

Dependency Grammars and Parsers

Deep Processing for NLP
Ling571
January 30, 2017

Roadmap

- Dependency Grammars
 - Definition
 - Motivation:
 - Limitations of Context-Free Grammars
- Dependency Parsing
 - By conversion to CFG
 - By Graph-based models
 - By transition-based parsing

Dependency Grammar

- CFGs:
 - Phrase-structure grammars
 - Focus on modeling constituent structure
- Dependency grammars:
 - Syntactic structure described in terms of
 - Words
 - Syntactic/Semantic relations between words

Dependency Parse

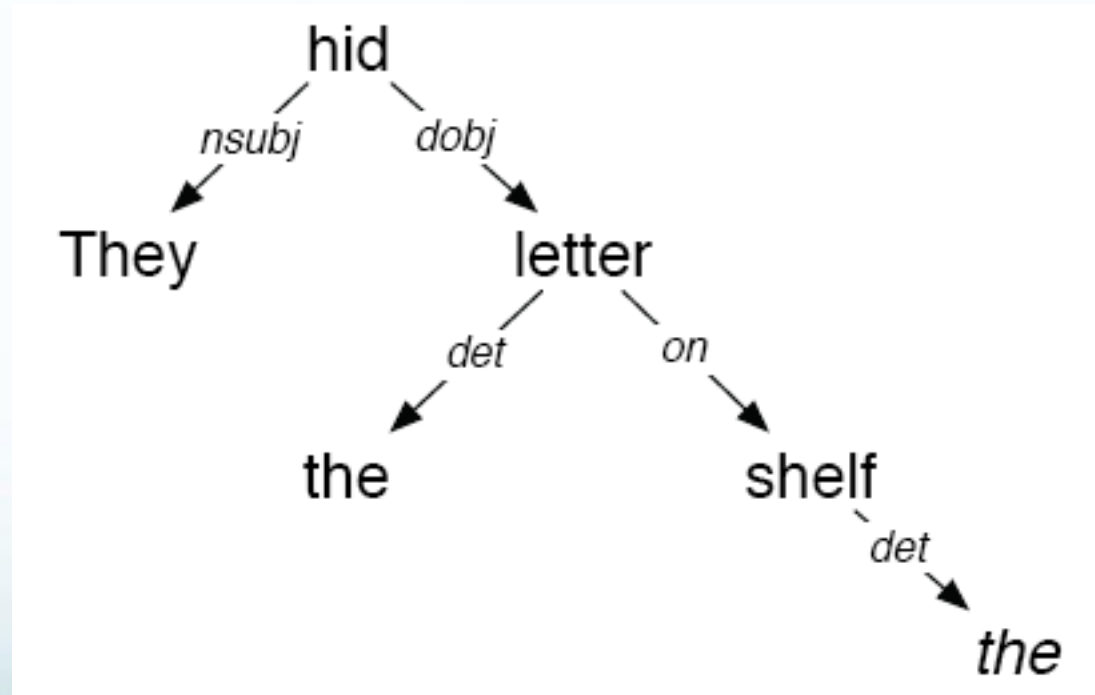
- A dependency parse is a tree, where
 - Nodes correspond to words in utterance
 - Edges between nodes represent dependency relations
 - Relations may be labeled (or not)

Dependency Relations

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	Description
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Dependency Parse Example

— They hid the letter on the shelf



Why Dependency Grammar?

- More natural representation for many tasks
 - Clear encapsulation of predicate-argument structure
 - Phrase structure may obscure, e.g. wh-movement
- Good match for question-answering, relation extraction
 - Who did what to whom
- Build on parallelism of relations between question/relation specifications and answer sentences

Why Dependency Grammar?

- Easier handling of flexible or free word order
 - How does CFG handle variations in word order?
 - Adds extra phrases structure rules for alternatives
 - Minor issue in English, explosive in other langs
 - What about dependency grammar?
 - No difference: link represents relation
 - Abstracts away from surface word order

Why Dependency Grammar?

