Dependency Parsing (3)

CMSC 470

Marine Carpuat

Fig credits: Joakim Nivre, Dan Jurafsky & James Martin

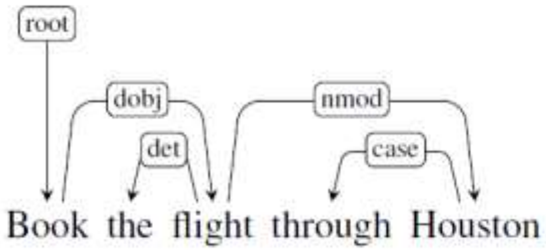
Dependency Parsing: what you should know

- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition systems: arc standard, arc eager
  - Oracle algorithm: how to obtain a transition sequence given a tree
  - How to construct a multiclass classifier to predict parsing actions
  - What transition-based parsers can and cannot do
  - That transition-based parsers provide a flexible framework that allows many extensions
    - such as RNNs vs feature engineering, non-projectivity (but I don’t expect you to memorize these algorithms)

- Next: Graph-based dependency parsing

Generating Training Examples

- What we have in a treebank



- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 14.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

Generating training examples

- Approach: simulate parsing to generate reference tree
- Given
  - A current config with stack S, dependency relations Rc
  - A reference parse (V,Rp)
- Do

LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$

RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t. (S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$

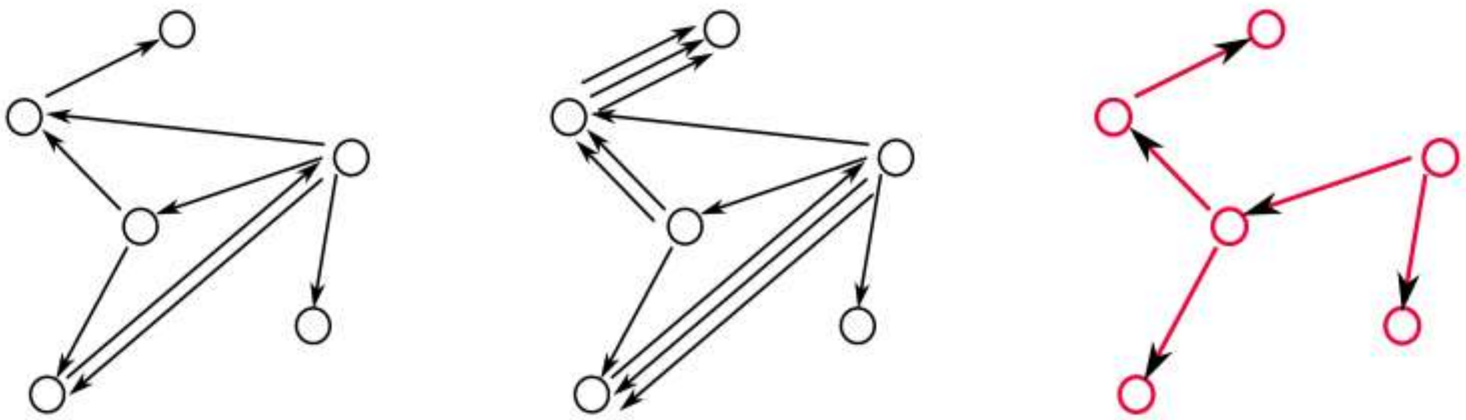
SHIFT: **otherwise**

Additional condition on RightArc makes sure a word is not removed from stack before its been attached to all its dependent

Graph-based Dependency Parsing

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



Dependency Parsing as Finding the Maximum Spanning Tree

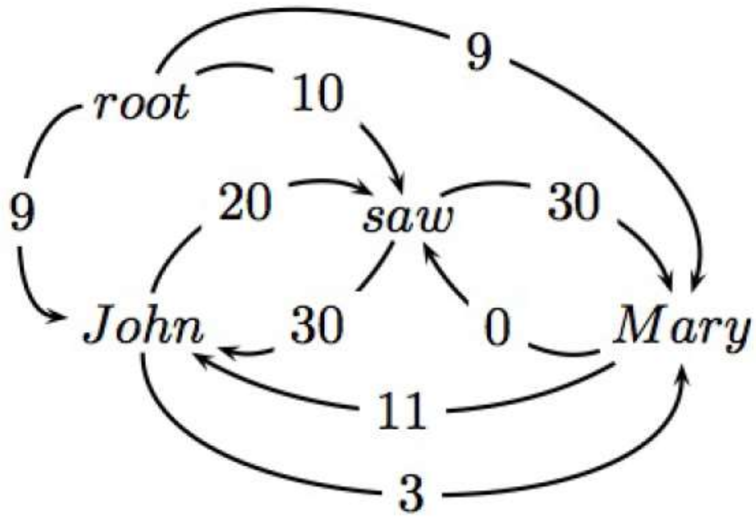
- Views parsing as finding the best directed spanning tree
  - of multi-digraph that captures all possible dependencies in a sentence
  - needs a score that quantifies how good a tree is

