Graph-Based and Transition-Based Dependency Parsing

Joakim Nivre

Uppsala University
Linguistics and Philology

Based on previous tutorials with Ryan McDonald

Overall Plan

1. Basic notions of dependency grammar and dependency parsing
2. Graph-based and transition-based dependency parsing
3. Advanced graph-based parsing techniques
4. Advanced transition-based parsing techniques
5. Neural network techniques in dependency parsing
6. Multilingual parsing from raw text to universal dependencies

Plan for this Lecture

- Graph-based dependency parsing
  - First-order model
  - Learning and inference
- Transition-based dependency parsing
  - Arc-eager transition system
  - Learning and inference
- Contrastive error analysis [McDonald and Nivre 2007]

Graph-Based Parsing

- Basic idea:
  - Define a space of candidate dependency graphs for a sentence.
  - Learning: Induce a model for scoring an entire dependency graph for a sentence.
  - Parsing: Find the highest-scoring dependency graph, given the induced model.
- Characteristics:
  - Global training of a model for optimal dependency graphs
  - Exhaustive search/inference

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \,|\, i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Key observation:
  - Valid dependency trees for x = directed spanning trees of G_x

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Key observation:
  - Valid dependency trees for x = directed spanning trees of G_x
- Score of dependency tree T factors by subgraphs $G_1, \ldots, G_m$:
  - $s(T) = \sum_{c=1}^{m} s(G_c)$

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Key observation:
  - Valid dependency trees for x = directed spanning trees of G_x
- Score of dependency tree T factors by subgraphs $G_1, \ldots, G_m$:
  - $s(T) = \sum_{c=1}^{m} s(G_c)$
- Learning: Scoring function $s(G_c)$ for subgraphs $G_c \in G$

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Key observation:
  - Valid dependency trees for x = directed spanning trees of G_x
- Score of dependency tree T factors by subgraphs $G_1, \ldots, G_m$:
  - $s(T) = \sum_{c=1}^{m} s(G_c)$
- Learning: Scoring function $s(G_c)$ for subgraphs $G_c \in G$
- Inference: Search for maximum spanning tree T^* of G_x

$$T^* = \operatorname*{argmax}_{T \in G_x} s(T) = \operatorname*{argmax}_{T \in G_x} \sum_{c=1}^{m} s(G_c)$$

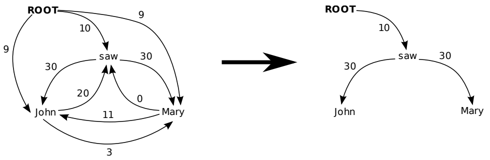
Learning

- Typical scoring function:
  - $s(G_i) = \mathbf{w} \cdot \mathbf{f}(G_i)$

  where
  - $\mathbf{f}(G_i)$ = high-dimensional feature vector over subgraphs
  - $\mathbf{w}$ = weight vector [$\mathbf{w}_j$ = weight of feature $\mathbf{f}_j(G_i)$]
- Structured learning [McDonald et al. 2005a]:
  - Learn weights that maximize the score of the correct dependency tree for every sentence in the training set

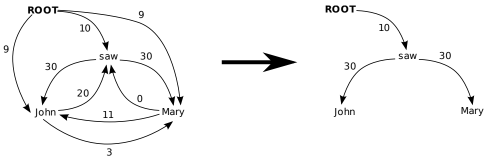
First-Order Model

- Scored subgraph G_c is a single arc (i, j, k)
- $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{(i,j,k) \in T} s(i, j, k)$
- Often we drop k, since it is rarely structurally relevant
  - $s(T) = \sum_{(i,j) \in T} s(i, j)$
  - $s(i, j) = max_k s(i, j, k)$



First-Order Model

- Scored subgraph G_c is a single arc (i, j, k)
- $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{(i,j,k)\in T} s(i, j, k)$
- Often we drop k, since it is rarely structurally relevant
  - $s(T) = \sum_{(i,j)\in T} s(i, j)$
  - $s(i, j) = max_k s(i, j, k)$



- This search is global: consider all possible trees

First-Order Projective Parsing

Eisner algorithm

[Eisner 1996]