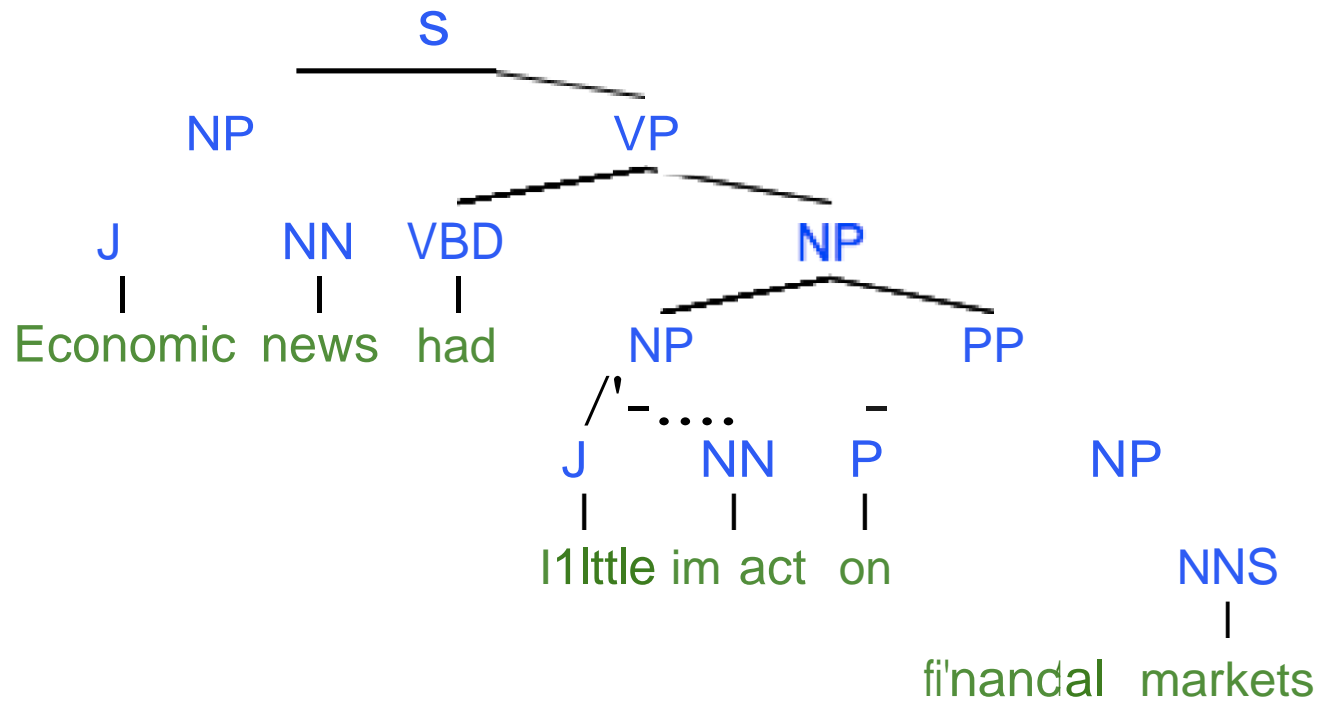
- Natural efficiencies:
 - CFG: Must derive full trees of many non-terminals
- Dependency parsing:
 - For each word, must identify
 - Syntactic head, h
 - Dependency label, d
 - Inherently lexicalized
 - Strong constraints hold between pairs of words

