Dependency Parsing

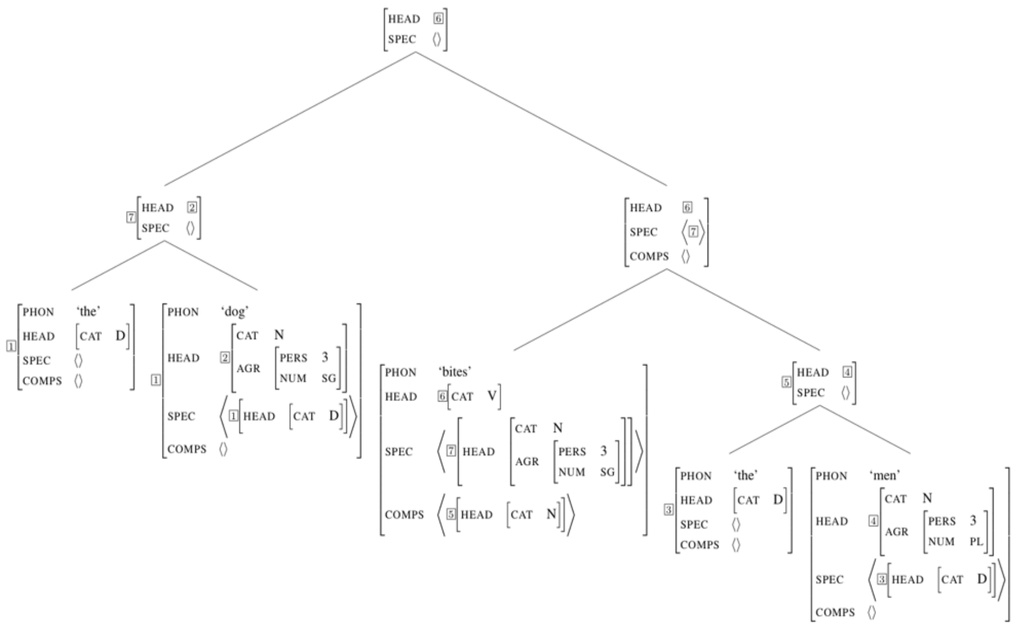
Language Technology 1
WS WS 2015

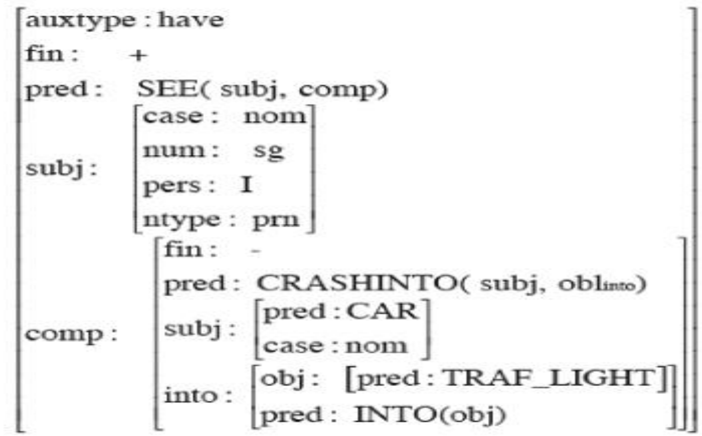
Günter Neumann

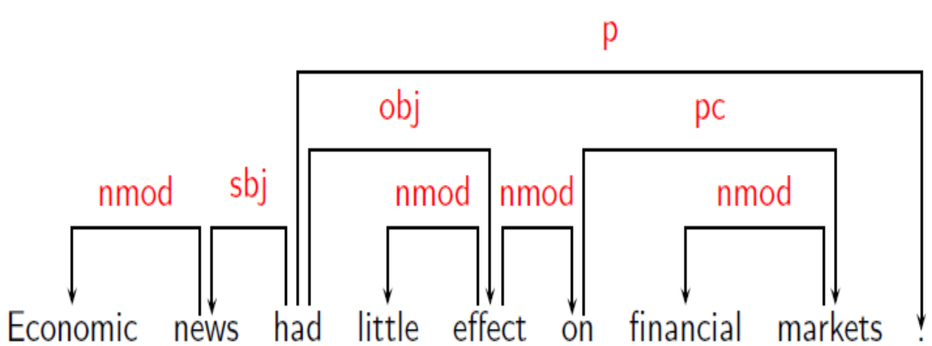
+ Overview

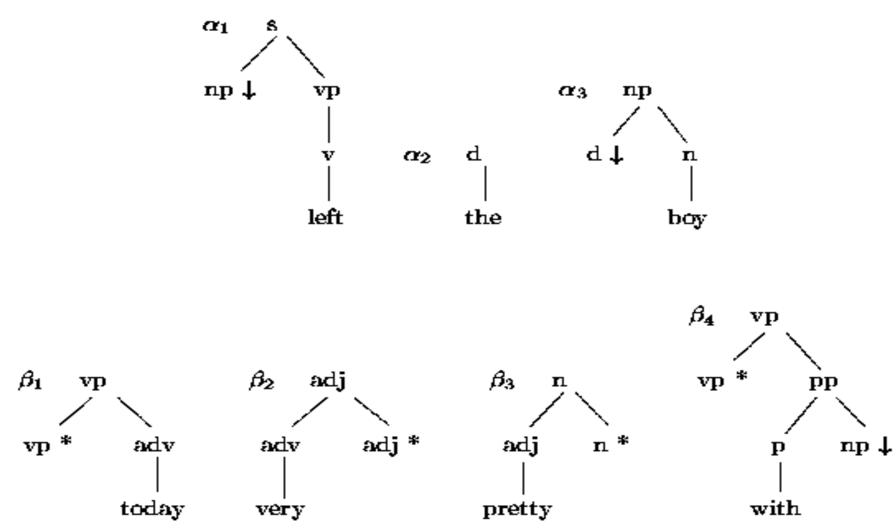
- Dependency Grammar vs Dependency Parsing
- Transition-Based vs Graph-Based Dependency Parsing