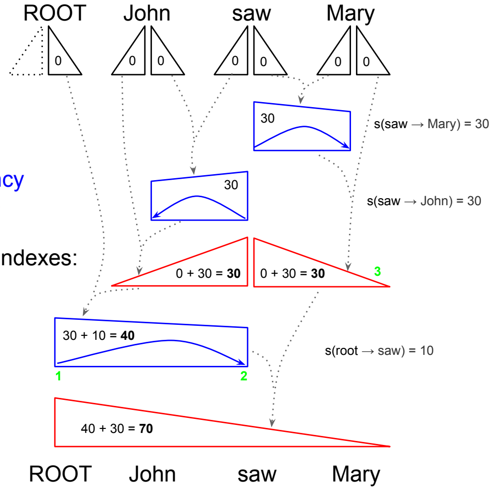


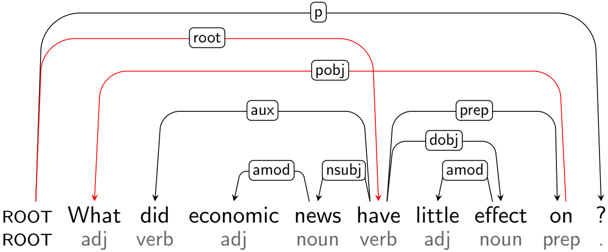
Chart items either:
1) Create a new dependency
2) Absorb left/right subtree

Each chart item store two indexes:
1) left boundary
2) right boundary

All operations require
3 indexes: **O(n^3)**

First-Order Non-Projective Parsing

- Equivalent to MST problem [McDonald et al. 2005b]
- For directed graphs, also called arboresence problem
- $O(n^2)$ parsing [Chu and Liu 1965, Edmonds 1967]
- Greedy algorithm, not dynamic programming



Feature Scope

- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c

Feature Scope

- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- For first-order models, G_c is an arc
  - $G_c = (i, j)$ for a head i and modifier j

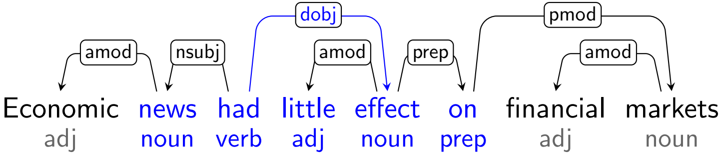
Feature Scope

- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- For first-order models, G_c is an arc
  - $G_c = (i, j)$ for a head i and modifier j
- This inherently limits features to a local scope

Feature Scope

- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- For first-order models, G_c is an arc
  - $G_c = (i, j)$ for a head i and modifier j
- This inherently limits features to a local scope

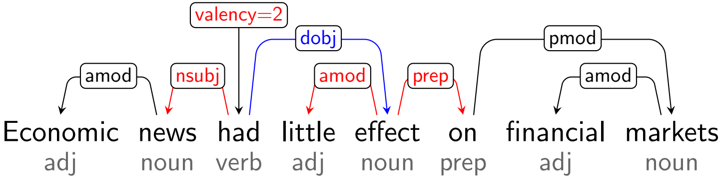
- For arc (had, effect) below, can have features over properties of arc and context within sentence



Feature Scope

- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- For first-order models, G_c is an arc
  - $G_c = (i, j)$ for a head i and modifier j
- This inherently limits features to a local scope

- For arc (had, effect) below, cannot have features over multiple arcs (siblings, grandparents), valency, etc.

