
Advanced Natural Language Processing

Lecture 18

Dependency Parsing

Frank Keller (slides by Mark Steedman and Bonnie Webber)

2 November 2011



What's the Problem?

- Natural Languages are only mildly context sensitive, and most constructions can be handled as context-free.

“the overwhelming majority of syntactic structures are projective in most languages” Nivre (2010)

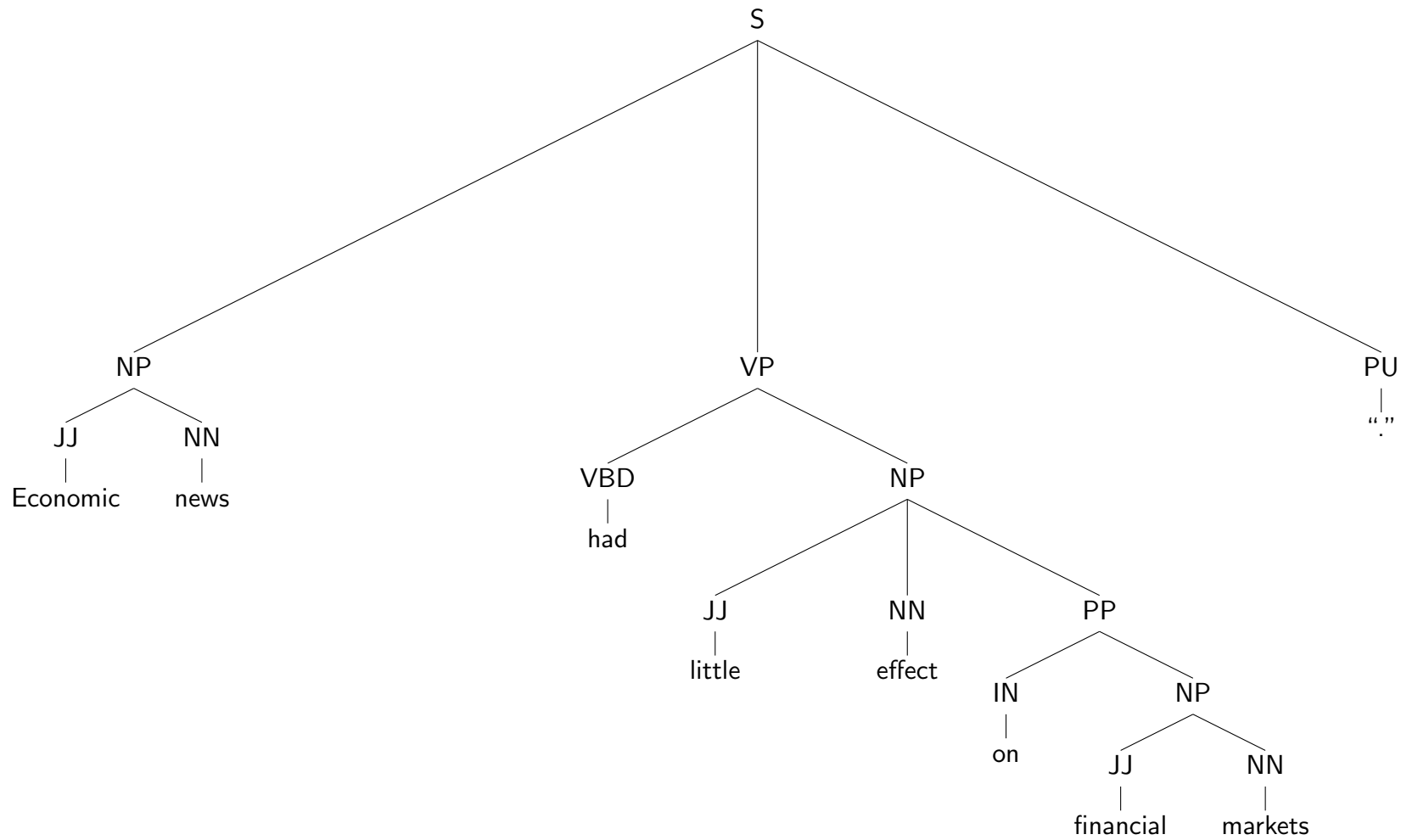
- We've already seen a number of parsing regimes for context-free languages (shift-reduce, CKY, active charts).
- Why consider dependency parsing as a distinct topic?

- All the parsing algorithms we've considered base their decisions on **adjacency**.
- Even with projective structures, a dependent need not be adjacent to its head.
- With dependency parsing, we'll consider
 - how CF parsing needs to be modified to handle non-adjacency;
 - how it needs to be extended slightly to handle non-projectivity.

