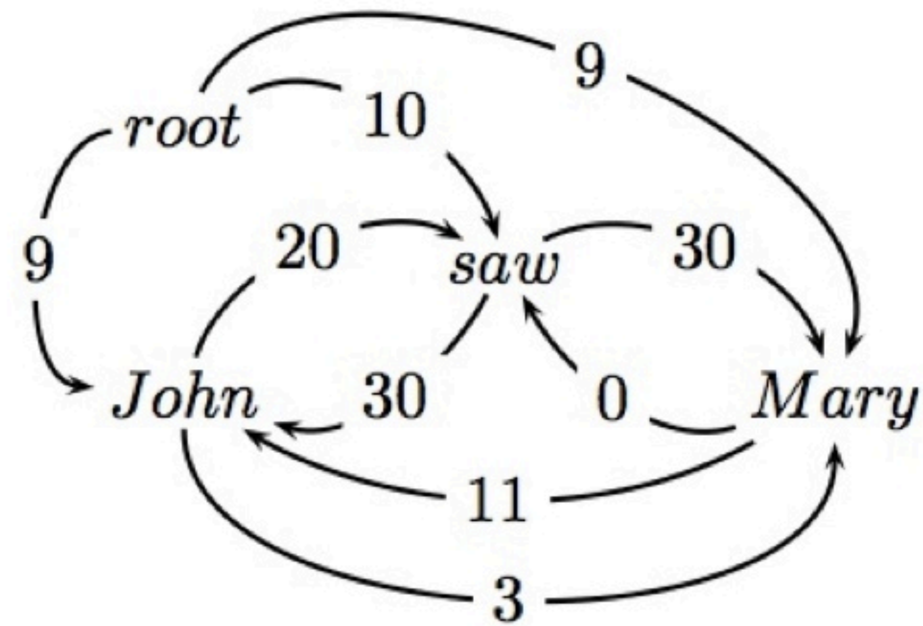
- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - G' is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs



Can use MST algorithms for nonprojective parsing!

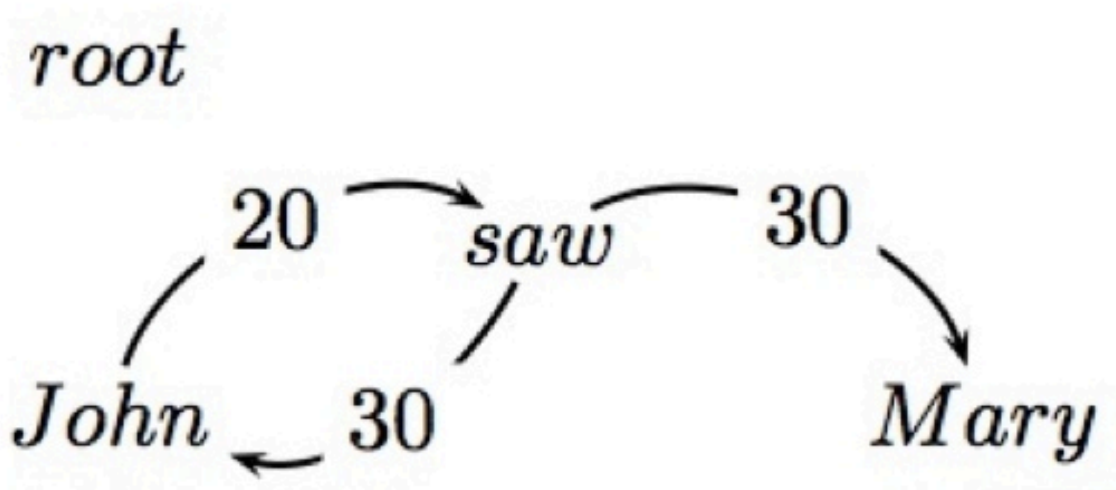
Chu-Liu-Edmonds

- x = root John saw Mary



Chu-Liu-Edmonds

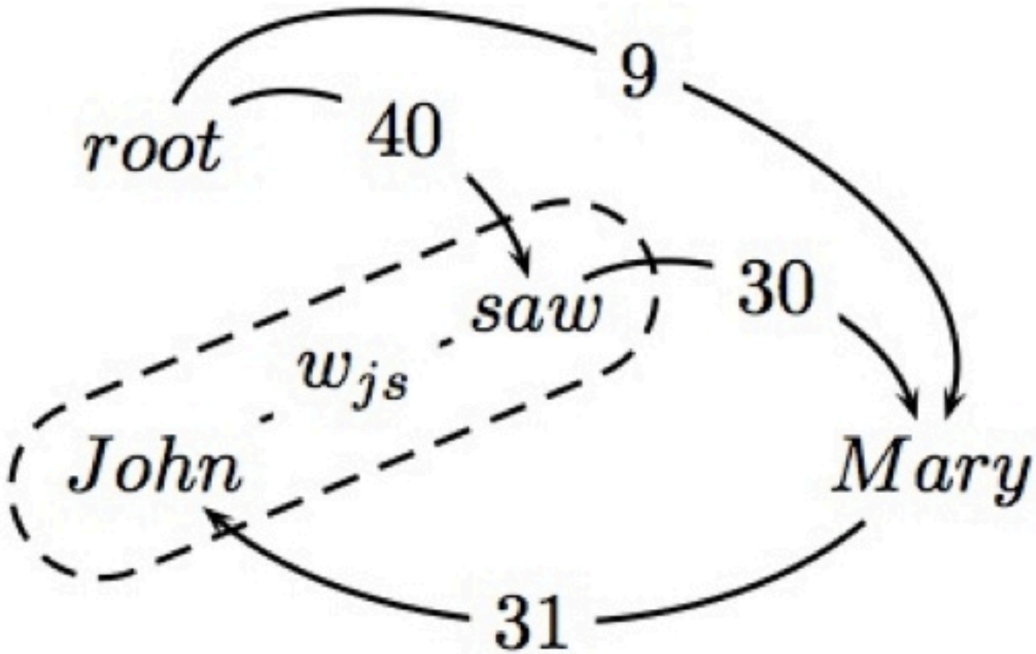
- Find highest scoring incoming arc for each vertex