- Assume we have an **arc factored** model

  i.e. weight of graph can be factored as sum or product of weights of its arcs

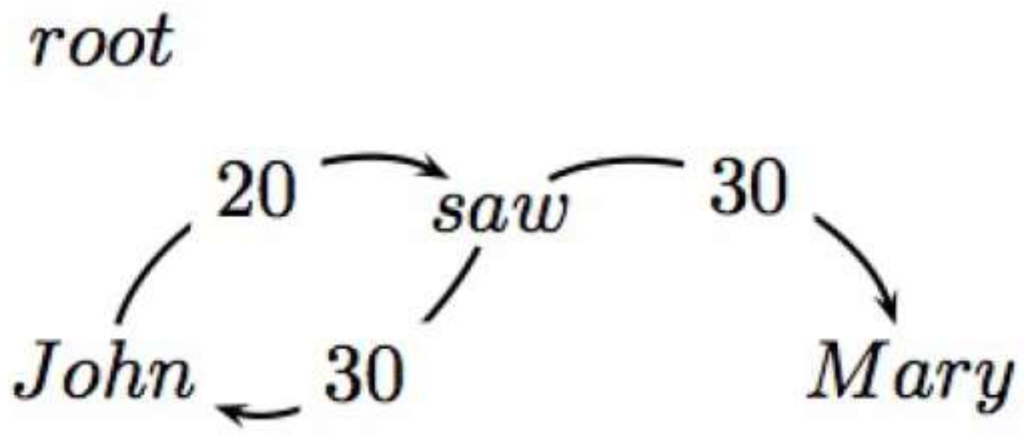
- Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us
  - Recursive algorithm
  - Naïve implementation: O(n^3)

Chu-Liu-Edmonds illustrated
(for unlabeled dependency parsing)



Chu-Liu-Edmonds illustrated

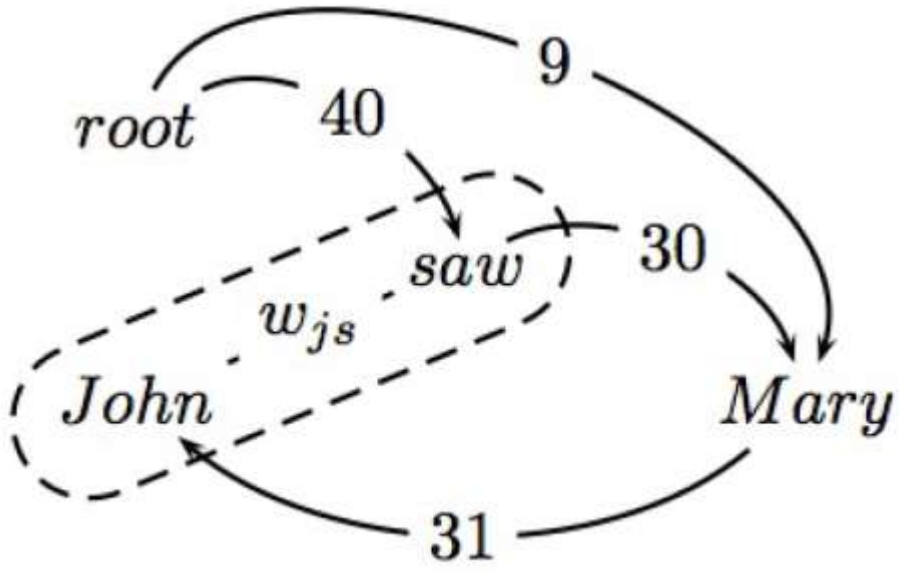
- Find highest scoring incoming arc for each vertex