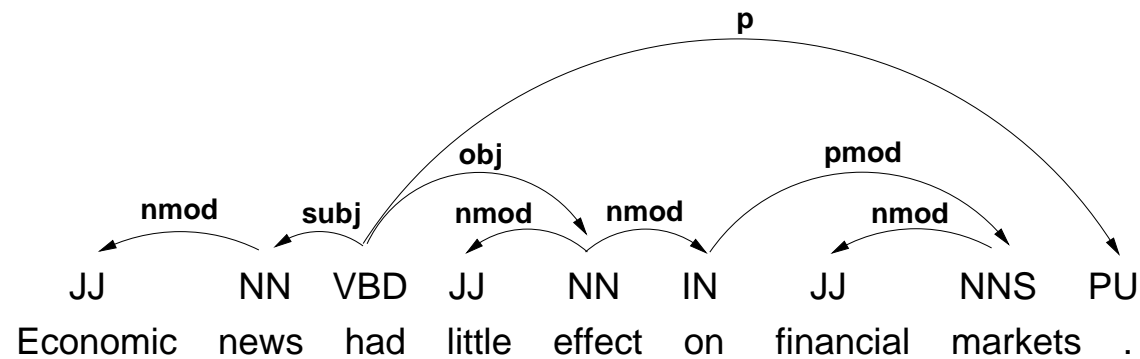
Three Types of Dependency Parsers Nivre (2010)

1. via a 1:1 mapping of dependency trees to phrase structure trees, then standard CF parsing (or a bespoke form, designed for *CF dependency grammars*);
2. graph-based dependency parsing, based on *maximum spanning trees* (MST parser)
3. transition-based dependency parsing, an extension of *shift-reduce parsing* (MALT parser).

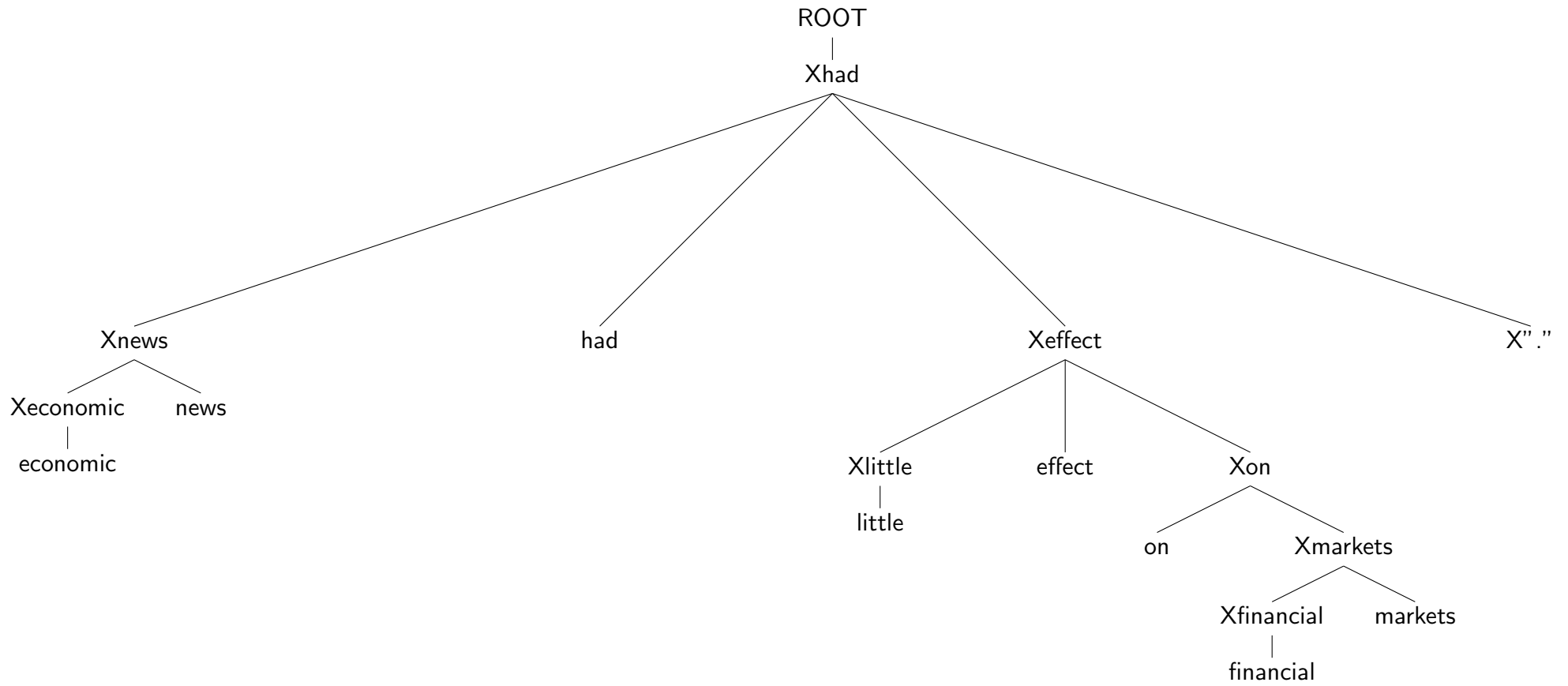
Start by looking at mapping dependency trees (Dep Trees) to phrase structure (PS) Trees. They will not be **standard** PS trees like:



