Natural Language Understanding

Lecture 8: Introduction to Dependency Parsing

Frank Keller

School of Informatics
University of Edinburgh
keller@inf.ed.ac.uk

Based on slides by Mark Steedman

February 13, 2017

1. Dependency Grammar
   - Constituents vs. Dependencies
   - Dependency Relations
   - Dependency Trees
2. Dependency Parsing
   - Constituent vs. Dependency Parsing
   - Graph-based Dependency Parsing
   - Transition-based Dependency Parsing

Reading: McDonald et al. (2005).
Background: Jurafsky and Martin (2009: Ch. 12.7).

Constituents vs. Dependencies

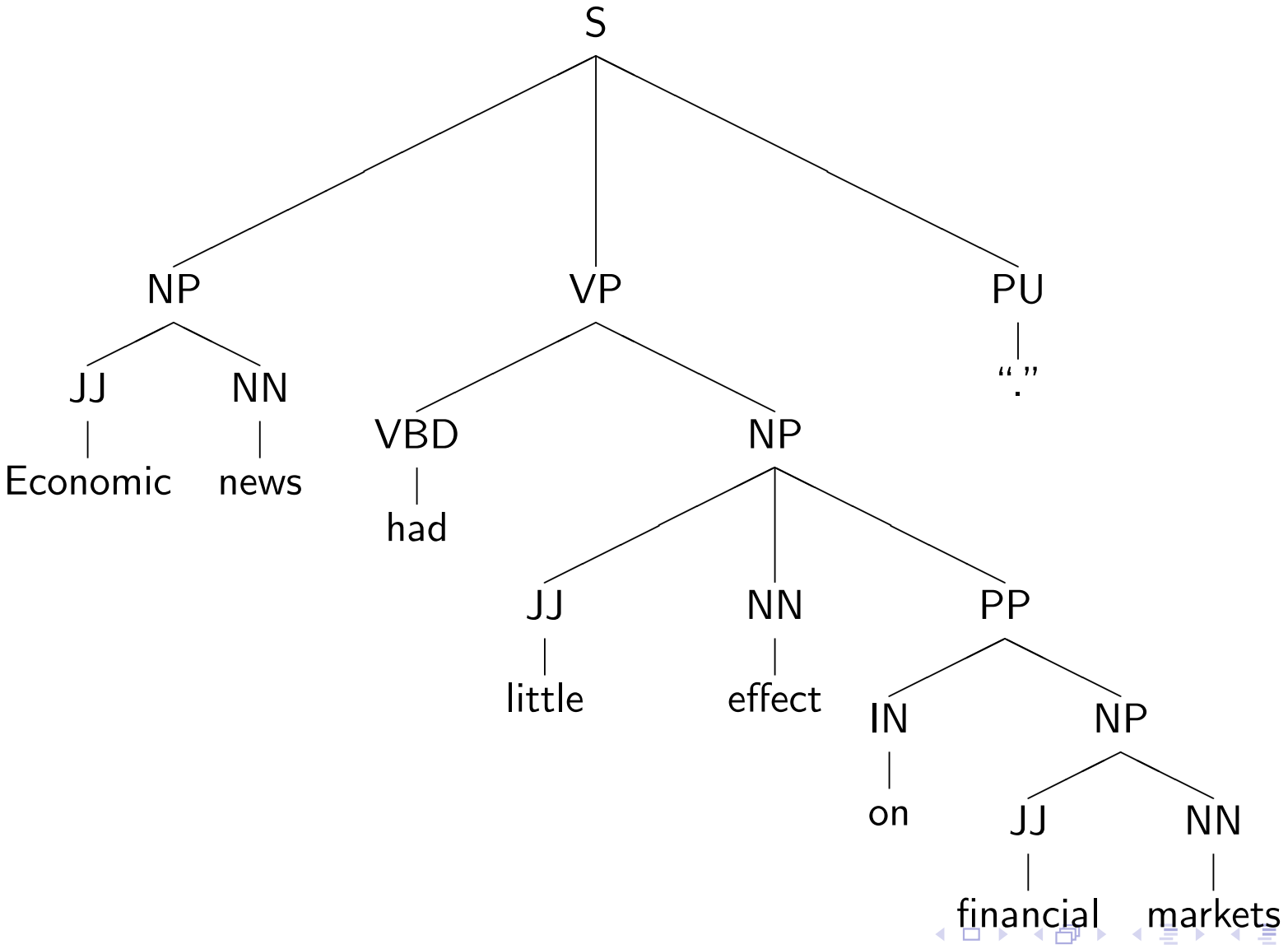
Traditional grammars model *constituent structure:* they capture the configurational patterns of sentences.