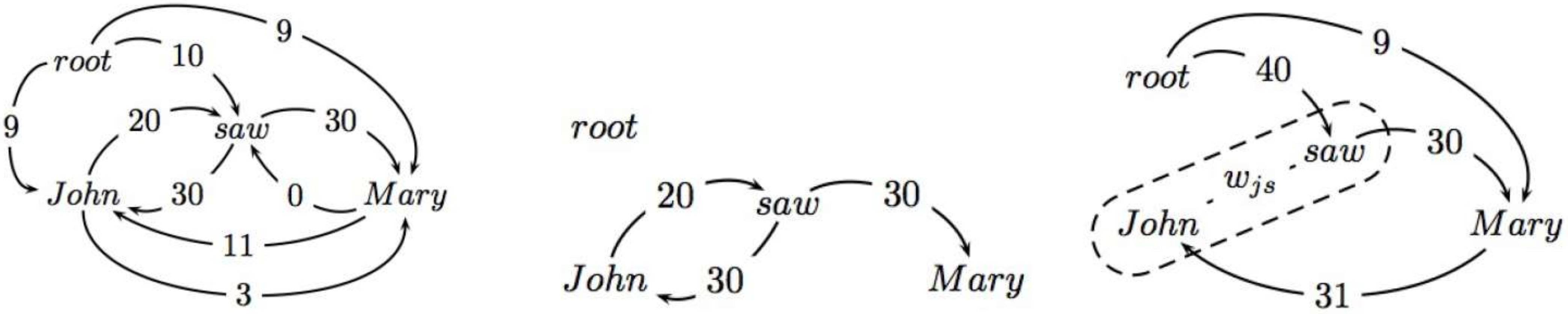
- If this is a tree, then we have found MST!!

Chu-Liu-Edmonds illustrated

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

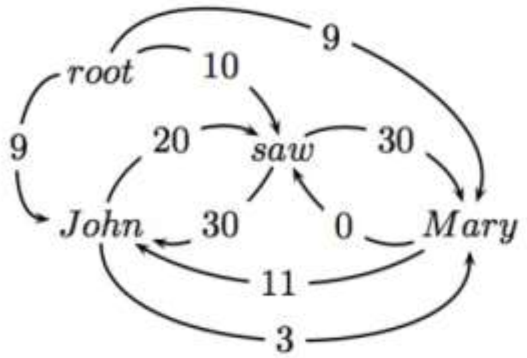


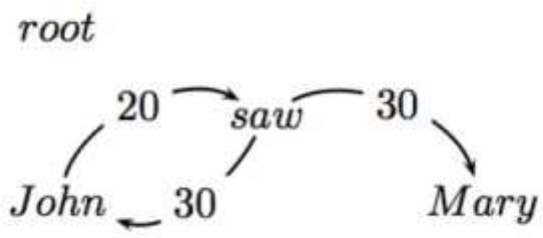
Chu-Liu-Edmonds illustrated

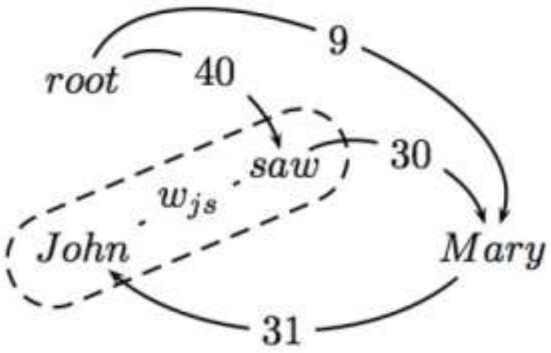


- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John $\rightarrow$ Mary is 3 and saw $\rightarrow$ Mary is 30