Dep Tree to PS Tree: Standard Dep Tree



- Map the subtree rooted at w , with direct arcs to subtrees t_1, \dots, t_n
- to a subtree rooted at a node labelled X_w dominating a terminal node labelled w and subtrees corresponding to t_1, \dots, t_n **in the order given by the original word order.**



Dep Tree to PS Tree: CF Dependency Tree

If labelled dependencies, indicate this on the labels going to non-head nodes – eg,

Xnews → Xnews:subj

Xmarkets → Xmarkets:pmod

Xlittle → Xlittle:nmod

Can now use any CF parser, since CF dependency trees reflect adjacency.

The resulting CFG is fully lexicalized, with an enormous number of rules: one or more for each “word phrase” (eg, Xnews) or labelled word phrase (eg, Xnews:subj).

Eisner (1996) showed that the worst-case complexity of parsing these lexicalized grammars can be reduced by processing left and right dependents **independently**.

Graph-based Dependency Parsing

Goal: To find the highest scoring dependency tree in the space of all possible trees for a sentence (whether it is unambiguous, with one correct structure, or ambiguous, with one most likely structure).

Given a dependency tree T , define its score as the sum of the scores of its relevant subgraphs G_i , of which the simplest subgraphs are just individual dependency arcs (w_i, l, w_j) :

$$\Phi(T) = \sum_{i=1}^m \Phi(G_i) = \sum_{(w_i, l, w_j) \in A} \Phi((w_i, l, w_j))$$

This **arc-factored model** is an example of the global linear models we saw in Lecture 15.

Reminder: Global Linear Models (GLMs)

- A generating function GEN maps an input x to candidates trees y_1, \dots, y_n , so that

$$GEN(x) = \{y_1, \dots, y_n\}$$

- Each feature function f_i maps a parse tree (x, y) to a feature value $f_i(x, y)$
- Features are combined in a linear model:

$$\sum_i \lambda_i f_i(x, y)$$

- The goal of learning is to find the feature weights λ_i

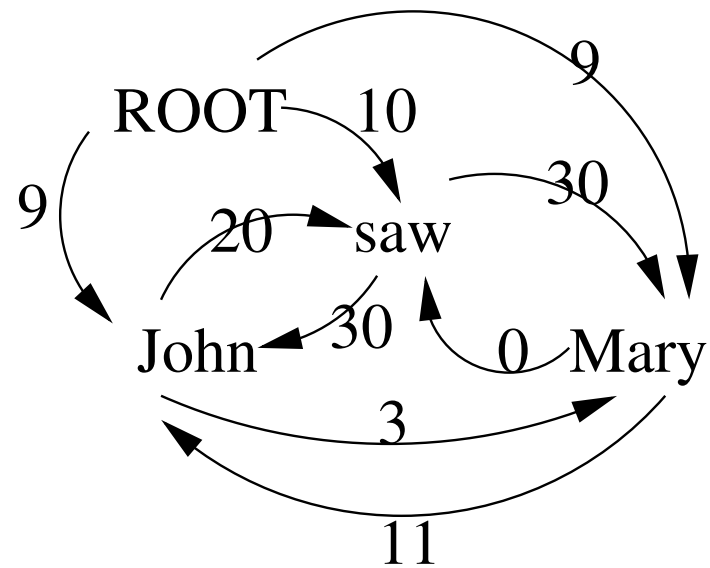
Graph-based Dependency Parsing as a GLM

- *GEN* maps an input sentence x to the set of all its dependency parses
- We have a set of feature functions f_i that map x and a dependency parse tree y to a feature value $f(x, y)$
- Features are combined using a linear model – usually the perceptron algorithm or a large-margin version of the perceptron.