For example, verb phrases (VPs) have certain properties in English:

(1) a. I *like ice cream.* Do you ∅? (*VP ellipsis*)
b. I *like ice cream* and *hate bananas.* (*VP conjunction*)
c. I said I would hit Fred, and *hit Fred* I did. (*VP fronting*)

In other languages (e.g., German), there is little evidence for the existence of a VP constituent.

Constituents vs. Dependencies



Constituents vs. Dependencies

But from a semantic point of view, the important thing about verbs such as *like* is that they license two NPs:

1. an *agent,* found in *subject* position or with *nominative* inflection;
2. a *patient,* found in *object* position or with *accusative* inflection.

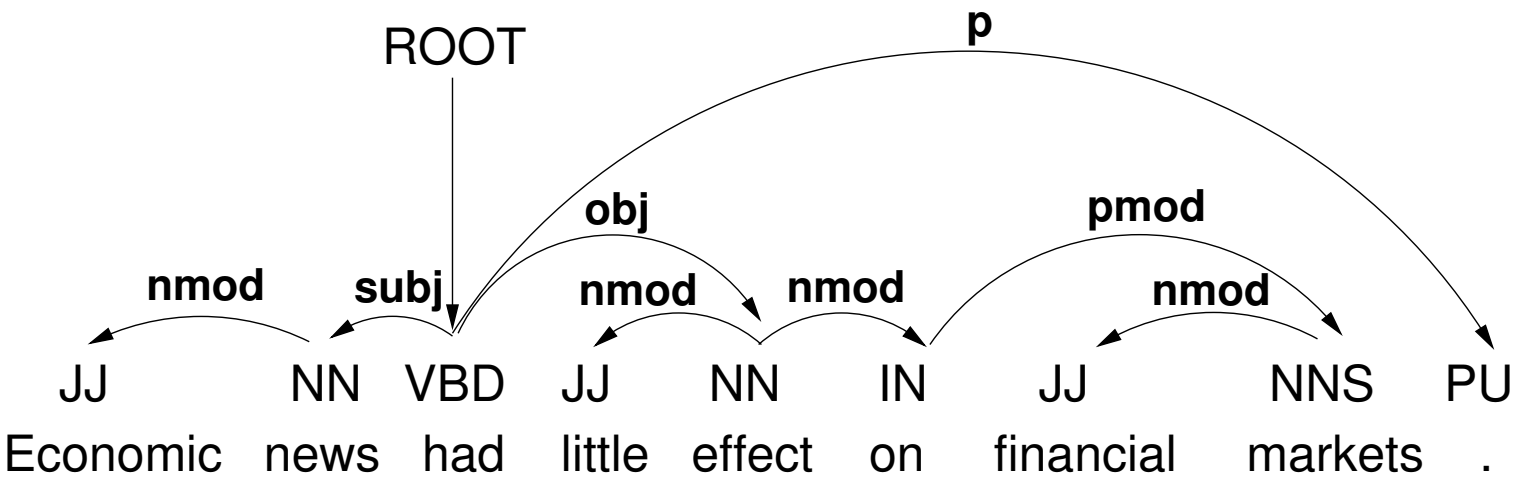
Which arguments are licensed, and which roles they play, depends on the verb (configuration is secondary).

To account for semantic patters, we focus *dependency.* Dependencies can be identified even in non-configurational languages.

Dependency Structure

A dependency structure consists of *dependency relations*, which are *binary* and *asymmetric*. A relation consists of:

- a *head* (H);
- a *dependent* (D);
- a *label* identifying the relation between H and D.



[From Joakim Nivre, Dependency Grammar and Dependency Parsing.]