Chu-Liu-Edmonds illustrated

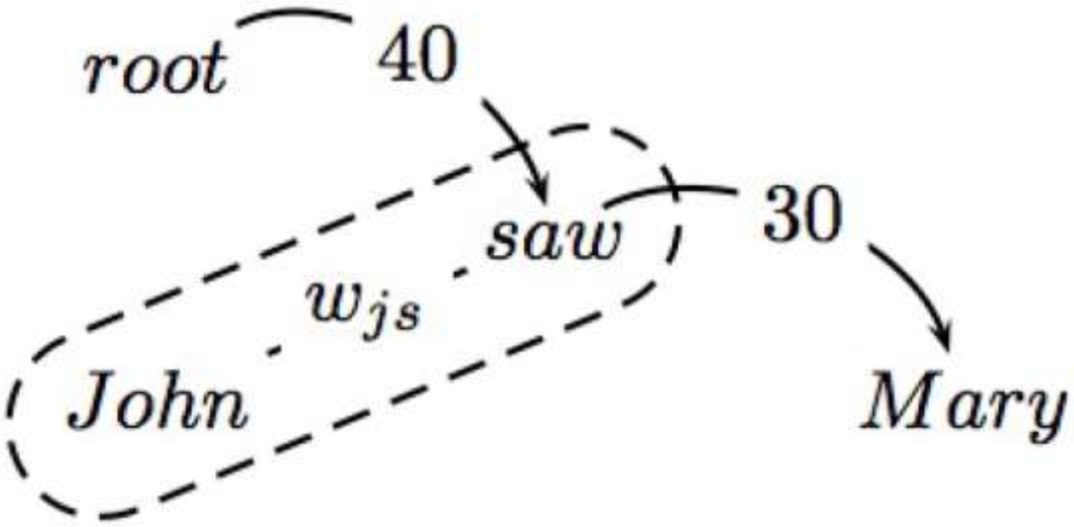






- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29

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



- Go back up recursive call and reconstruct final graph

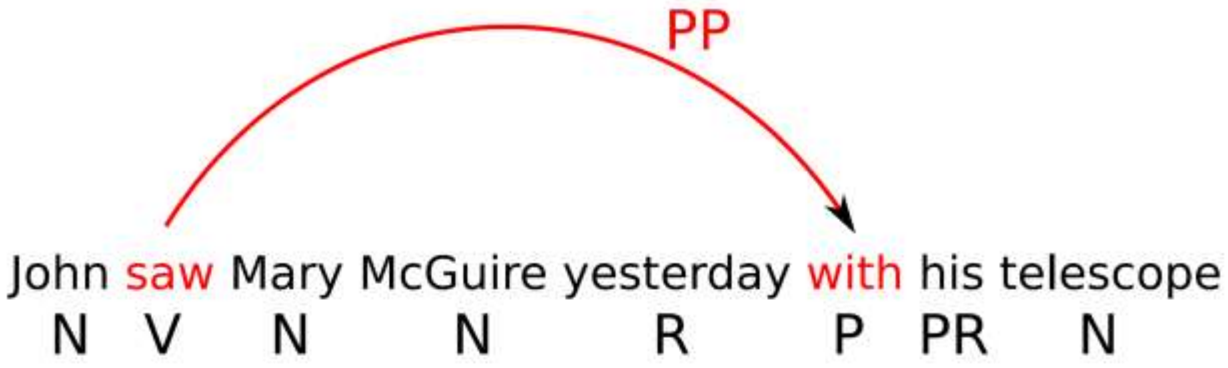
Arc weights as linear classifiers

Weight of arc from head **i** to dependent **j**, with label **k**

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

- Arc weights are a linear combination of features of the arc, $\mathbf{f}$, and a corresponding weight vector $\mathbf{w}$
- Raised to an exponent (simplifies some math ...)
- What arc features?

Example of classifier features



- Features from [McDonald et al. 2005]:
  - Identities of the words w_i and w_j and the label l_k

head=saw & dependent=with

Typical classifier features

- Word forms, lemmas, and parts of speech of the headword and its dependent
- Corresponding features derived from the contexts before, after and between the words
- Word embeddings
- The dependency relation itself
- The direction of the relation (to the right or left)
- The distance from the head to the dependent
- …

How to score a graph G using features?

Arc-factored model assumption

By definition of arc weights as linear classifiers

$$
\begin{aligned}
G &= \operatorname*{arg\,max}_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^{k} = \operatorname*{arg\,max}_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)} \\
&= \operatorname*{arg\,max}_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)} \\
&= \operatorname*{arg\,max}_{G \in T(G_x)} \sum_{(i,j,k) \in G} \mathbf{w} \cdot \mathbf{f}(i,j,k) \\
&= \operatorname*{arg\,max}_{G \in T(G_x)} \mathbf{w} \cdot \sum_{(i,j,k) \in G} \mathbf{f}(i,j,k) = \operatorname*{arg\,max}_{G \in T(G_x)} \mathbf{w} \cdot \mathbf{f}(G)
\end{aligned}
$$

Learning parameters with the Structured Perceptron

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

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

Dependency parsing algorithms

Transition-based

- Locally trained
- Use greedy search algorithms
- Define features over a rich history of parsing decisions

Graph-based

- Globally trained
- Use exact (or near exact) search algorithms
- Define features over a limited history of parsing decisions

Dependency Parsing: what you should know

- Interpreting dependency trees
- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition system: arc standard, arc eager
  - Oracle
  - Learning/predicting parsing actions
- Graph-based dependency parsing
- A flexible framework that allows many extensions
  - RNNs vs feature engineering, non-projectivity