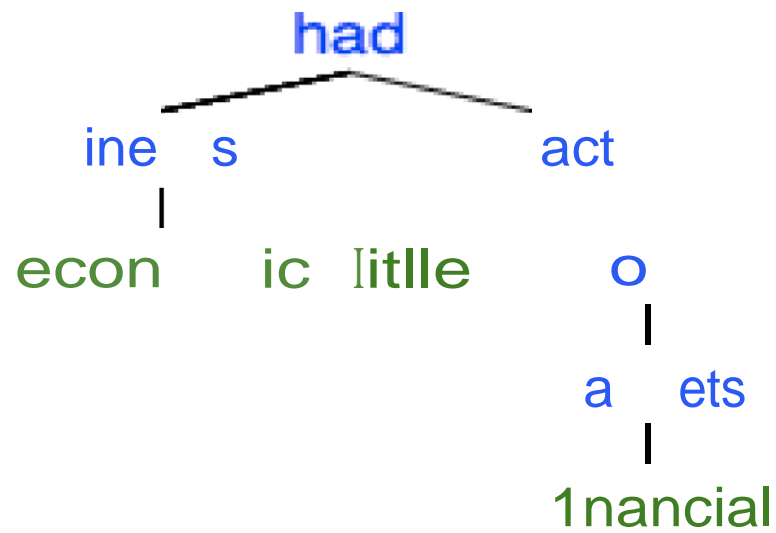
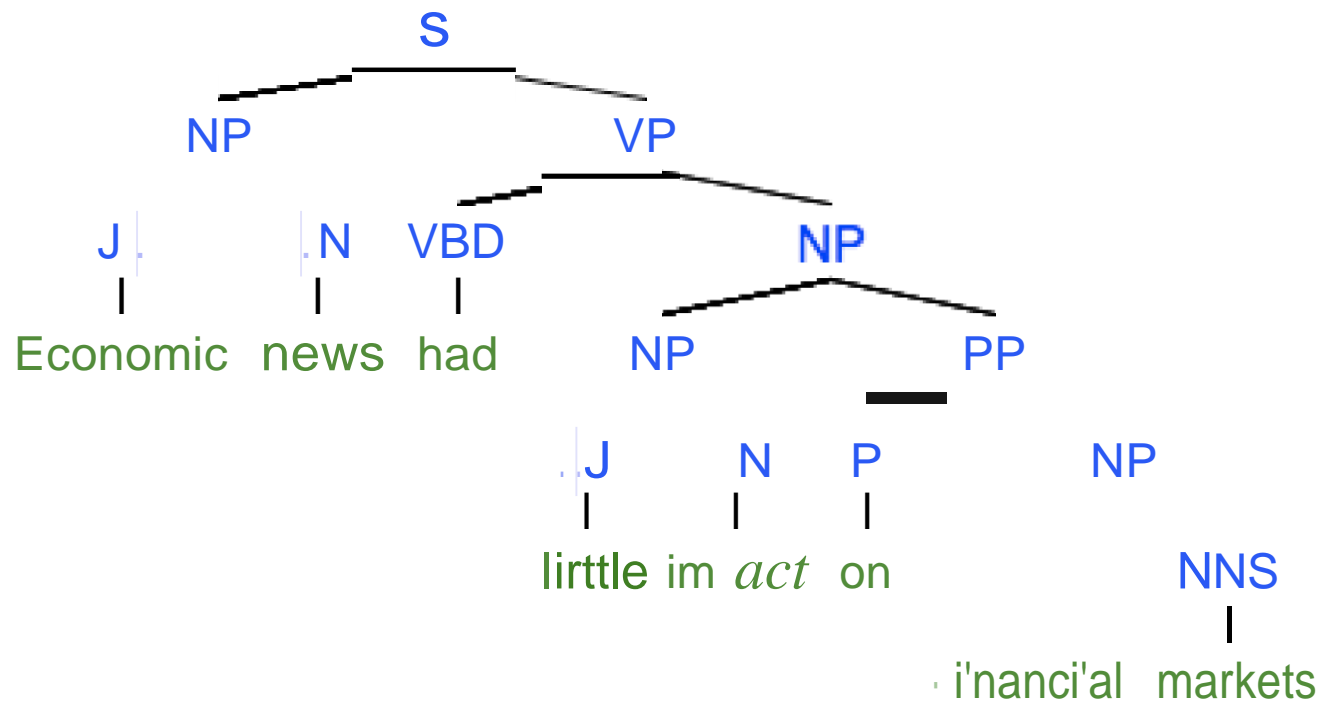
Summary

- Dependency grammar balances complexity and expressiveness
- Sufficiently expressive to capture predicate-argument structure
- Sufficiently constrained to allow efficient parsing

Conversion

- Can convert phrase structure to dependency trees
 - Unlabeled dependencies
- Algorithm:
 - Identify all head children in PS tree
 - Make head of each non-head-child depend on head of head-child



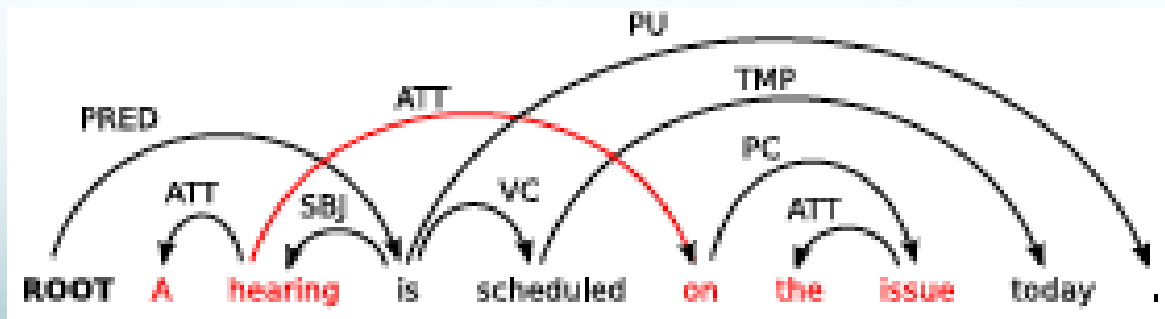
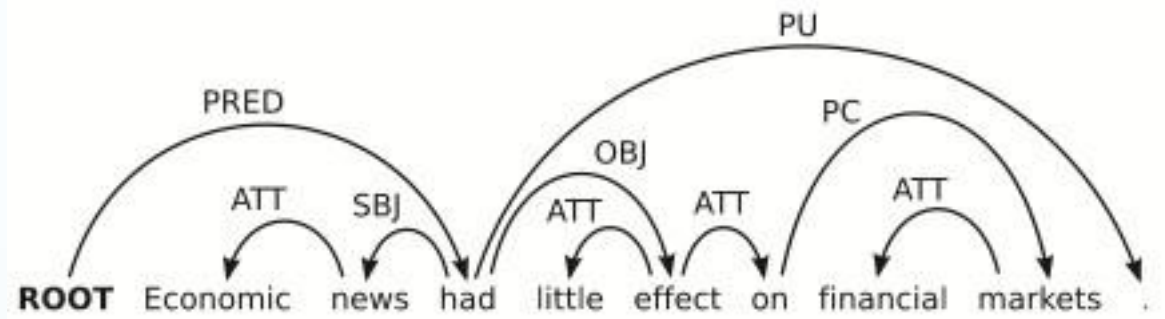


