

Dependency Parsing

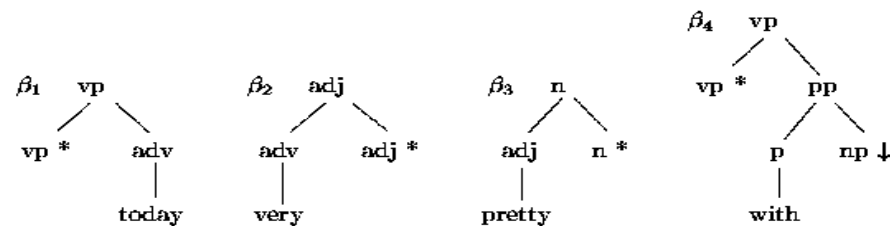
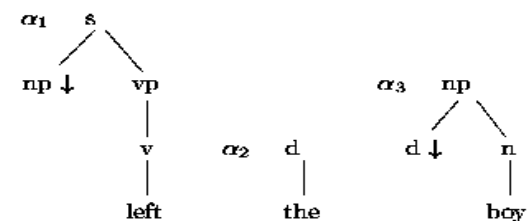
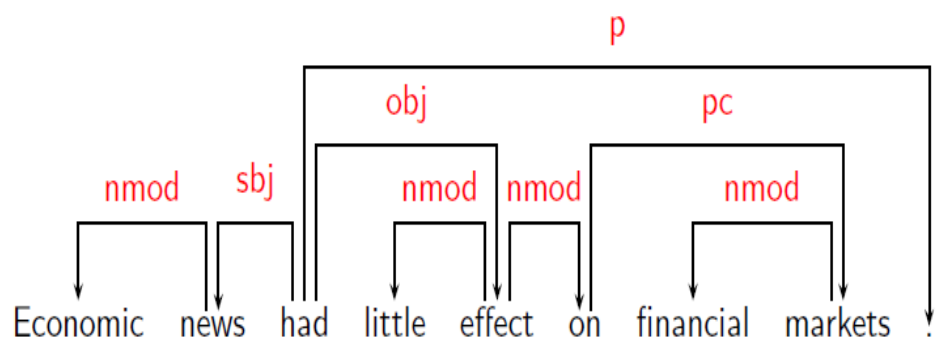
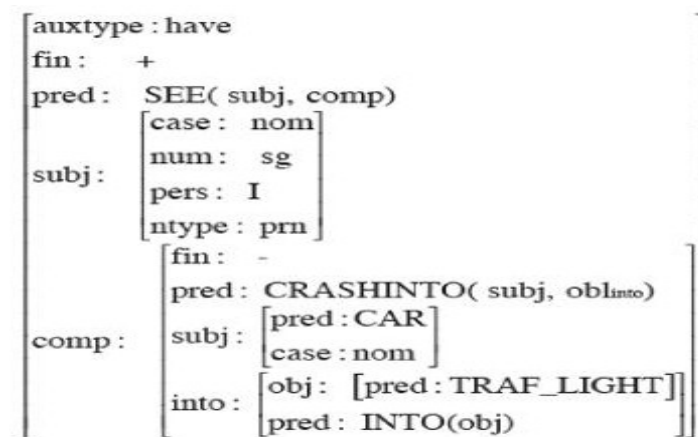
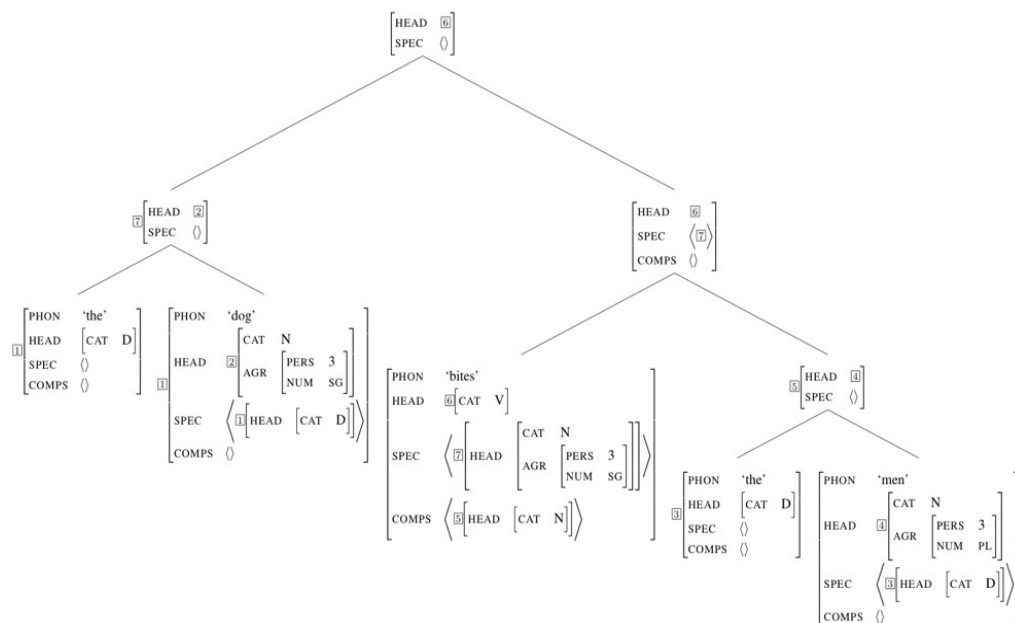
**Computational Linguistics
SS 2011**