Dependency Trees

Formally, the dependency structure of a sentence is a graph with the words of the sentence as its nodes, linked by directed, labeled edges, with the following properties:

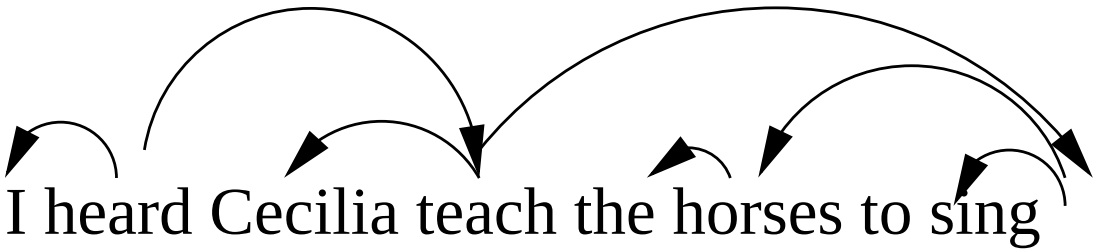
- *connected:* every node is related to at least one other node, and (through transitivity) to ROOT;
- *single headed:* every node (except ROOT) has exactly one incoming edge (from its head);
- *acyclic:* the graph cannot contain cycles of directed edges.

These conditions ensure that the dependency structure is a tree.

Dependency Trees: Projectivity

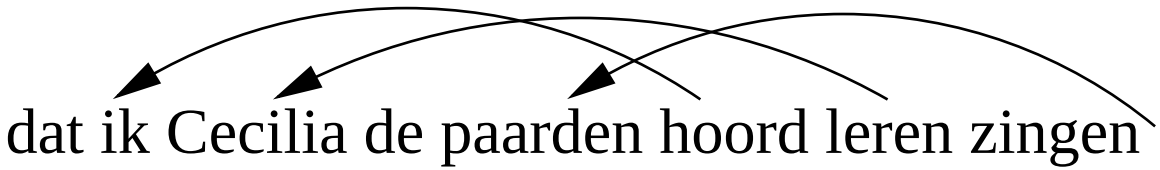
We distinguish projective and non-projective dependency trees:

A dependency tree is *projective* wrt. a particular linear order of its nodes if, for all edges $h \rightarrow d$ and nodes w, w occurs between h and d in linear order only if w is *dominated* by h.



Dependency Trees: Projectivity

A dependency tree is *non-projective* if w can occur between h and d in linear order without being dominated by h.



that I Cecilia the horses heard teach sing

"That I heard Cecilia teach the horses to sing"

A non-projective dependency grammar is not context-free. But efficient non-projective parsers exist, e.g., the MST parser.

Constituent vs. Dependency Parsing

In ANLP and FNLP, we've already seen various parsing algorithms for context-free languages (shift-reduce, CKY, active chart).

Why consider *dependency parsing* as a distinct topic?

- context-free parsing algorithms base their decisions on *adjacency;*
- in a dependency structure, a dependent need not be adjacent to its head (even if the structure is projective);
- we need new parsing algorithms to deal with non-adjacency (and with non-projectivity if present).

Two Types of Dependency Parsers

We will consider two types of dependency parsers:

1. graph-based dependency parsing, based on *maximum spanning trees* (MST parser, McDonald et al. 2005);
2. transition-based dependency parsing, an extension of *shift-reduce parsing* (MALT parser, Nivre 2003).

Alternative: map dependency trees to phrase structure trees and do standard CF parsing (not covered here).

Graph-based Dependency Parsing

Goal: find the highest scoring dependency tree in the space of all possible trees for a sentence.

Let $\mathbf{x} = x_1 \cdots x_n$ be the input sentence, and $\mathbf{y}$ a dependency tree for $\mathbf{x}$. Here, $\mathbf{y}$ is a set of dependency edges, with $(i, j) \in \mathbf{y}$ if there is an edge from x_i to x_j.

In order to assign scores to dependency trees, we *edge factorize* them: the score of a tree is the sum of the scores of all its edges.