Dependency Parsing

- Three main strategies:
 - Convert dependency trees to PS trees
 - Parse using standard algorithms $O(n^3)$
 - Employ graph-based optimization
 - Weights learned by machine learning
 - Shift-reduce approaches based on current word/state
 - Attachment based on machine learning

Parsing by PS Conversion

- Can map any projective dependency tree to PS tree
- Non-terminals indexed by words
 - “Projective”: no crossing dependency arcs for ordered words

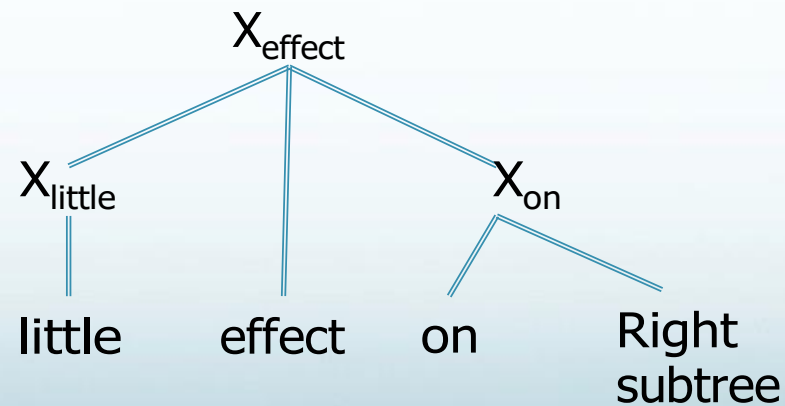
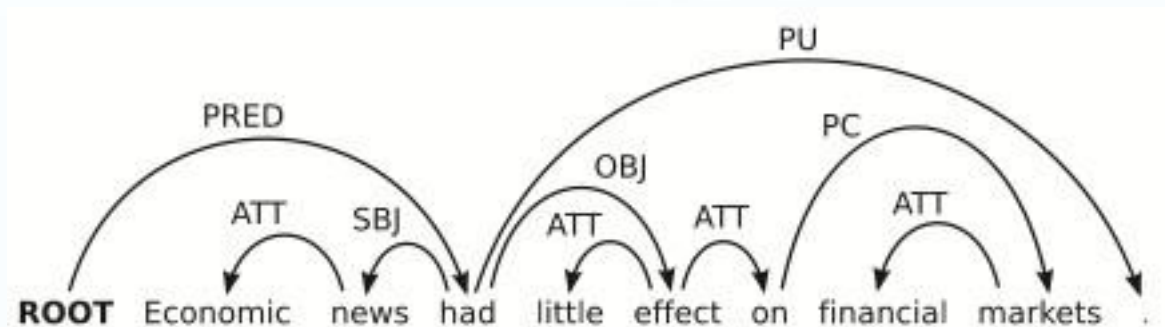


Dep to PS Tree Conversion

- For each node w with outgoing arcs,
 - Convert the subtree w and its dependents t_1, \dots, t_n to
 - New subtree rooted at X_w with child w and
 - Subtrees at t_1, \dots, t_n in the original sentence order