Alexander Volokh
alexander.volokh@dfki.de

Overview

- Dependency Grammar vs Dependency Parsing
- Transition-Based vs Graph-Based Dependency Parsing
- Algorithms
- Learning
- Classification

Syntactic Theories



Dependency Representation

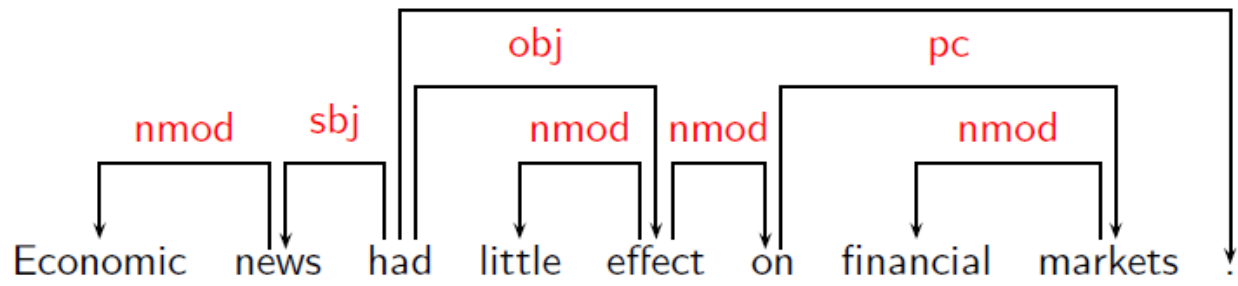
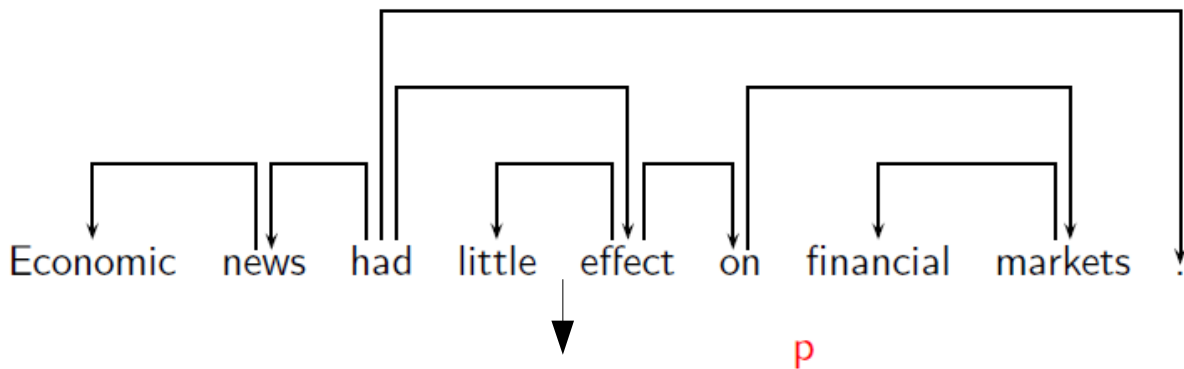
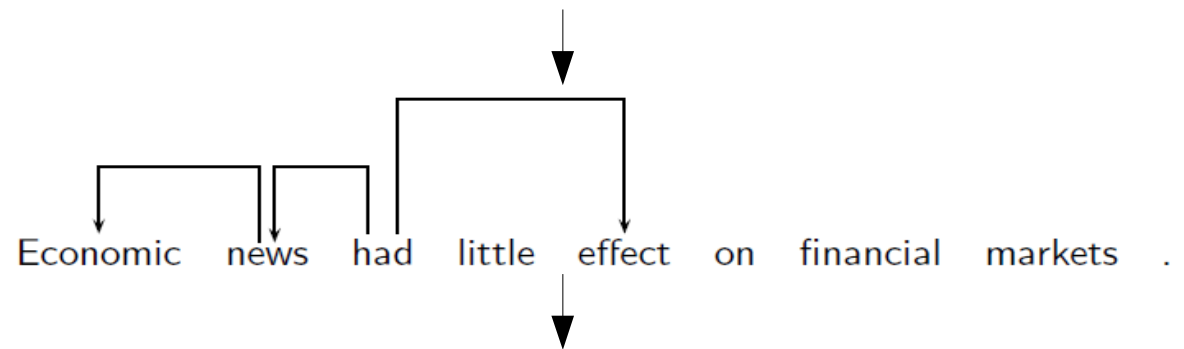
- The basic idea:
Syntactic structure consists of **lexical items**, linked by
binary, asymmetric, directed, anti-reflexive, anti-transitive, labeled
relations called **dependencies**.
- $A \rightarrow B; \langle B, A \rangle$
(A is head/parent/governor; B is dependent/child/subordinate)
- Syntactic structures are **usually** trees, i.e. they have the
following properties:
connectedness, single-headiness, rooted, acyclicity, projectivity