Syntactic Theories









+ Dependency Representation

- The basic idea:
  Syntactic structure consists of lexical items, linked by binary, asymmetric, directed, anti-reflexive, anti-transitive, labeled relations called dependencies.

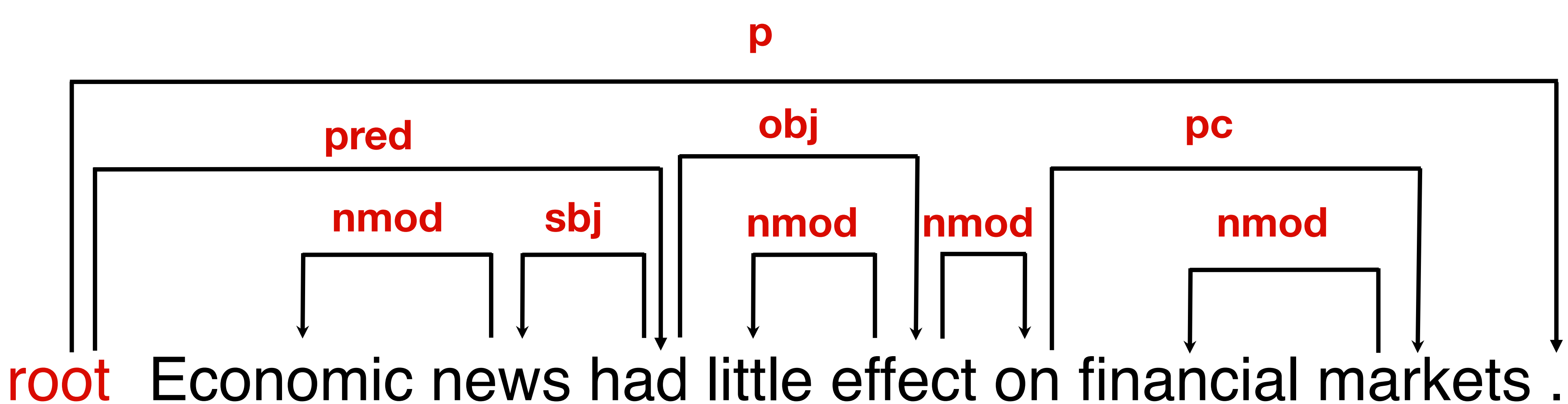
- A → B; <B,A>

  (A is head/parent/governor; B is dependent/child/subordinate)