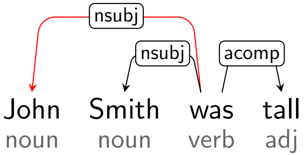


Graph-Based Parsing Trade-Off

- Learning and inference are global
  - Decoding guaranteed to find highest scoring tree
  - Training algorithms use global structure learning

Graph-Based Parsing Trade-Off

- Learning and inference are global
  - Decoding guaranteed to find highest scoring tree
  - Training algorithms use global structure learning
- But this is only possible with local feature factorizations
  - Must limit context statistical model can look at
  - Results in bad 'easy' decisions
    - For example, first-order models often predict two subjects
    - No parameter exists to discourage this



Transition-Based Parsing

- Basic idea:
  - Define a transition system (state machine) for mapping a sentence to its dependency graph.
  - Learning: Induce a model for predicting the next state transition, given the transition history.
  - Parsing: Construct the optimal transition sequence, given the induced model.
- Characteristics:
  - Local training of a model for optimal transitions
  - Greedy search/inference

Transition-Based Parsing

- A transition system for dependency parsing defines
  - a set C of parser configurations
  - a set T of transitions, each a function $t : C \rightarrow C$
  - initial configuration and terminal configurations for sentence x
- Key idea:
  - Valid dependency trees for S defined by terminating transition sequences $C_{0,m} = t_1(c_0), \ldots, t_m(c_{m-1})$
- Score of $C_{0,m}$ factors by config-transition pairs (c_{i-1}, t_i):
  - $s(C_{0,m}) = \sum_{i=1}^{m} s(c_{i-1}, t_i)$
- Learning:
  - Scoring function $s(c_{i-1}, t_i)$ for $t_i(c_{i-1}) \in C_{0,m}$
- Inference:
  - Search for highest scoring sequence $C^*_{0,m}$ given $s(c_{i-1}, t_i)$

Arc-Eager Transition System [Nivre 2003]

Configuration: (S, B, A) [S = Stack, B = Buffer, A = Arcs]

Initial: $([\,], [0, 1, \ldots, n], \{\,\})$

Terminal: $(S, [\,], A)$

Shift:	$(S, i\vert B, A)$	$\Rightarrow$	$(S\vert i, B, A)$	
Reduce:	$(S\vert i, B, A)$	$\Rightarrow$	(S, B, A)	$h(i, A)$
Right-Arc(k):	$(S\vert i, j\vert B, A)$	$\Rightarrow$	$(S\vert i\vert j, B, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\vert i, j\vert B, A)$	$\Rightarrow$	$(S, j\vert B, A \cup \{(j, i, k)\})$	$\neg h(i, A) \wedge i \neq 0$

Notation: $S|i$ = stack with top i and remainder S
$j|B$ = buffer with head j and remainder B
$h(i, A)$ = i has a head in A

Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{Economic, news, had, little, effect, on, financial, markets, .}]_B$

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

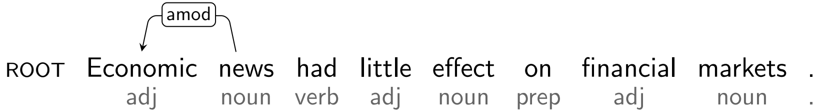
Example Transition Sequence

$[\text{ROOT, Economic}]_S$ $[\text{news, had, little, effect, on, financial, markets, .}]_B$

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

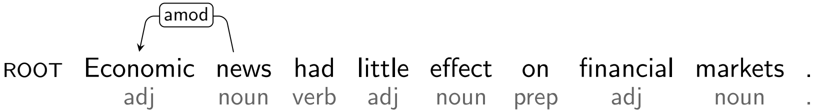
Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{news, had, little, effect, on, financial, markets, .}]_B$



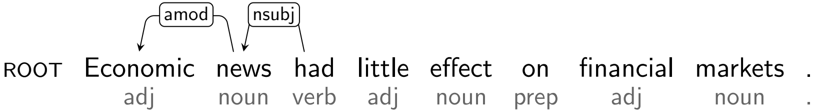
Example Transition Sequence

$[\text{ROOT, news}]_S$ $[\text{had, little, effect, on, financial, markets, .}]_B$



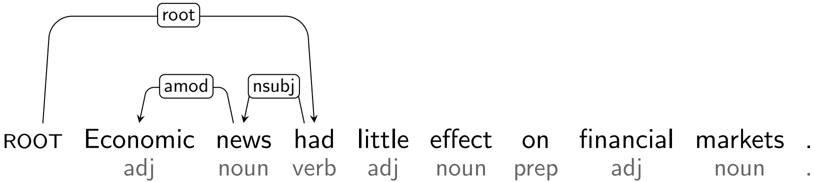
Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{had, little, effect, on, financial, markets, .}]_B$