Graph-based Dependency Parsing

The score of a dependency edge (i, j) is:

$$s(i, j) = \mathbf{w} \cdot \mathbf{f}(i, j)$$

where $\mathbf{w}$ is a weight vector and $\mathbf{f}(i, j)$ is a feature vector. Then the score of dependency tree $\mathbf{y}$ for sentence $\mathbf{x}$ is:

$$s(\mathbf{x}, \mathbf{y}) = \sum_{(i,j) \in \mathbf{y}} s(i, j) = \sum_{(i,j) \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(i, j)$$

Dependency parsing is the task of finding the tree $\mathbf{y}$ with highest score for a given sentence $\mathbf{x}$.

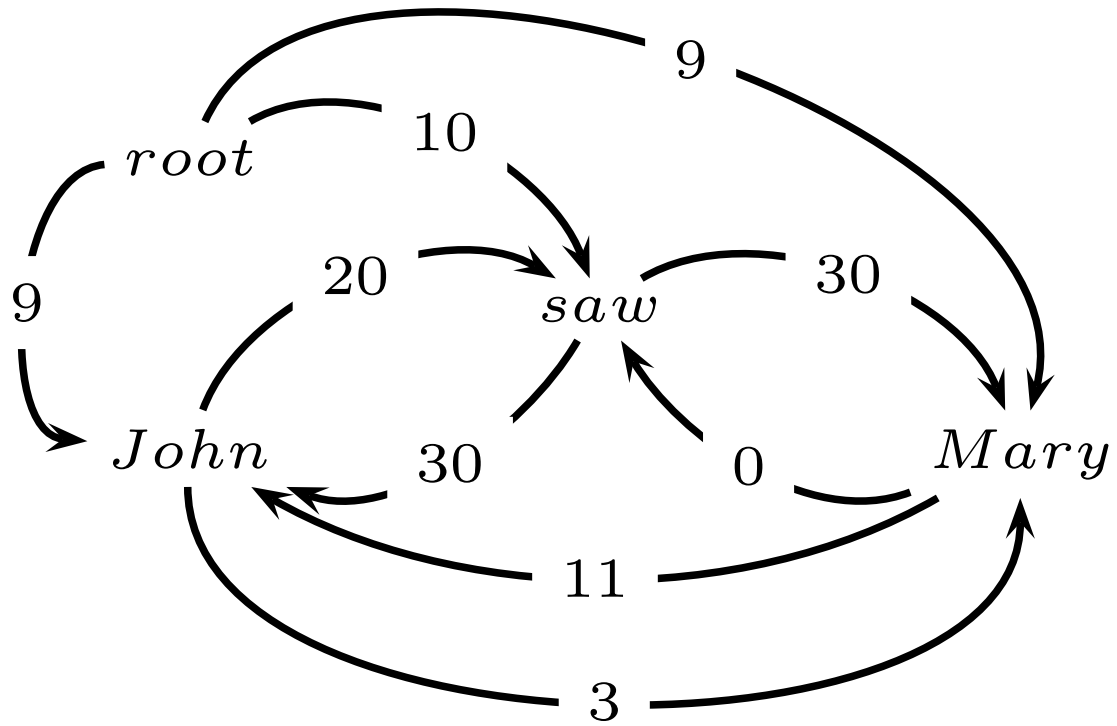
Maximum Spanning Tree Parsing

This task can be achieved using the following approach (McDonald et al. 2005):

- start with a *totally connected graph* G, i.e., assume a directed edge between every pair of words;
- assume you have a scoring function that assigns a score $s(i, j)$ to every edge (i, j);
- find the *maximum spanning tree* (MST) of G, i.e., the tree with the highest overall score that includes all nodes of G;
- this is possible in $O(n^2)$ time using the Chu-Liu-Edmonds algorithm; it finds both projective and non-projective MSTs;
- the correct parse is the MST of G.