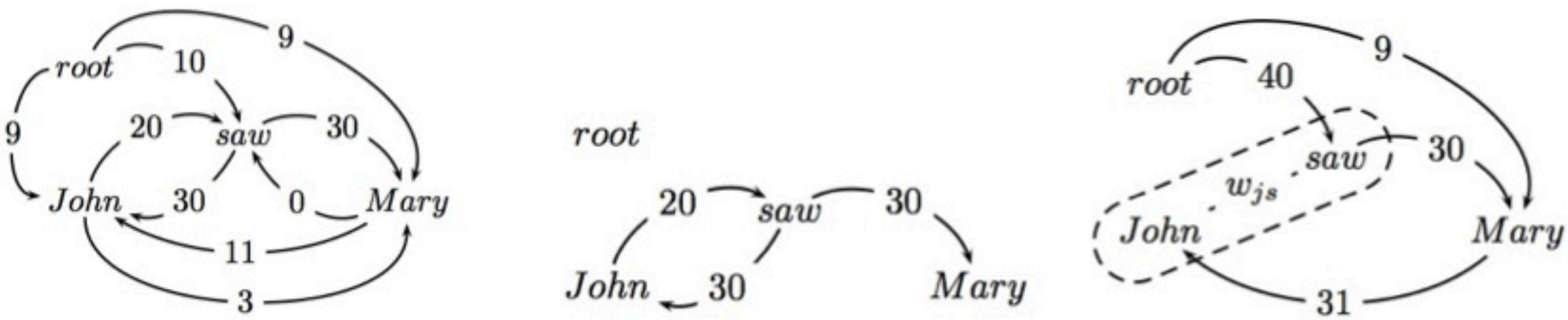
- If this is a tree, then we have found MST!!