Dep to PS Tree Conversion

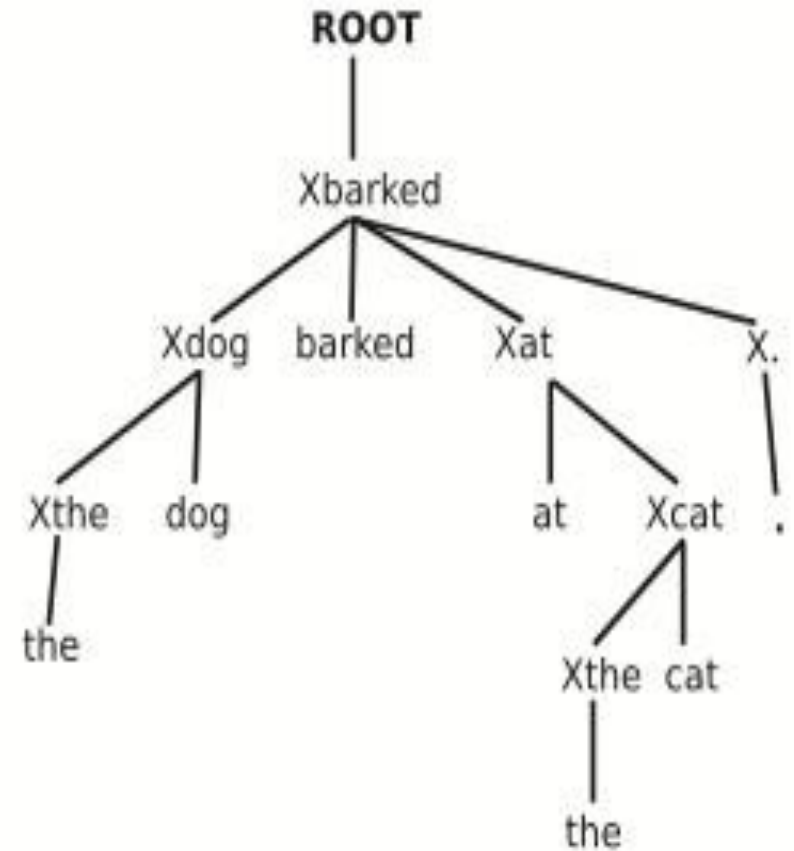
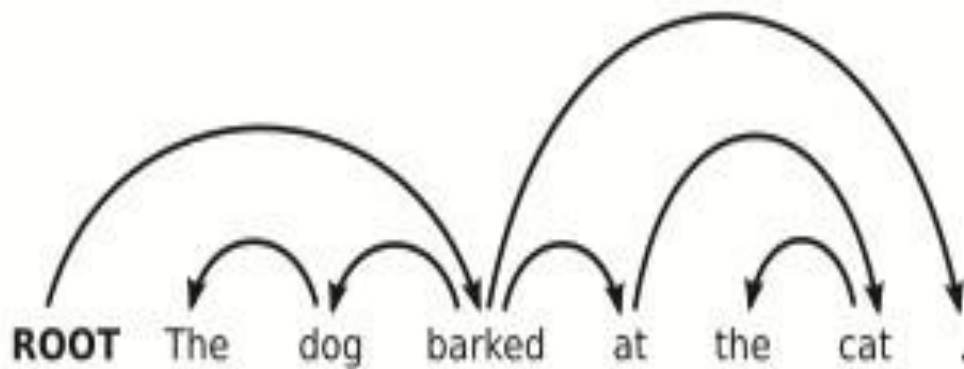
E.g ., for 'effect'



Dep to PS Tree Conversion

- What about the dependency labels?
 - Attach labels to non-terminals associated with non-heads
 - E.g. $X_{\text{little}} \rightarrow\rightarrow X_{\text{little:nmod}}$
- Doesn't create typical PS trees
 - Does create fully lexicalized, context-free trees
 - Also labeled
- Can be parsed with any standard CFG parser
 - E.g. CKY, Earley

Full Example Trees



Example from J. Moore, 2013

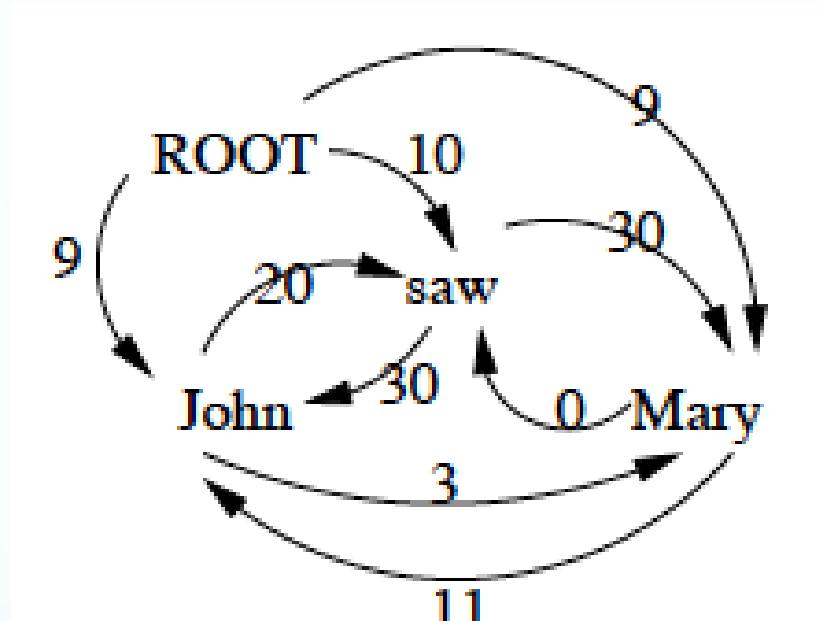
Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree T for sentence S
 - If S is unambiguous, T is the correct parse.
 - If S is ambiguous, T is the highest scoring parse.
- Where do scores come from?
 - Weights on dependency edges by machine learning
 - Learned from large dependency treebank
- Where are the grammar rules?
 - There aren't any; data-driven processing

Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree
- Idea:
 - Build initial graph: fully connected
 - Nodes: words in sentence to parse
 - Edges: Directed edges between all words
 - + Edges from ROOT to all words
 - Identify maximum spanning tree
 - Tree s.t. all nodes are connected
 - Select such tree with highest weight
 - Arc-factored model: Weights depend on end nodes & link
 - Weight of tree is sum of participating arcs

Initial Tree

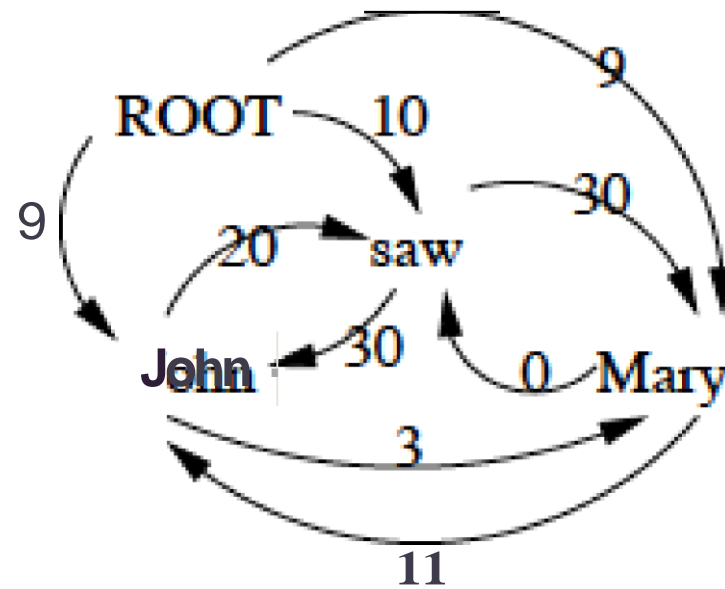


- Sentence: John saw Mary (McDonald et al, 2005)
 - All words connected; ROOT only has outgoing arcs
- Goal: Remove arcs to create a tree covering all words
 - Resulting tree is dependency parse

Maximum Spanning Tree

- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)
- Sketch of algorithm:
 - For each node, greedily select incoming arc with max w
 - If the resulting set of arcs forms a tree, this is the MST.
 - If not, there must be a cycle.
 - “Contract” the cycle: Treat it as a single vertex
 - Recalculate weights into/out of the new vertex
 - Recursively do MST algorithm on resulting graph
- Running time: naïve: $O(n^3)$; Tarjan: $O(n^2)$
 - Applicable to non-projective graphs

Initial Tree



CLE: Step 1

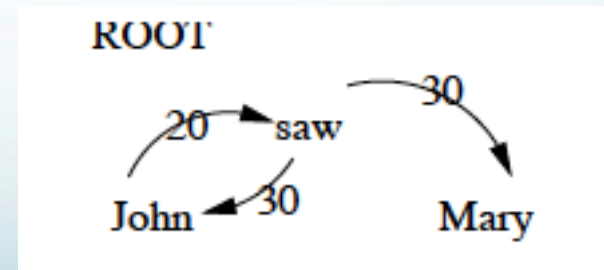
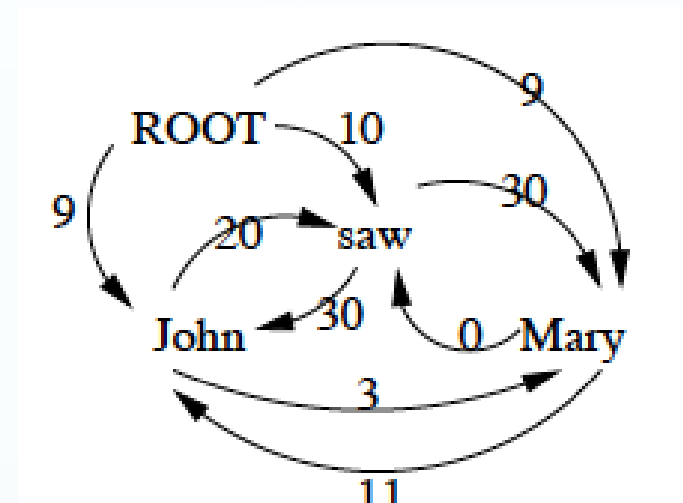
Find maximum incoming arcs

Is the result a tree?