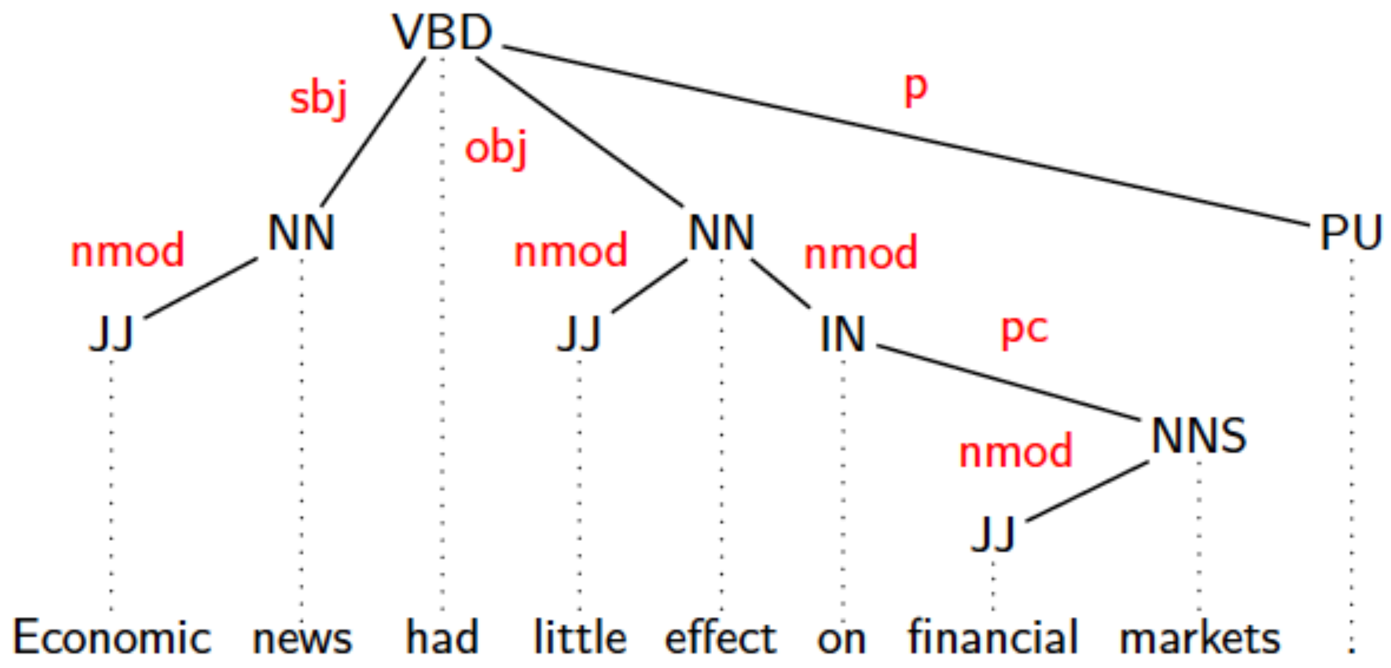
- Syntactic structures are usually trees, i.e. they have the following properties:
  connectedness, single-headiness, rooted, acyclicity, (projectivity)

Connected, A-cyclic, Single-head

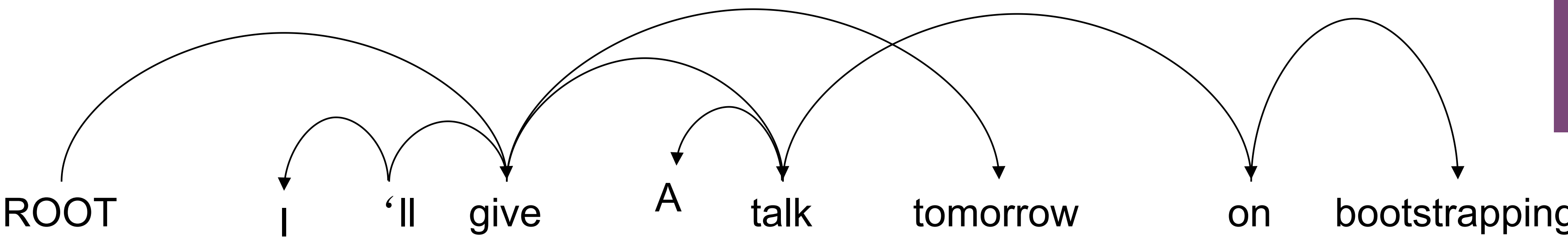
- A syntactic structure is complete (connected)
- A syntactic structure is hierarchical (acyclic)
- Each word has at most one head (single head)
- Adding a special root-node can enforce connectedness.