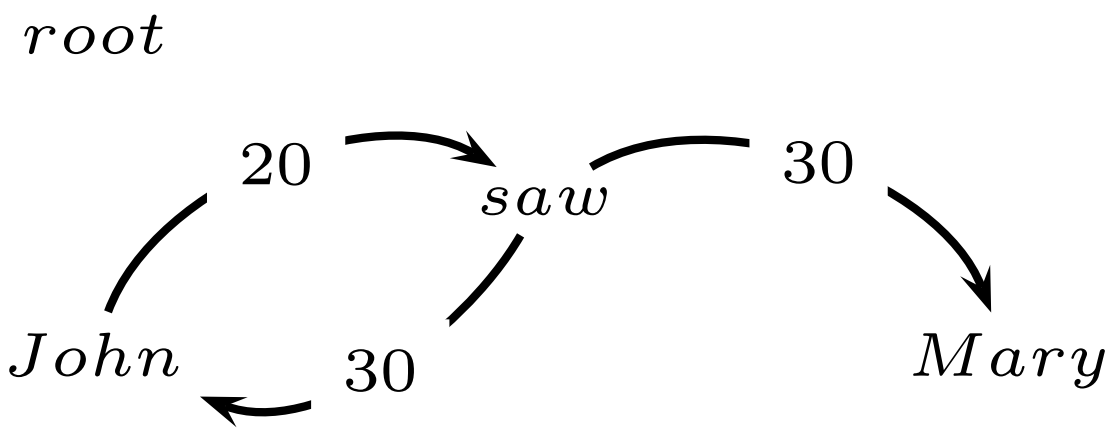
Chu-Liu-Edmonds (CLE) Algorithm

Example: $\mathbf{x} = \textit{John saw Mary}$, with graph $G_{\mathbf{x}}$. Start with the fully connected graph, with scores:



Chu-Liu-Edmonds (CLE) Algorithm

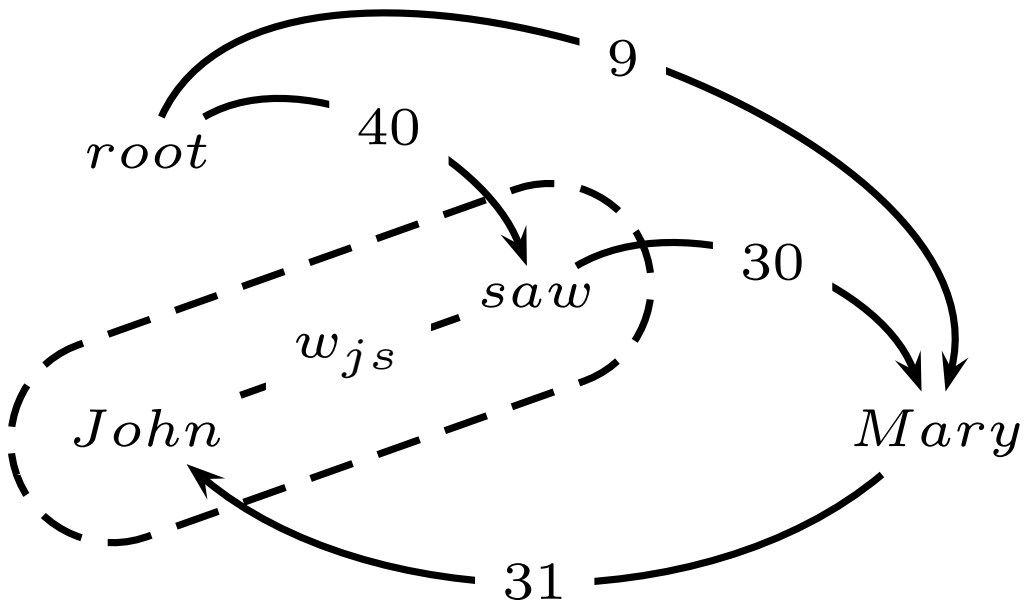
Each node j in the graph greedily selects the incoming edge with the highest score $s(i, j)$:



If a tree results, it is the maximum spanning tree. If not, there must be a cycle.