No

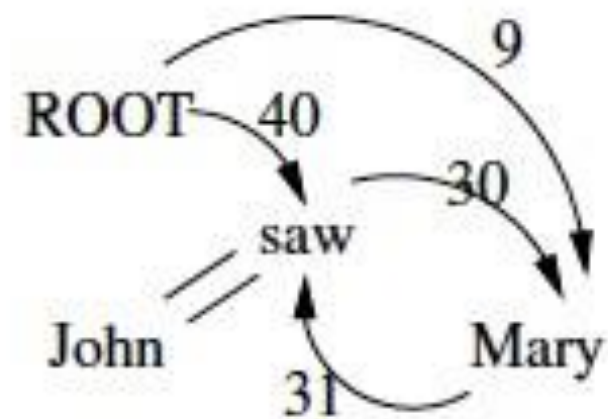
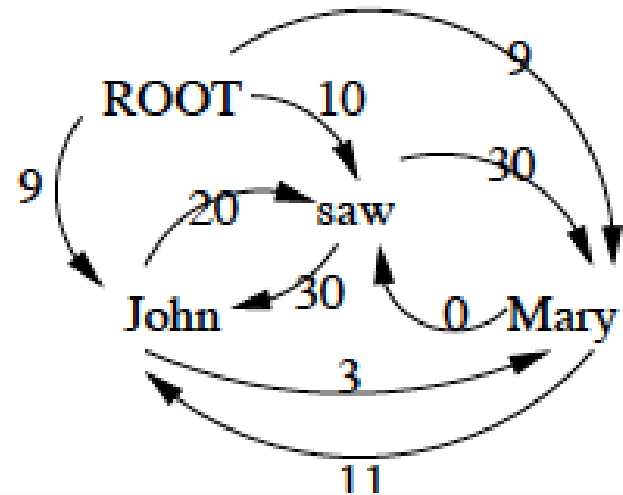
Is there a cycle?

Yes, John/saw

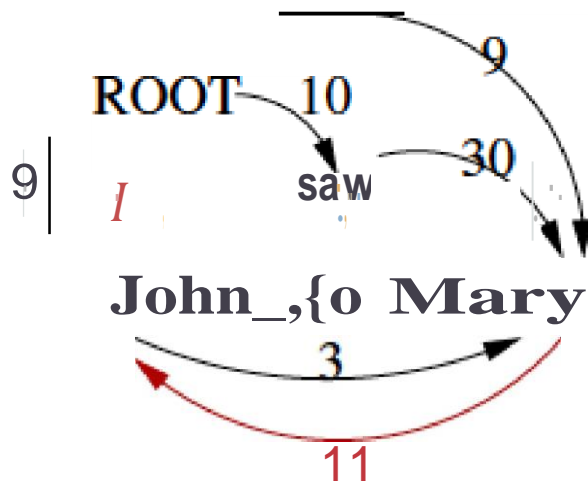


CLE: Step 2

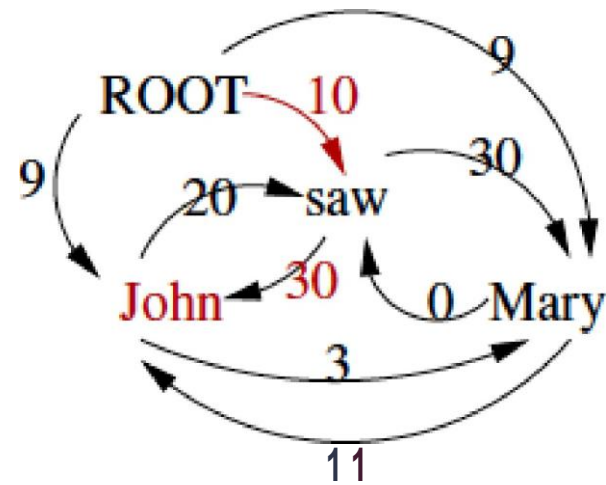
- Since there's a cycle:
 - Contract cycle & reweight
 - John+saw as single vertex
 - Calculate weights in & out as:
 - Maximum based on internal arc
 - and original nodes
- Recurse