MST Parsing

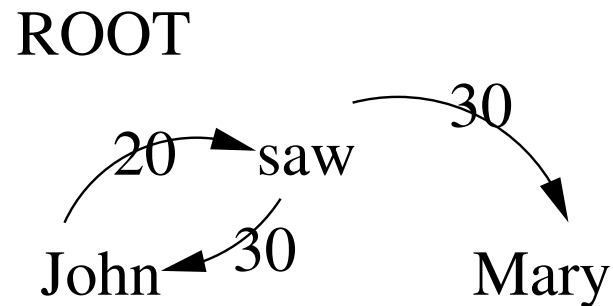
- McDonald et al. (2005) propose an extreme version of GLM graph-based dependency parsing that starts with a **totally connected dependency graph**, with a directed edge between every ordered pair of distinct words and a directed edge from the root to every word.
- A linear model learned from labeled data assigns a weighted score $s(i, j)$ to each directed edge, where $s(i, j) = \lambda f(i, j)$
- In general, $s(i, j) \neq s(j, i)$
- Finding the highest scoring dependency tree \equiv finding the **maximum spanning tree** (MST) in a graph containing all possible arcs.
- This can be solved in $O(n^2)$ time using the Chu-Liu-Edmonds algorithm, which can find both projective and non-projective MSTs.

Initial graph



Chu-Liu-Edmonds (CLE) Algorithm

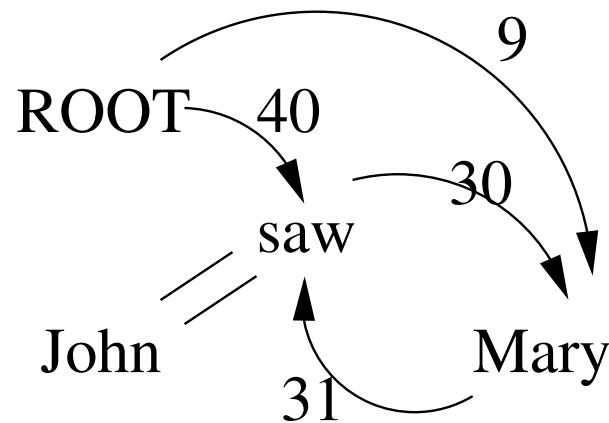
- Each node j in the graph greedily selects the incoming edge with the highest score $s(i, j)$.



- If a tree results, it is the maximum spanning tree.

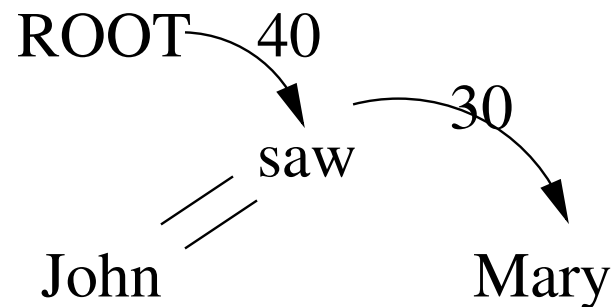
Recursion Step for CLE Algorithm

- If there is a cycle, as here, the nodes in the cycle are contracted to a single node, and the scores are recalculated for all inward and outward edges to that node, keeping track of the real word endpoints of all edges into and out of it.



Recursion Step for CLE Algorithm

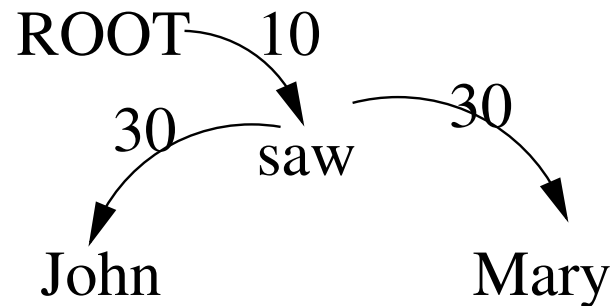
- An MST in the contracted graph can be transformed into an MST for the original graph, so we recursively call CLE algorithm on the contracted graph—
 - Greedily collecting incoming edges to all nodes:



- This is a tree, hence the MST, so we pop

Reconstruction Step for CLE Algorithm

- and reconstruct the uncontracted graph, in which the outward edge from the contracted node to *Mary* was from the word *saw*, while the inward edge from *ROOT* was to *saw* and from there to *John*. The uncontracted tree is therefore:



CLE Algorithm

- Parsing accuracy depends on how the scoring function for subgraphs is learned from treebank data.
- Standard approach, pioneered by McDonald et al. (2005) uses the perceptron algorithm or a large-margin version, to learn a weight vector that favors globally optimal dependency trees.
- Simple categorical features used in scoring (w_i, l, w_j) could include:
 - Identity of w_i , PoS-tag of w_i
 - Identity of w_j , PoS-tag of w_j
 - Label of l , Direction of l
 - Sequence of PoS-tags between w_i and w_j

PoS-tag of w_{i-1} , PoS-tag of w_{i+1}

PoS-tag of w_{j-1} , PoS-tag of w_{j+1}

⚡ While searching all possible nonprojective trees risks finding extremely bad trees for languages that are “primarily projective,” like English, or Czech, which has around 2% NPD, the model does learn weights that favor projective over nonprojective edges.