CLE Algorithm: Recursion

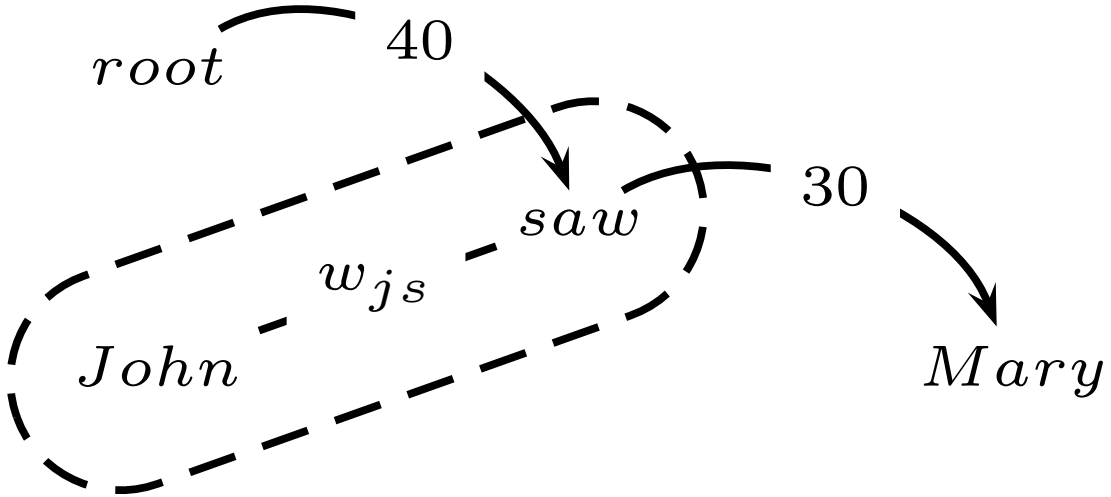
Identify the cycle and contract it into a single node and recalculate scores of incoming and outgoing edges:



Now call CLE recursively on this contracted graph. MST on the contracted graph is equivalent to MST on the original graph.

CLE Algorithm: Recursion

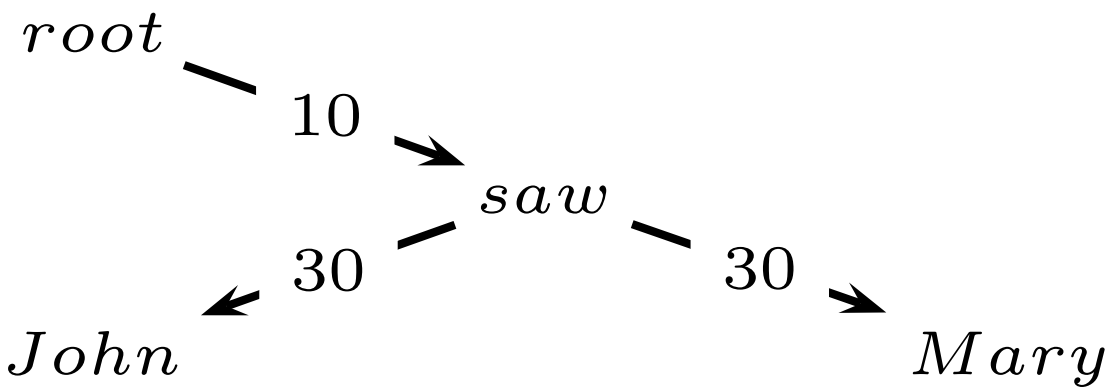
Again, greedily collect incoming edges to all nodes:



This is a tree, hence it must be the MST of the graph.

CLE Algorithm: Reconstruction

Now reconstruct the uncontracted graph: the edge from w_{js} to *Mary* was from *saw*. The edge from ROOT to w_{js} was a tree from ROOT to *saw* to *John*, so we include these edges too:



MST Parser: Features

Parsing accuracy depends the scoring function, i.e., the features $\mathbf{f}(i, j)$ and the weight vector $\mathbf{w}$.

The Margin Infused Relaxed Algorithm (MIRA) can be used to learn $\mathbf{w}$ (McDonald et al. 2005).

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

Unigram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

Bigram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

MST Parser: Features

More features:

In Between PoS Features
p-pos, b-pos, c-pos
Surrounding Word PoS Features
p-pos, p-pos+1, c-pos−1, c-pos
p-pos−1, p-pos, c-pos−1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos−1, p-pos, c-pos, c-pos+1

p: parent, c: child, b: between parent and child, -1: to left, $+1$: to right.

Total feature: 6,998,447 for English 13,450,672 for Czech.