Calculating Graph



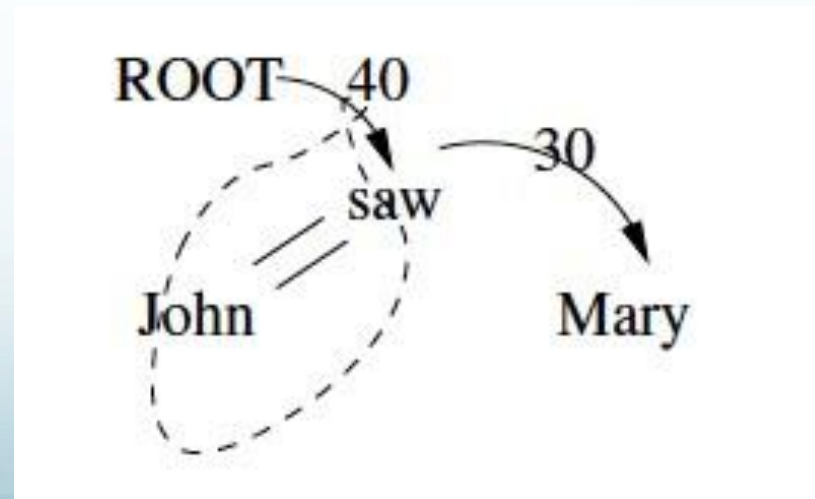
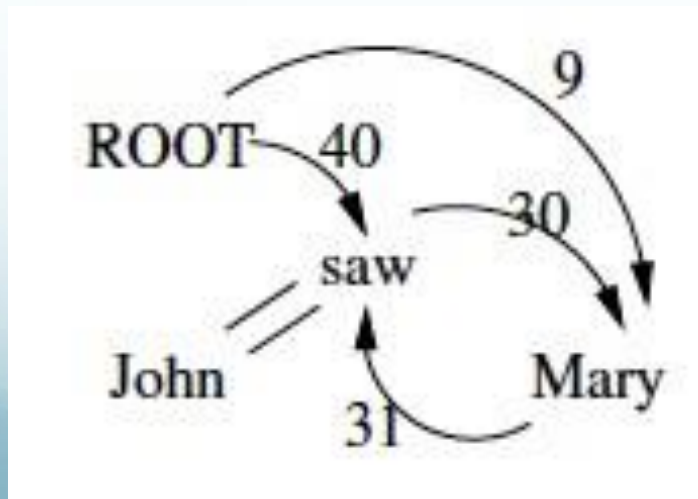
$$s(\text{Mary}, C) = 11 + 20 = 31$$



$$s(\text{ROOT}, C) = 10 + 30 = 40$$

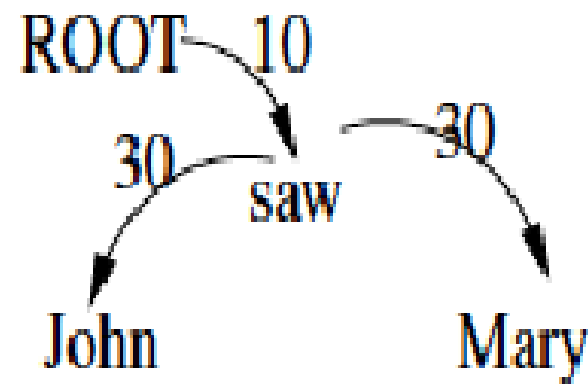
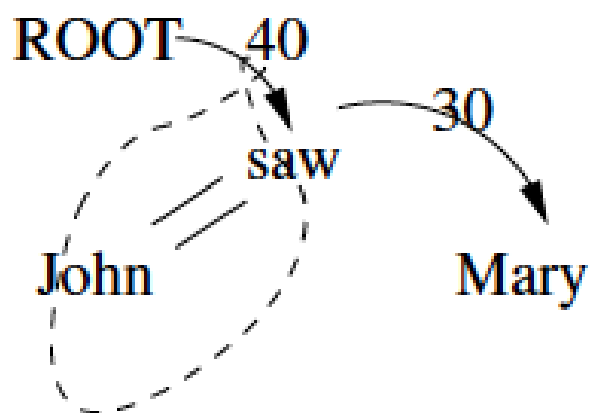
CLE: Recursive Step

- In new graph, find graph of
 - Max weight incoming arc for each word
- Is it a tree? Yes!
 - MST, but must recover internal arcs $\rightarrow\rightarrow$ parse



CLE: Recovering Graph

- Found maximum spanning tree
 - Need to 'pop' collapsed nodes
- Expand "ROOT $\rightarrow\rightarrow$ John+saw" = 40
- MST and complete dependency parse



Learning Weights

- Weights for arc-factored model learned from corpus
 - Weights learned for tuple (w_i, w_j, l)
- McDonald et al, 2005 employed discriminative ML
 - Perceptron algorithm or large margin variant
- Operates on vector of local features

Features for Learning Weights

- Simple categorical features for (w_i, L, w_j) including:
 - Identity of w_i (or char 5-gram prefix), POS of w_i
 - Identity of w_j (or char 5-gram prefix), POS of w_j
 - Label of L , direction of L
 - Sequence of POS tags b/t w_i, w_j
 - Number of words b/t w_i, 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}
- Features conjoined with direction of attachment and distance b/t words

Dependency Parsing

- Dependency grammars:
 - Compactly represent pred-arg structure
 - Lexicalized, localized
 - Natural handling of flexible word order
- Dependency parsing:
 - Conversion to phrase structure trees
 - Graph-based parsing (MST), efficient non-proj $O(n^2)$
 - Transition-based parser
 - MALTparser: very efficient $O(n)$
 - Optimizes local decisions based on many rich features