Find Cycle and Contract

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



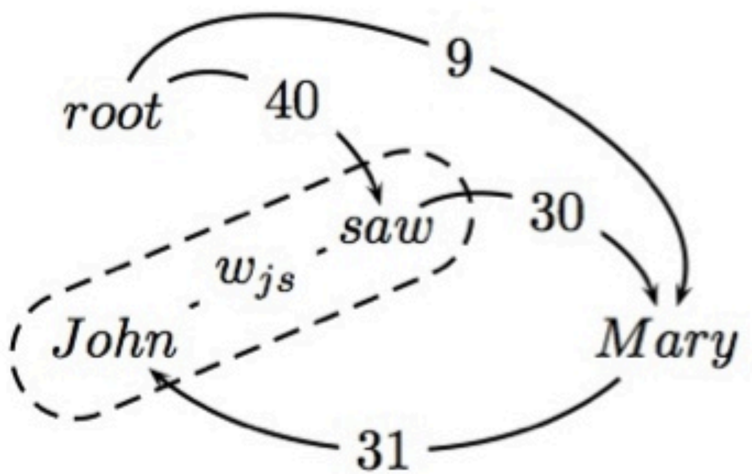
Recalculate Edge Weights



- 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

Theorem

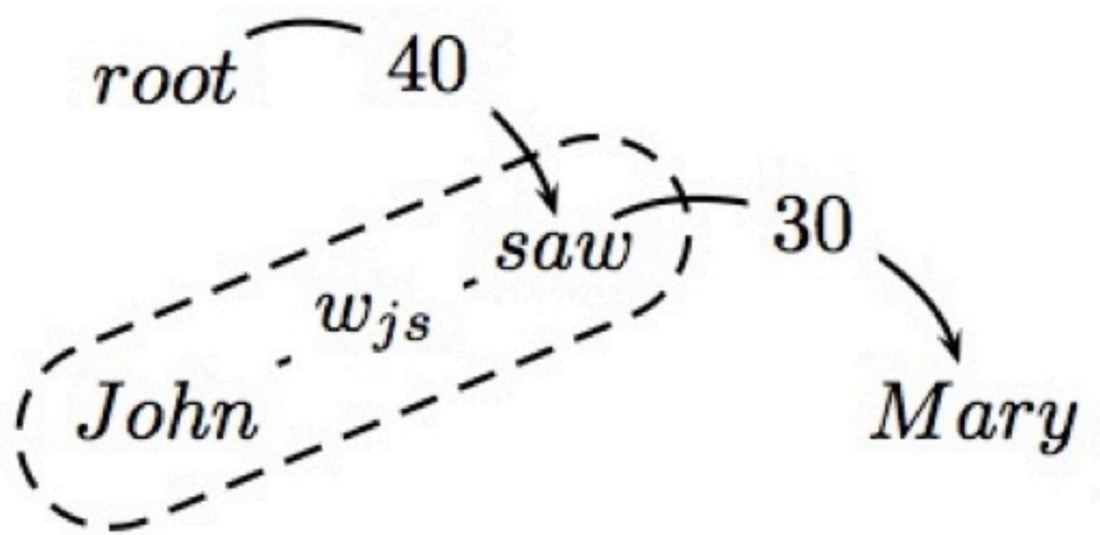
The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph



- Therefore, recursively call algorithm on new graph

Final MST

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



- Go back up recursive call and reconstruct final graph

Chu-Liu-Edmonds PseudoCode

Chu-Liu-Edmonds(G_x, w)

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

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

Chu-Liu-Edmonds PseudoCode

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

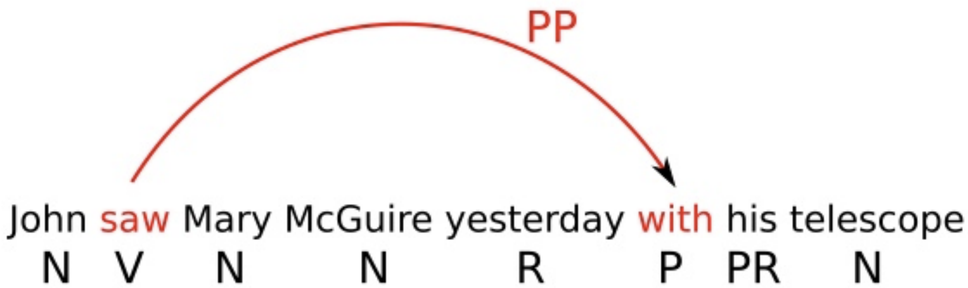
1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $i \in V - C : \exists_{i' \in C}(i', i) \in A$
   - Add arc (c, i) to G_C with
     - $ma(c, i) = \arg\max_{i' \in C} score(i', i)$
     - $i' = ma(c, i)$
     - $score(c, i) = score(i', i)$
4. For $i \in V - C : \exists_{i' \in C}(i, i') \in A$
   - 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 $< G_C, c, ma >$

Arc Weights

$$w_{ij}^{k} = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

- Arc weights are a linear combination of features of the arc, **f**, and a corresponding weight vector **w**
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features

Arc Feature Ideas for f(i,j,k)



- Identities of the words wi and wj and the label lk
- Part-of-speech tags of the words wi and wj and the label lk
- Part-of-speech of words surrounding and between wi and wj
- Number of words between wi and wj , and their orientation
- Combinations of the above

(Structured) Perceptron

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

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

Partition Function

Partition Function: $Z_x = \sum_{G \in T(G_x)} w(G)$

- Lapacian Matrix Q for graph $G_x = (V_x, A_x)$

$$Q_{jj} = \sum_{i \neq j, (i,j,k) \in A_x} w_{ij}^k \quad \text{and} \quad Q_{ij} = \sum_{i \neq j, (i,j,k) \in A_x} -w_{ij}^k$$

- Cofactor Q^i is the matrix Q with the i^{th} row and column removed

The Matrix Tree Theorem [Tutte 1984]
The determinant of the cofactor Q^0 is equal to Z_x

- Thus $Z_x = |Q^0|$ – determinants can be calculated in $O(n^3)$
- Constructing Q takes $O(|L|n^2)$
- Therefore the whole process takes $O(n^3 + |L|n^2)$

Arc Expectations

$$\langle i, j, k \rangle_x = \sum_{G \in T(G_x)} w(G) \times \mathbb{1}[(i, j, k) \in A]$$

- Can easily be calculated, first reset some weights

$$w_{i'j}^{k'} = 0 \; \forall i' \neq i \text{ and } k' \neq k$$

- Now, $\langle i, j, k \rangle_x = Z_x$
- Why? All competing arc weights to zero, therefore every non-zero weighted graph must contain (i, j, k)
- Naively takes $O(n^5 + |L|n^2)$ to compute all expectations
- But can be calculated in $O(n^3 + |L|n^2)$ (see [McDonald and Satta 2007, Smith and Smith 2007, Koo et al. 2007])

Summary

- Constituency Parsing
  - CKY Algorithm
  - Lexicalized Grammars
  - Latent Variable Grammars

- Dependency Parsing
  - Eisner Algorithm
  - Maximum Spanning Tree Algorithm

There is lots more and these models are being actively used in practice!