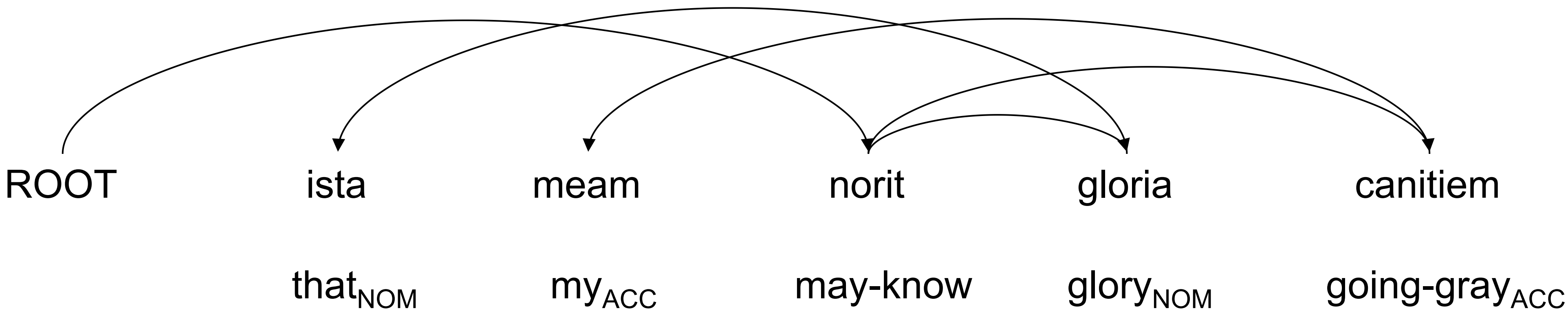


Example of a Projective Dependency Tree



Nonprojective Syntax





That glory shall last till I go gray

Slide from [Smith & Smith, EMNLP, 2007]