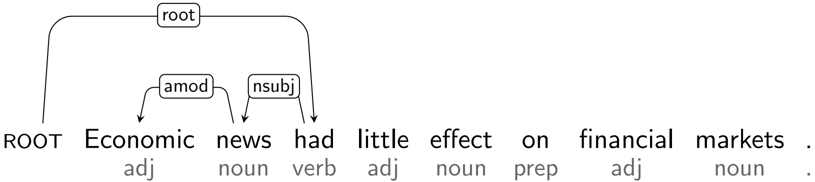
Example Transition Sequence

$[\text{ROOT, had}]_S$ $[\text{little, effect, on, financial, markets, .}]_B$



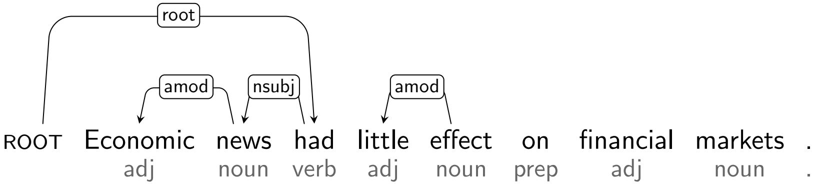
Example Transition Sequence

$[$ROOT, had, little$]_S$ $[$effect, on, financial, markets, .$]_B$



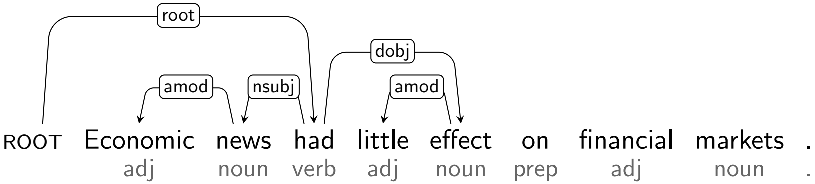
Example Transition Sequence

$[\text{ROOT, had}]_S$ $[\text{effect, on, financial, markets, .}]_B$



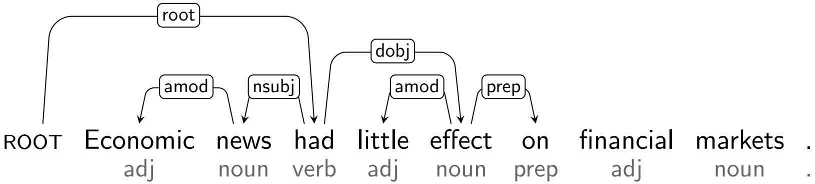
Example Transition Sequence

[ROOT, had, effect]$_S$ [on, financial, markets, .]$_B$



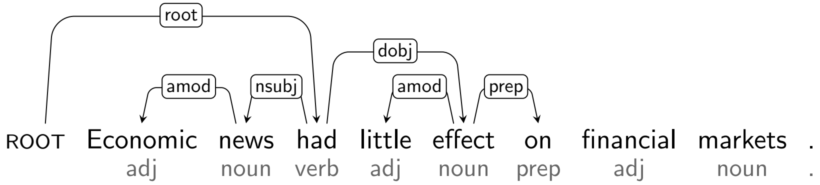
Example Transition Sequence

$[\text{ROOT, had, effect, on}]_S$ $[\text{financial, markets, .}]_B$



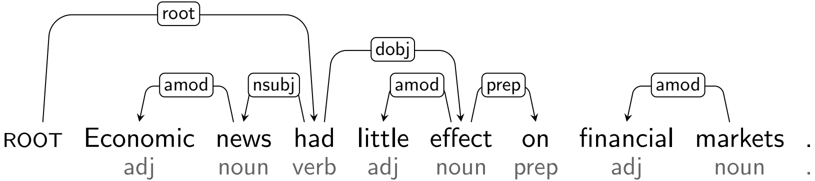
Example Transition Sequence

$[\text{ROOT, had, effect, on, financial}]_S$ $[\text{markets, .}]_B$



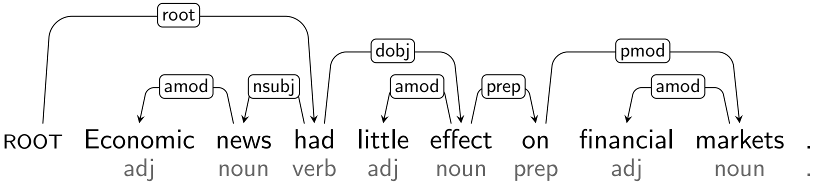
Example Transition Sequence

$[\text{ROOT, had, effect, on}]_S$ $[\text{markets, .}]_B$



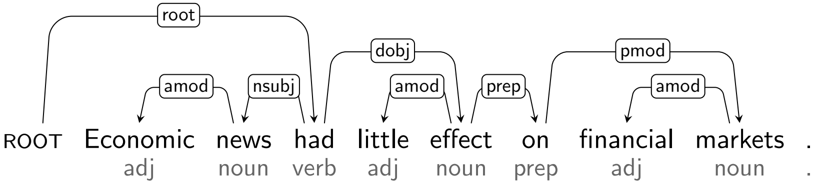
Example Transition Sequence