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)

Parsing

Economic news had little effect on financial markets .



Dependency Parsing

- Easy to implement
 - no artificial nodes
 - linear complexity possible
- 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
- Similar to human parsing
 - “He sang songs loudly“

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

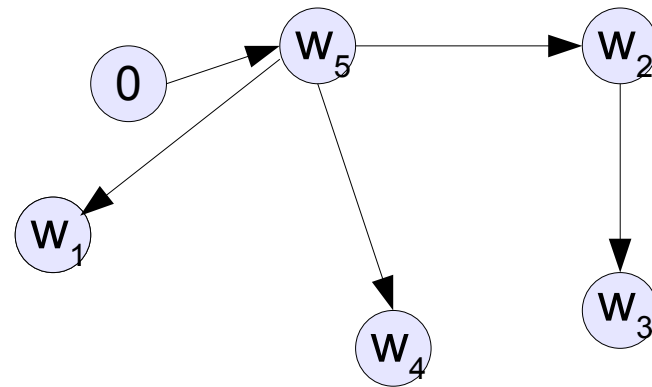
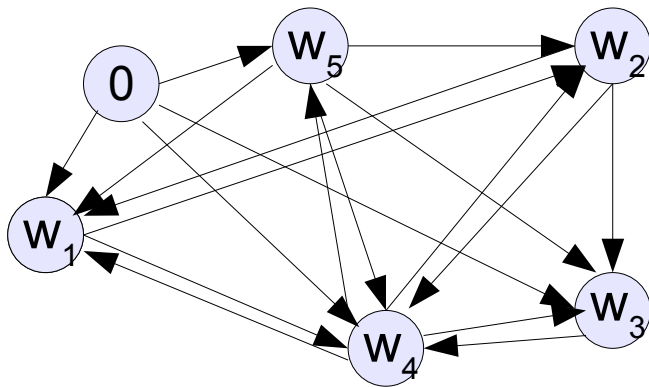
- Rule systems:
 - Lists of words for every category
 - Which categories occur with which categories
 - Valency
- Data-driven systems:
 - Use treebanks to learn how to link words
 - Dependency treebanks are available for many languages (CoNLL-X shared task)

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)