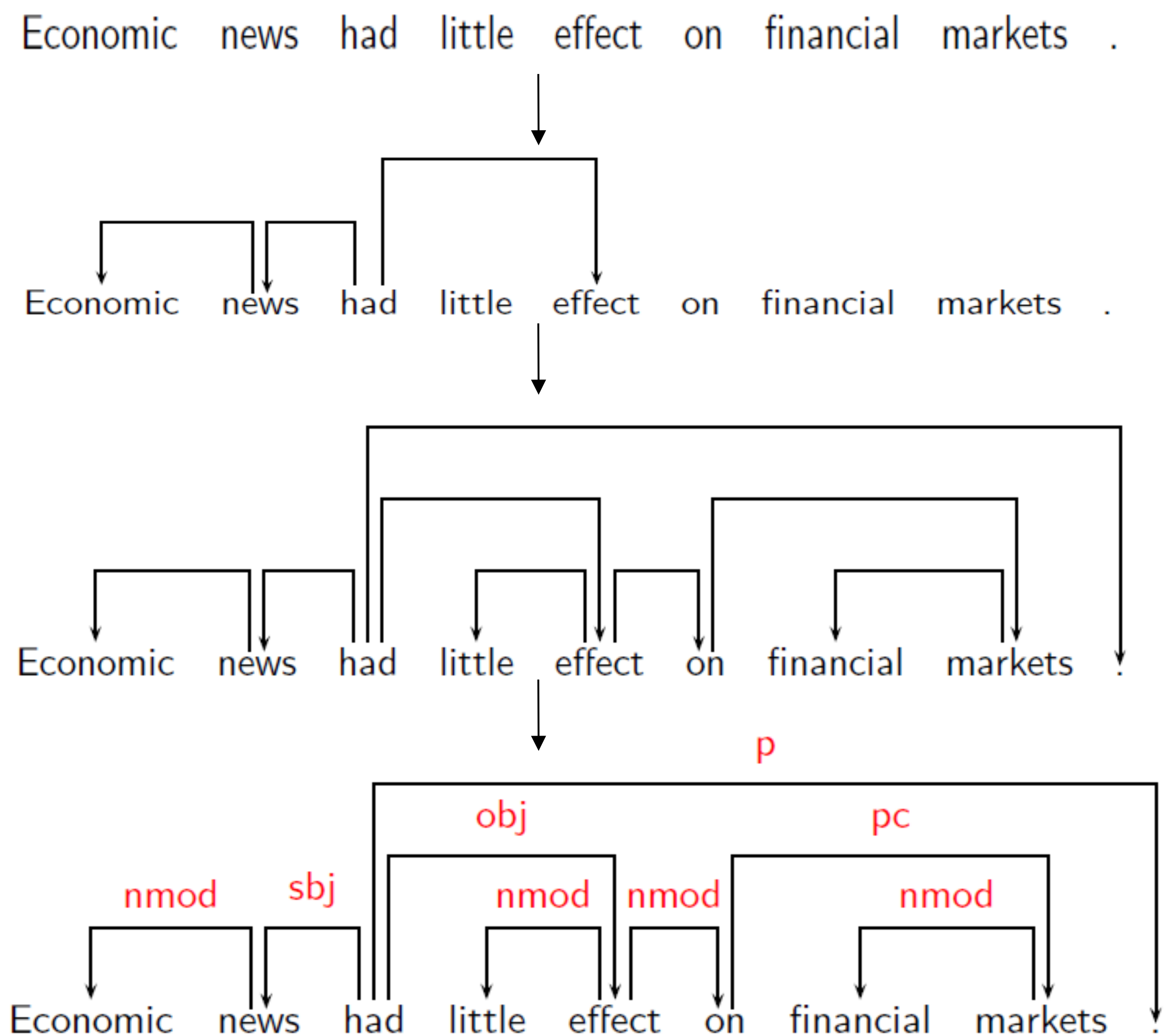
+ Dependency Grammar

- History:
  ancient Greek, Sanskrit, Latin, Arabic, medieval Europe, 1900s
- Problematic phenomena:
  coordination, no groupings, auxiliaries
- Variations:
  single vs. multiple layers (morphology, syntax), different tagsets and structures (Stanford vs. CoNLL)

Dependency Parsing

- The problem
  - Input:
    sentence $x = w_0, w_1, \ldots, w_n$ with $w_0 =$ root
  - Output:
    dependency graph $G = (V, A)$ for x whereby:
    - $V = \{0, 1, \ldots, n\}$ is the node set
    - A is the edge set, i.e., $(i, j, k) \in A$ represents a dependency from w_i to w_j with label $l_k \in L$

Parsing


Economic news had little effect on financial markets .
Economic news had little effect on financial markets .
Economic news had little effect on financial markets .
p
obj
pc
nmod
sbj
nmod
nmod
nmod
Economic news had little effect on financial markets .

+ Dependency Parsing

- Easy to implement
  - No artificial (non-terminal) nodes
  - Linear complexity possible (deterministic parsing)
- Easy to evaluate
  - Attachment scores are very straightforward
- Very expressive
  - Suitable for free word order languages
- Useful representations
  - Very close to semantics, which is very often done next

Applications

- Almost any language technology can profit from dependency parsing:
  - Machine Translation
  - Information Extraction
  - Textual Entailment
  - Question Answering
  - Summarisation
  - Text Generation

Grammar vs. Data-Driven

- Rule systems:
  - Lists of words for every category
  - Which categories occur with which categories
  - Valency
- Data-driven systems:
  - Use tree banks to learn how to link words
  - Dependency tree banks are available for many languages (CoNLL-X shared task)