[ROOT, had, effect, on, markets]$_S$ [.]$_B$



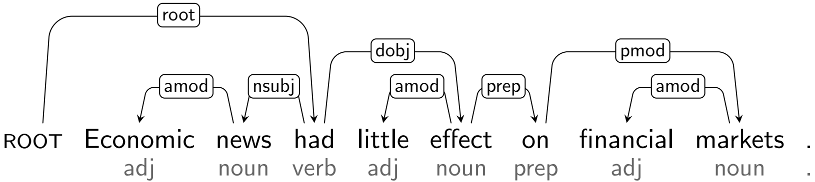
Example Transition Sequence

$[\text{ROOT, had, effect, on}]_S \quad [.]_B$



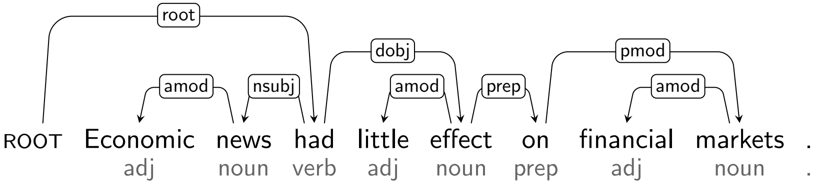
Example Transition Sequence

$[\text{ROOT, had, effect}]_S \quad [.]_B$



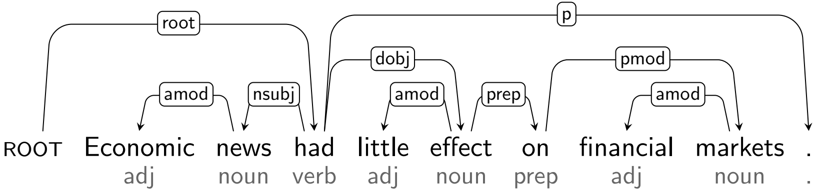
Example Transition Sequence

$[\text{ROOT, had}]_S \quad [.]_B$



Example Transition Sequence

$[$ROOT, had, .$]_S$ $[\ ]_B$



Greedy Inference

- Given an oracle o that correctly predicts the next transition $o(c)$, parsing is deterministic:

$$
\begin{array}{l}
\text{Parse}(w_1, \ldots, w_n) \\
\quad c \leftarrow ([\ ]_S, [0, 1, \ldots, n]_B, \{\ \}) \\
\quad \textbf{while } B_c \neq [\ ] \\
\qquad t \leftarrow o(c) \\
\qquad c \leftarrow t(c) \\
\quad \textbf{return } G = (\{0, 1, \ldots, n\}, A_c)
\end{array}
$$

- Complexity given by upper bound on number of transitions
- Parsing in $O(n)$ time for the arc-eager transition system

From Oracles to Classifiers

- An oracle can be approximated by a (linear) classifier:

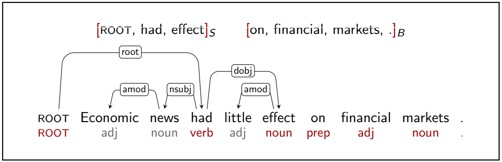
$$o(c) = \underset{t}{\operatorname{argmax}}\, \mathbf{w} \cdot \mathbf{f}(c, t)$$

- History-based feature representation $\mathbf{f}(c, t)$
- Weight vector $\mathbf{w}$ learned from treebank data

Feature Representation

- Features over input tokens relative to S and B

Configuration



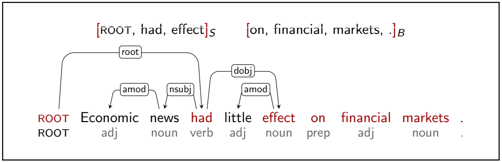
Features

$\text{pos}(S_2)$	=	ROOT
$\text{pos}(S_1)$	=	verb
$\text{pos}(S_0)$	=	noun
$\text{pos}(B_0)$	=	prep
$\text{pos}(B_1)$	=	adj
$\text{pos}(B_2)$	=	noun

Feature Representation

- Features over input tokens relative to S and B

Configuration



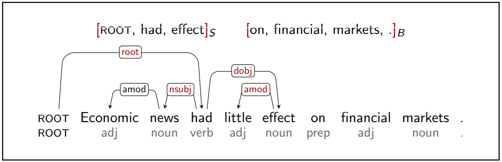
Features

$$
\begin{aligned}
\text{word}(S_2) &= \text{ROOT} \\
\text{word}(S_1) &= \text{had} \\
\text{word}(S_0) &= \text{effect} \\
\text{word}(B_0) &= \text{on} \\
\text{word}(B_1) &= \text{financial} \\
\text{word}(B_2) &= \text{markets}
\end{aligned}
$$

Feature Representation

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A

Configuration



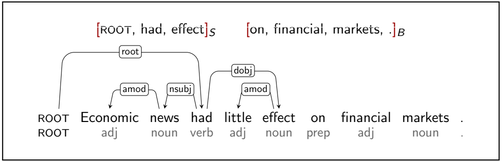
Features

$\text{dep}(S_1) = \text{root}$
$\text{dep}(\text{lc}(S_1)) = \text{nsubj}$
$\text{dep}(\text{rc}(S_1)) = \text{dobj}$
$\text{dep}(S_0) = \text{dobj}$
$\text{dep}(\text{lc}(S_0)) = \text{amod}$
$\text{dep}(\text{rc}(S_0)) = \text{NIL}$

Feature Representation

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A
- Features over the (partial) transition sequence

Configuration



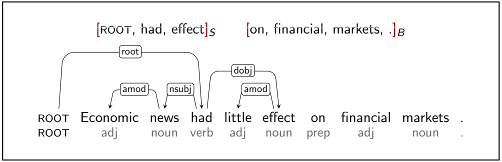
Features

t_{i-1} = Right-Arc(dobj)
t_{i-2} = Left-Arc(amod)
t_{i-3} = Shift
t_{i-4} = Right-Arc(root)
t_{i-5} = Left-Arc(nsubj)
t_{i-6} = Shift

Feature Representation

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A
- Features over the (partial) transition sequence

Configuration



Features

$$
\begin{aligned}
t_{i-1} &= \text{Right-Arc(dobj)} \\
t_{i-2} &= \text{Left-Arc(amod)} \\
t_{i-3} &= \text{Shift} \\
t_{i-4} &= \text{Right-Arc(root)} \\
t_{i-5} &= \text{Left-Arc(nsubj)} \\
t_{i-6} &= \text{Shift}
\end{aligned}
$$

- Feature representation unconstrained by parsing algorithm

Local Learning

- Given a treebank:
  - Reconstruct oracle transition sequence for each sentence
  - Construct training data set $D = \{(c, t) \mid o(c) = t\}$
  - Maximize accuracy of local predictions $o(c) = t$
- Any (unstructured) classifier will do (SVMs are popular)
- Training is local and restricted to oracle configurations

Transition-Based Parsing Trade-Off

- Advantages:
  - Highly efficient parsing – linear time complexity with constant time oracles and transitions
  - Rich history-based feature representations – no rigid constraints from inference algorithm
- Drawback:
  - Sensitive to search errors and error propagation due to greedy inference and local learning

CoNLL 2006

- CoNLL 2006: Shared Task on Dependency Parsing
  - Evaluation of 13 different languages
- Top 2 systems statistically identical: One graph-based (MSTParser) and the other transition-based (MaltParser)
- Question: do the systems learn the same things?