Graph-Based Parsing

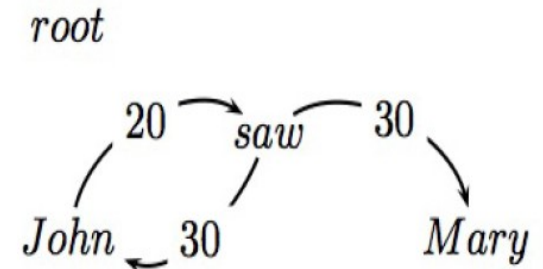
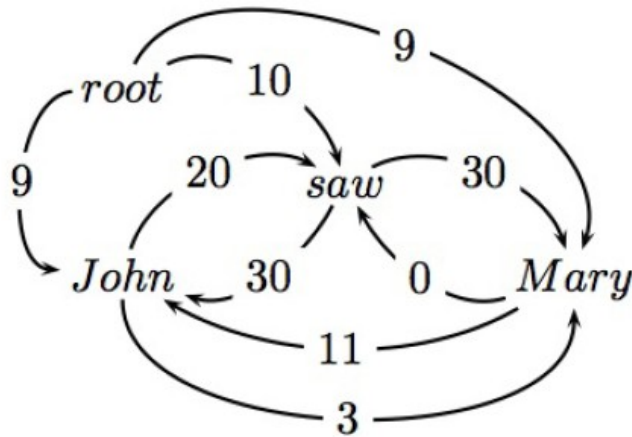
- Given the input $I = w_1, w_2, \dots, w_n$, where each word corresponds to a vertex v_1, v_2, \dots, v_n find a graph $G = (V, A)$, such that G is a rooted tree and $A = \{ \langle A_1, B_1 \rangle, \langle A_2, B_2 \rangle, \dots, \langle A_3, B_3 \rangle \}$ corresponds to the correct dependency tree
- Solution:
Maximum Spanning Trees



Chu-Liu-Edmonds

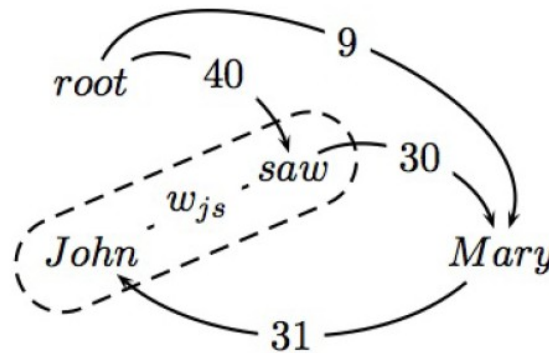
$x = \text{root John saw Mary}$

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



Graph-Based Parsing

- Advantages:
 - State-of-the art performance
 - Works well for long sentences/dependencies
- Disadvantages:
 - Not incremental
 - Computationally expensive
 - Difficult to implement

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 $I = w_1, w_2, \dots, w_n$ perform $S = c_0, c_1, \dots, c_n$, such that $A = \{ \langle A_1, B_1 \rangle, \langle A_2, B_2 \rangle, \dots, \langle A_n, B_n \rangle \}$ corresponds to the correct dependency tree

Configuration – state of the parser

- Define the set of possible transitions, e.g.:
- Conditions (permissibility):

$\text{left_link}(a,b)$ – b should not have a parent; if $\langle A,B \rangle$ is introduced to A , A should not contain a cycle etc.

- Effects:

$\text{left_link}(a,b) \rightarrow a$ becomes the parent of b

$\text{right_link}(a,b) \rightarrow b$ becomes the parent of a

$\text{shift}(a,b) \rightarrow$ move on to next pair

- Initial configuration / terminal configuration

Parsing Algorithms

- Naive:
 - For every word j in the sentence try to combine it with other words i in the sentence:
 - 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

Naive Algorithm

c_0 : $j=1; i=0, A = \{\}$

$c_0 \rightarrow c_1$: do not combine; $i+1$

$(j=1, i=1, A = \{\})$

$c_{12} \rightarrow c_{13}$: make j the part of i

$(j=2, i=4, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_1 \rightarrow c_2$: do not combine; $i+1$

$(j=1, i=2, A = \{\})$

$c_{13} \rightarrow c_{14}$: do not combine; $j+1$

$(j=3, i=0, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_2 \rightarrow c_3$: make i the parent of j ;

$(j=1, i=2, A = \{<1,2>\})$

$c_{14} \rightarrow c_{15}$: do not combine; $i+1$

$(j=3, i=1, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_3 \rightarrow c_4$: do not combine; $i+1$