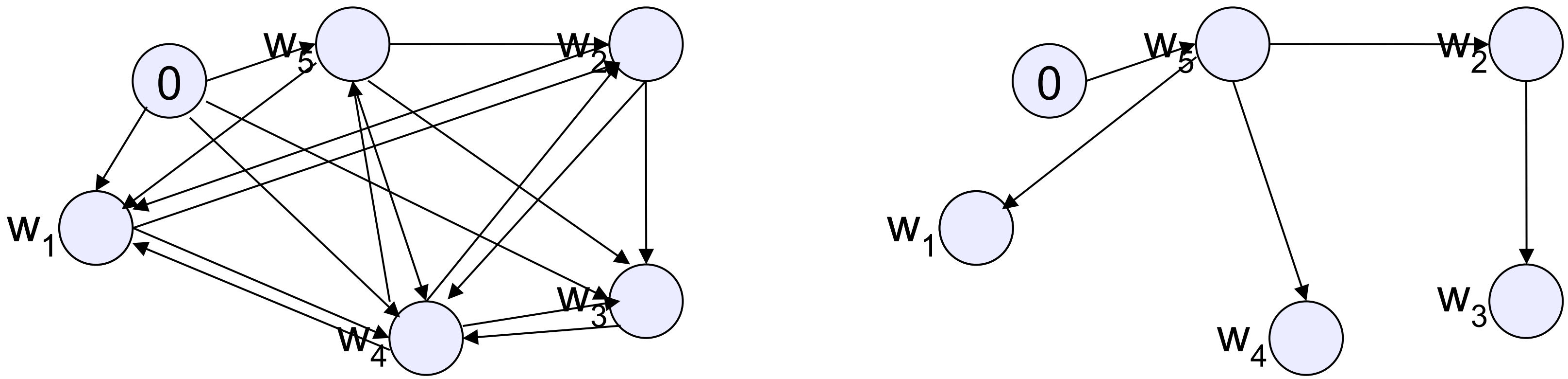
Transition-Based vs. Graph-Based

- Two predominant parser types
  - similar performance
  - completely different approaches
- Transition-based:
  - the result is constructed after a series of transitions (local decisions)
- Graph-based:
  - the result is constructed in few steps (global decisions)
  - Details from here: http://www.ryanmcd.com/courses/esslli2007/esslli4.pdf

Graph-Based Parsing

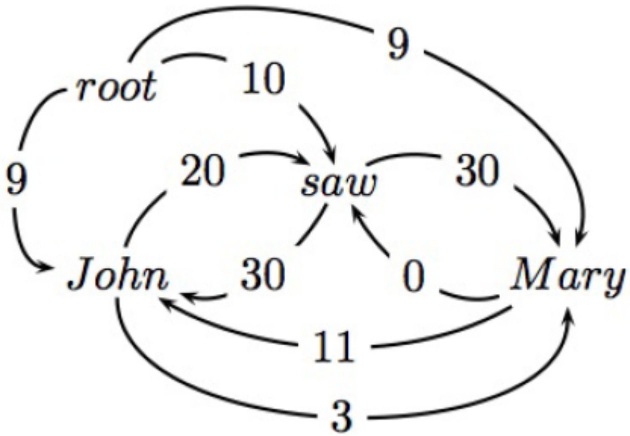
- Given the input $I = w_1, w_2, \ldots, w_n$, where each word corresponds to a node $v_1, v_2, \ldots, v_n$, find a graph $G = (V, A)$, such that G is a rooted tree and $A = \{<A_1, B_1>, <A_2, B_2>, \ldots, <A_n, B_n>\}$ corresponds to the correct dependency tree.

- Solution: Maximum Spanning Trees (MST) (the tree with the highest weight)

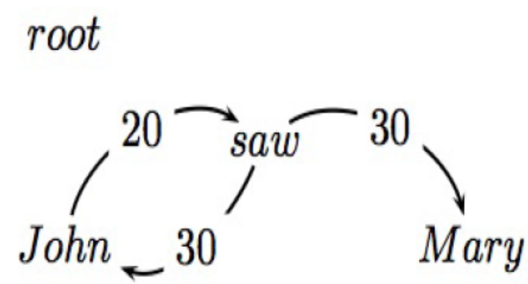


Chu-Liu-Edmonds

x = root John saw Mary



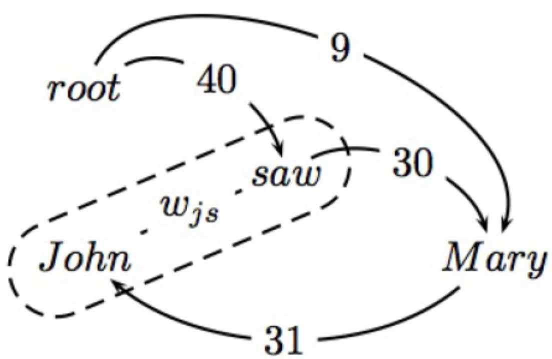
Find highest scoring incoming arc for each vertex