MSTParser and MaltParser

	MSTParser	MaltParser
Arabic	66.91	66.71
Bulgarian	87.57	87.41
Chinese	85.90	86.92
Czech	80.18	78.42
Danish	84.79	84.77
Dutch	79.19	78.59
German	87.34	85.82
Japanese	90.71	91.65
Portuguese	86.82	87.60
Slovene	73.44	70.30
Spanish	82.25	81.29
Swedish	82.55	84.58
Turkish	63.19	65.68
Overall	80.83	80.75

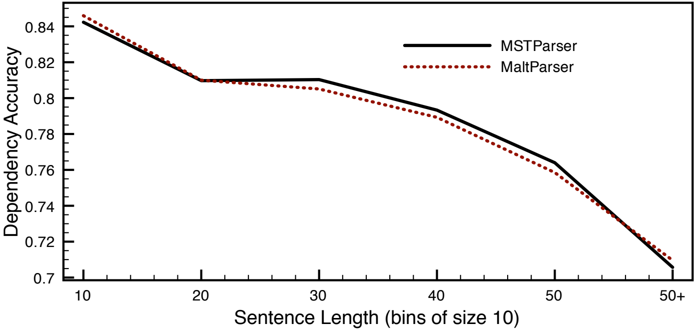
Comparing the Models

- **Inference**:
  - Exhaustive (MSTParser)
  - Greedy (MaltParser)
- **Training**:
  - Global structure learning (MSTParser)
  - Local decision learning (MaltParser)
- **Features**:
  - Local features (MSTParser)
  - Rich decision history (MaltParser)
- **Fundamental trade-off:**
  - Global learning and inference vs. rich feature space

Error Analysis [McDonald and Nivre 2007]

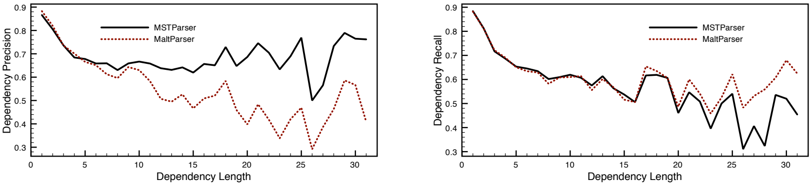
- Aim:
  - Relate parsing errors to linguistic and structural properties of the input and predicted/gold standard dependency graphs
- Three types of factors:
  - Length factors: sentence length, dependency length
  - Graph factors: tree depth, branching factor, non-projectivity
  - Linguistic factors: part of speech, dependency type
- Statistics:
  - Labeled accuracy, precision and recall
  - Computed over the test sets for all 13 languages

Sentence Length



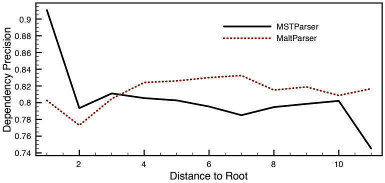
- MaltParser is more accurate than MSTParser for short sentences (1–10 words) but its performance degrades more with increasing sentence length.

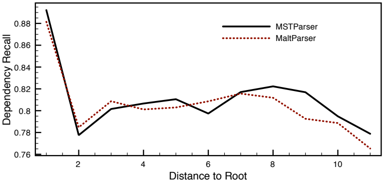
Dependency Length



- MaltParser is more precise than MSTParser for short dependencies (1–3 words) but its performance degrades drastically with increasing dependency length ($>$ 10 words).
- MSTParser has more or less constant precision for dependencies longer than 3 words.
- Recall is very similar across systems.