Transition-based Dependency Parsing

The MST parser builds a dependency tree though graph surgery.
The main alternative is the MALT parser (Nivre 2003):

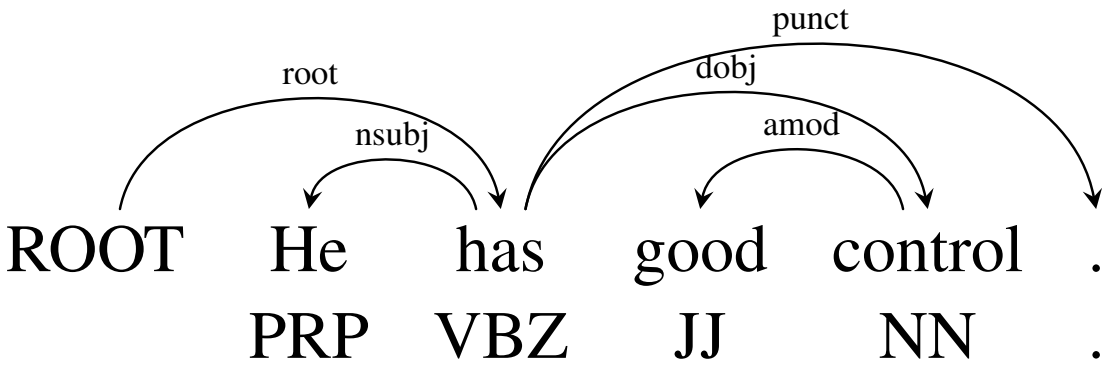
- the MALT parser is a *shift-reduce parser,* defined through a transition system;
- for a given parse state, the transition system defines a set of actions $\mathcal{T}$ which the parser can take;
- if more than one action is applicable, a classifier (e.g., an SVM) is used to decide which action to take;
- just like in the MST model, this requires a large set of hand-engineered features.

Transition-based Dependency Parsing

The arc-standard transition system (Nivre 2003):

- configuration $c = (s, b, A)$ with stack s, buffer b, set of dependency arcs A;
- initial configuration for sentence $w_1, \ldots, w_n$ is $s = [\texttt{ROOT}], b = [w_1, \ldots, w_n], A = \emptyset$;
- c is terminal if buffer is empty, stack contains only ROOT, and parse tree is given by A_c;
- if s_i is the ith top element on stack, and b_i the ith element on buffer, then we have the following transitions:
- LEFT-ARC(l): adds arc $s_1 \rightarrow s_2$ with label l and removes s_2 from stack; precondition: $|s| \geq 2$;
- RIGHT-ARC(l): adds arc $s_2 \rightarrow s_1$ with label l and removes s_1 from stack; precondition: $|s| \geq 2$;
- SHIFT: moves b_1 from buffer to stack; recondition: $|b| \geq 1$.

Transition-based Dependency Parsing



Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	$\emptyset$
SHIFT	[ROOT He]	[has good control .]	
SHIFT	[ROOT He has]	[good control .]	
LEFT-ARC(nsubj)	[ROOT has]	[good control .]	$A \cup$ nsubj(has,He)
SHIFT	[ROOT has good]	[control .]	
SHIFT	[ROOT has good control]	[.]	
LEFT-ARC(amod)	[ROOT has control]	[.]	$A \cup$amod(control,good)
RIGHT-ARC(dobj)	[ROOT has]	[.]	$A \cup$ dobj(has,control)
...	...	...	...
RIGHT-ARC(root)	[ROOT]	[]	$A \cup$ root(ROOT,has)

Summary

Comparing the MST and MALT parsers:

- the MST parser selects the globally optimal tree, given a set of edges with scores;
- it can naturally handle projective and non-projective trees;
- the MALT parser makes a sequence of local decisions about the best parse action;
- it can be extended to "pseudo-projective" dependency trees with transformation techniques;
- accuracy of MST and MALT is similar, but MALT is faster;
- both require a set of manually engineered featured and a model that learns feature weights.

Recent work on dependency parsing uses *neural networks to learn both features and feature weights:* next lecture.

References

Jurafsky, Daniel, and James H. Martin. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Upper Saddle River, NJ: Pearson Education, 2nd edn.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing*, 523–530. Vancouver.

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the International Workshop on Parsing Technologies*, 149–160. Nancy.