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

If not a tree, identify cycle and contract

Recalculate arc weights into and out-of cycle



Taken from Introduction to Data-Driven Dependency Parsing (Ryan McDonald, Joakim Nivre)

Edmonds Algorithm

- For all nodes (modulo root node): Choose the best incoming edge
- Repeat (greedily) until the graph contains no a cycle
  - Consider each cycle as a virtual node. Compute modified edge weights for all edges which enter the cycle from outside
  - Idea: distribute (add) weights of edges of cycle to the incoming edges of the virtual node, e.g.,
    - w_n(root,saw) = w(root,saw) + w(saw,john)
    - 40 = 10 + 30

Graph-Based Parsing

- Advantages:
  - State-of-the art performance
  - Works well for long sentences/dependencies
- Disadvantages:
  - Not incremental
  - Computationally expensive (Chu-Liu-Edmonds need O(n*n) to find MST)

+ Transition-Based Parsing

- The parse of the sentence is a sequence of operations (transitions)
- The result is a complete set of dependency pairs, which satisfy tree constraints
- An oracle tells the parser what action should be taken in every step:
  - Training - use training data for simulating a perfect oracle (you have the desired result given)
  - Application - use classifiers for simulating an oracle (train models, that allow the oracle to choose correct actions)

Transition System

- Given the input $I = w_1, w_2, \ldots, w_n$ perform $S = c_0, c_1, \ldots, c_n$, such that
  $A = \{<A_1, B_1>, <A_2, B_2>, \ldots, <A_n, B_n>\}$ corresponds to the correct dependency tree

- Configuration – state of the parser
  - Define the set of possible transitions, e.g.: left_link(a, b)
  - Conditions (permissibility):
    - b should not have a parent; if <a, b> is added to A, A should not contain a cycle etc.

- Effects:
  - left_link(a, b) → a becomes the parent of b
  - right_link(a, b) → b becomes the parent of a
  - shift(a, b) → move on to next pair

- Initial configuration / terminal configuration

Parsing Algorithms

- Naïve:
  - For every word *j* in the sentence try to combine it with other words *i* in the sentence (i < j):
  - Possible operations:

    make j the parent of i

    make i the parent of j

    do not combine and j+1, i = 0

    do not combine and i+1

    Initial state: Start with the first word

    Terminal state: j > sentence length
- Nivre (Arc-Eager, Arc-Standard)
- Covington's parsing strategy

Ex: $_0$John$_1$saw$_2$Mary$_3$.$_4$

- c_0: j =1; i = 0, A = {}: initial state
- $c_0 \rightarrow c_1$: do not combine; i+1 (j=1, i=1, A = {})
- $c_1 \rightarrow c_2$: do not combine; i+1 (j=1, i=2, A = {})
- $c_2 \rightarrow c_3$: make i the parent of j; (j=1, i=2, A = {<1,2>})
- $c_3 \rightarrow c_4$: do not combine; i+1 (j=1, i=3, A = {<1,2>})
- $c_4 \rightarrow c_5$: do not combine; i+1 (j=1, i=4, A = {<1,2>})
- $c_5 \rightarrow c_6$: do not combine; j+1 (j=2, i=0, A = {<1,2>})
- $c_6 \rightarrow c_7$: make i the parent of j (j=2, i=0, A = {<1,2>,<2,0>})
- $c_7 \rightarrow c_8$: do not combine; i+1 (j=2, i=1, A = {<1,2>,<2,0>})
- $c_8 \rightarrow c_9$: do not combine; i+1 (j=2, i=2, A = {<1,2>,<2,0>})
- $c_9 \rightarrow c_{10}$: do not combine; i+1 (j=2, i=3, A = {<1,2>,<2,0>})
- $c_{10} \rightarrow c_{11}$: make j the part of i (j=2, i=3, A = {<1,2>,<2,0>, <3,2>})
- $c_{11} \rightarrow c_{12}$: do not combine; i+1 (j=2, i=4, A = {<1,2>,<2,0>, <3,2>})
- $c_{12} \rightarrow c_{13}$: make j the part of i (j=2, i=4, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{13} \rightarrow c_{14}$: do not combine; j+1 (j=3, i=0, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{14} \rightarrow c_{15}$: do not combine;i+1 (j=3, i=1, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{15} \rightarrow c_{16}$: do not combine;i+1 (j=3, i=2, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{16} \rightarrow c_{17}$: do not combine;i+1 (j=3, i=3, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{17} \rightarrow c_{18}$: do not combine;i+1 (j=3, i=4, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{18} \rightarrow c_{19}$: do not combine; j+1 (j=4, i=0, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{19} \rightarrow c_{20}$: do not combine;i+1 (j=4, i=1, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{20} \rightarrow c_{21}$: do not combine;i+1 (j=4, i=2, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{21} \rightarrow c_{22}$: do not combine;i+1 (j=4, i=3, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- $c_{22} \rightarrow c_{23}$: do not combine;i+1 (j=4, i=4, A = {<1,2>,<2,0>, <3,2>,<4,2>})
- c_{23}: terminal configuration