- In practice it does better on Czech than Eisner’s projective (CF) algorithm.

MST and Grammar

- ◊ Where is the **grammar** in the MST parser?
 - It is implicit in the **features**, expressing what is to the left and right, what your grandaparent is, etc.

- ◊ What **class** of grammars can be expressed in this way?
 - If we are allowed features like “isomorphic treestructures” and “unboundedly c-commanding *wh* element”, probably Type 0.
 - ◊ it might be quite hard to write such a feature grammar for e.g. Dutch.
 - It might also be quite hard to learn the model over such features

Transition-based Dependency Parsing

- Transition-based Dependency Parsing, like Graph-based parsing, has no explicit grammar.
- What is learned resembles learning a **shift-reduce parser** – ie, learning the best action to take next. Just a richer repertoire of actions.
- Transition system consists of
 - Set of configurations C representing partial analyses of the sentence.
 - Set of transitions D between them.
- Every sentence S has a unique initial configuration $c_s(S)$ and a set of terminal configurations $C_t(S)$, each representing a complete analysis of S .

Example transition system in Nivre (2010)

- A configuration $c = (\sigma, \beta, A)$, where σ is a **stack** of nodes, β is a buffer of remaining input nodes (ie, words), and A is a set of labelled dependency arcs.
- For sentence S , initial configuration $c_i(S) = ([w_0], [w_1, \dots, w_n], \emptyset)$
- Its set of terminal configurations $C_t(S) = \{c \in C \mid c = ([w_0], [], A)\}$

- Transition set D includes
 - SHIFT: $(\sigma, [w_i | \beta], A) \Rightarrow ([\sigma | w_i], \beta, A)$
 - RIGHT-ARC(l): $([\sigma | w_i, w_j], \beta, A) \Rightarrow ([\sigma | w_i], \beta, A \cup \{(w_i, l, w_j)\})$
 - LEFT-ARC(l): $([\sigma | w_i, w_j], \beta, A) \Rightarrow ([\sigma | w_j], \beta, A \cup \{(w_j, l, w_i)\})$

SHIFT pushes the head of the buffer onto the stack.

RIGHT-ARC and LEFT-ARC add a dependency arc between the top two nodes popped from the stack and pushes the head node of the arc.

Learning Task in Transition-based Parsing

- Learn the sequence of actions that yields the most likely dependency analysis.
- According to Nivre (2010), most important features involved attributes of input words, defined by their position in the configuration.
- Each such feature can be defined in terms of two simpler features:
 - An *address function* – eg, “the top node on the stack”, “the second node on the stack”, “the dependents of the top node on the stack”, “the front of the buffer”, “the next node in the buffer”, etc.
 - An *attribute function* – eg, its PoS-tag (static), its dependency label (computed during parse process).

Nivre (2010) lists papers describing work on learning for Transition-based Parsing.

McDonald and Nivre (2007)

- Transition-based parsing optimizes with respect to local decisions based on a very rich set of features.
- MALT parser is a greedy deterministic transition-based dependency parser that uses a SVM-learned state-transition table, augmented with “pseudo-projective” dependency-graph transformation techniques.
- McDonald and Nivre compare the accuracy for many languages of MST versus MALT.
- The comparison shows that MST has quite high precision (around 70%) but low recall (falling to around 40%) of long range dependencies, including the non projective ones.

- MALT is a bit better on very short range dependencies.
 - MST provides an important baseline for parsers that capture long-range dependencies using linguistically expressive grammars.
- ⚡ like all discriminative methods, it depends on being fed **linguistically informed features**.

References

- Eisner, Jason. 1996. Three new probabilistic models for dependency parsing. In *Proceedings of the 16th Conference on Computational Linguistics (CoLING 96)*, 340–345. Saarbruecken: ACL.
- McDonald, Ryan, and Joakim Nivre. 2007. Characterizing the errors of data-driven parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 122–131. Prague: ACL.
- McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 523–530. Vancouver: ACL.
- Nivre, Joakim. 2010. Dependency parsing. *Language and Linguistics Compass* 4/3: 138–152.