$(j=1, i=3, A = \{<1,2>\})$

$c_{15} \rightarrow c_{16}$: do not combine; $i+1$

$(j=3, i=2, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_4 \rightarrow c_5$: do not combine; $i+1$

$(j=1, i=4, A = \{<1,2>\})$

$c_{16} \rightarrow c_{17}$: do not combine; $i+1$

$(j=3, i=3, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_5 \rightarrow c_6$: do not combine; $j+1$

$(j=2, i=0, A = \{<1,2>\})$

$c_{17} \rightarrow c_{18}$: do not combine; $i+1$

$(j=3, i=4, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_6 \rightarrow c_7$: make i the parent of j

$(j=2, i=0, A = \{<1,2>, <2,0>\})$

$c_{18} \rightarrow c_{19}$: do not combine; $j+1$

$(j=4, i=0, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_7 \rightarrow c_8$: do not combine; $i+1$

$(j=2, i=1, A = \{<1,2>, <2,0>\})$

$c_{19} \rightarrow c_{20}$: do not combine; $i+1$

$(j=4, i=1, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_8 \rightarrow c_9$: do not combine; $i+1$

$(j=2, i=2, A = \{<1,2>, <2,0>\})$

$c_{20} \rightarrow c_{21}$: do not combine; $i+1$

$(j=4, i=2, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_9 \rightarrow c_{10}$: do not combine; $i+1$

$(j=2, i=3, A = \{<1,2>, <2,0>\})$

$c_{21} \rightarrow c_{22}$: do not combine; $i+1$

$(j=4, i=3, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_{10} \rightarrow c_{11}$: make j the part of i

$(j=2, i=3, A = \{<1,2>, <2,0>, <3,2>\})$

$c_{22} \rightarrow c_{23}$: do not combine; $i+1$

$(j=4, i=4, A = \{<1,2>, <2,0>, <3,2>, <4,2>\})$

$c_{11} \rightarrow c_{12}$: do not combine; $i+1$

$(j=2, i=4, A = \{<1,2>, <2,0>, <3,2>\})$ c_{23} : terminal configuration

Naive Algorithm

- Obvious disadvantages:
 - Too many senseless configurations
 - $O(n^2)$ runtime
 - Not incremental
- Advantages:
 - Deterministic
 - 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

- Sample configuration: ($j=2$, $i=3$, $A = \{<1,2>, <2,0>\}$)
- Feature templates: $wf(x)$, $pos(x)$, $dist(x,y)$, $isRoot(x)$
- 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

Learning

- Print all feature vectors into a file in 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

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

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:
- $\text{sum}(c_1) = d_1 + w_{1,c1} + w_{2,c1} + w_{3,c1} + \dots + w_{n,c1}$
- $\text{sum}(c_2) = d_2 + w_{1,c2} + w_{2,c2} + \dots + w_{n,c2}$
- $\text{sum}(c_3) = d_3 + w_{1,c3} + w_{2,c3} + \dots + w_{n,c3}$
- Biggest $\text{sum}(c_j)$: $\max = \max\{\text{sum}(c_1), \text{sum}(c_2), \text{sum}(c_3)\}$
- Probability of c_j : $p(c_j) = \frac{\exp(\text{sum}(c_j) - \max)}{\sum \exp(\text{sum}(c_j) - \max)}$
- Normalisation: $p(c_j) = \frac{\exp(\text{sum}(c_j) - \max)}{\sum \exp(\text{sum}(c_j) - \max)}$

Classification

- $\text{sum}(c_1)=1.323$, $\text{sum}(c_2)=-0.119$, $\text{sum}(c_3)=-1.204$
- The maximum is obviously $\text{max}=\text{sum}(c_1)=1.323$
- $p(c_1)=\exp(\text{sum}(c_1)-\text{max})=\exp(0)=1$
- $p(c_2)=\exp(\text{sum}(c_2)-\text{max})=\exp(-1.442)=0.236$
- $p(c_3)=\exp(\text{sum}(c_3)-\text{max})=\exp(-2.527)=0.08$
- The sum of all $\text{sum}(c_j)$ 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
- Transition-based approach
 - Idea
 - Algorithms (next lecture)
- Learning and Classification
 - Idea
 - Approaches and Data (next lecture)

Exercise

- Will appear on the course page very soon