+ Naive Algorithm

- Obvious disadvantages:
  - Too many senseless configurations
  - $O(n^2)$ runtime (if no readings are considered)

- Advantage:
  - Simple to implement

Oracle

- Which transition to chose in which state?

- Every configuration is transformed to a feature vector:
  - The history of previous transitions can be used
  - Word information and context information is available
  - External resources can be used

+ Feature Models: : ${}_0$John${}_1$saw${}_2$Mary${}_3$.${}_4$

- Sample configuration:
  - (j=2, i=3, A = {<1,2>,<2,0>})
- Feature templates:
  - Word form of token x: wf(x)
  - Pos tag of token x: pos(x)
  - Distance between tokens x and y: dist(x,y)
  - Is token x the root node?: isRoot(x)
- Features:
  - wf(2)=saw, wf(3)=Mary, pos(2)=VBD, pos(3)=NNP, dist(2,3)=1, isRoot(2)=true, wf(1)=John, pos(1)=NNP
- Transition: *make j the part of i*
- For some learning approaches very complex feature engineering is required

Supervised Machine Learning

- Compute all feature vectors for all annotated sentences from training corpus
- Print all feature vectors into a file in the format required by the machine learning method of your choice:
  - wfi=Mary posi=NNP wfj=saw posj=VBD link2
  - wfi=Mary posi=NNP wfj=John posj=NNP shift
- Or
  - 1:1 2:1 3:1 4:1 0
  - 1:1 2:1 5:1 6:1 1
  - Define alphabet:
    - (1 - wfi=Mary; 2 - posi=NNP; 3 - wfj=saw; 4 - posj=VBD; 5 - wfj=John; 6 - posj=NNP); (0 - link2, 1 - shift)
- Or Weka ARFF (Weka is a Machine Learning tool box)

Classification

- Instance: wfi=Mary posi=NNP wfj=saw posj=VBD ?
- Classes: c_1 – link(i,j), c_2 – link(j,i),c_3 – shift etc.
- Classification:
  - $sum(c_1)=d_1+w_{1,c1}+w_{2,c1}+w_{3,c1}w_{n,c1}$
  - $sum(c_2)=d_2+w_{1,c2}+w_{2,c2}+w_{n,c2}$
  - $sum(c_3)=d_3+w_{1,c3}+w_{2',c3}+w_{n,c3}$
- Highest $sum(c_j)$:
  - $max = max\{sum(c_1),sum(c_2),sum(c_3)\}$
- Probability of c_j:
  - $p(c_j)=exp(sum(c_j)-max)$
- Normalisation:
  - $p(c_j)=$ $$\frac{p(c_j)}{\sum \text{sum}\, p(c_j)}$$

Classification

- $sum(c_1)=1.323$, $sum(c_2)=-0.119$, $sum(c_3)=-1.204$
- The maximum is obviously $max=sum(c_1)=1.323$
- $p(c_1)=exp(sum(c_1)-max)=exp(0)=1$
- $p(c_2)=exp(sum(c_2)-max)=exp(-1.442)=0.236$
- $p(c_3)=exp(sum(c_3)-max)=exp(-2.527)=0.08$
- The sum of all $sum(c_j)$ is 1.316. Thus the normalised probability distribution is:
- $p(c_1)=\frac{1}{1.316}=0.76$
- $p(c_2)=\frac{0.236}{1.316}=0.18$
- $p(c_3)=\frac{0.08}{1.316}=0.06$

Summary

- Dependency Grammar and Parsing
- Graph-based parsing
- Transition-based approach
- Learning and Classification