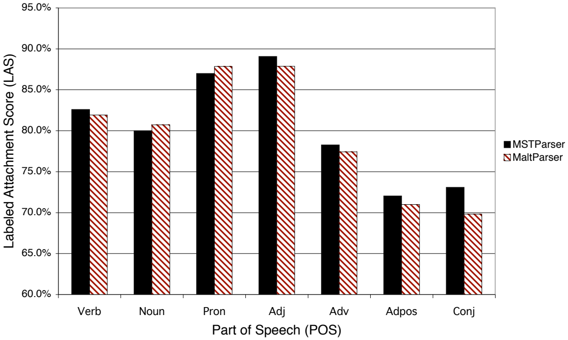
Tree Depth (Distance to Root)





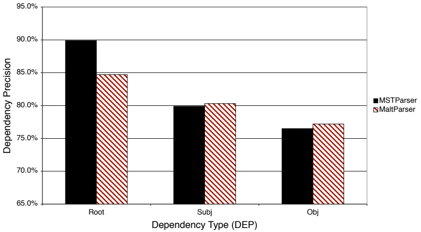
- MSTParser is much more precise than MaltParser for dependents of the root and has roughly constant precision for depth > 1, while MaltParser's precision improves with increasing depth (up to 7 arcs).
- Recall is very similar across systems.

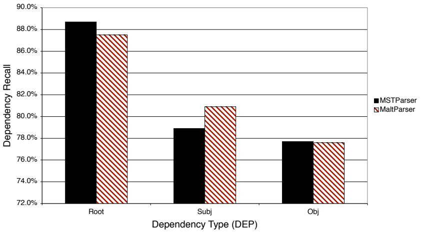
Part of Speech



- MSTParser is more accurate for verbs, adjectives, adverbs, adpositions, and conjunctions.
- MaltParser is more accurate for nouns and pronouns.

Dependency Type: Root, Subject, Object





- MSTParser has higher precision (and recall) for roots.
- MSTParser has higher recall (and precision) for subjects.

Discussion

- Many of the results are indicative of the fundamental trade-off: global learning/inference versus rich features.
- Global inference improves decisions for long sentences and those near the top of graphs.
- Rich features improve decisions for short sentences and those near the leaves of the graphs.
- Dependency parsing post-2007:
  - How do we use this to improve parser performance?

Voting and Stacking

- Early improvements were based on system combination
- Voting:
  - Let parsers vote for heads [Zeman and Žabokrtský 2005]
  - Use MST algorithm for tree constraint [Sagae and Lavie 2006]
- Stacking:
  - Use the output of one parser as features for the other [Nivre and McDonald 2008, Torres Martins et al. 2008]
- Focus in these lectures:
  - Work on evolving the approaches themselves
  - Richer feature representations in graph-based parsing
  - Improved learning and inference in transition-based parsing

Coming Up Next

1. Basic notions of dependency grammar and dependency parsing
2. Graph-based and transition-based dependency parsing
3. Advanced graph-based parsing techniques
4. Advanced transition-based parsing techniques
5. Neural network techniques in dependency parsing
6. Multilingual parsing from raw text to universal dependencies

References and Further Reading

- Giuseppe Attardi. 2006.
  Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.

- Y. J. Chu and T. J. Liu. 1965.
  On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

- Michael A. Covington. 2001.
  A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

- J. Edmonds. 1967.
  Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

- Jason M. Eisner. 1996.
  Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

- Ryan McDonald and Joakim Nivre. 2007.

Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the Join Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning (EMNLP-CoNLL)*.

- Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. 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 (HLT/EMNLP)*, pages 523–530.
- Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 950–958.
- Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In Hwee Tou Ng and Ellen Riloff, editors, *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56.

- Joakim Nivre. 2003.
  An efficient algorithm for projective dependency parsing. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.
- Joakim Nivre. 2007.
  Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 396–403.
- Joakim Nivre. 2008.
  Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.
- Kenji Sagae and Alon Lavie. 2006.
  Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132.
- André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008.
  Stacking dependency parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 157–166.

- Hiroyasu Yamada and Yuji Matsumoto. 2003.
  Statistical dependency analysis with support vector machines. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

- Daniel Zeman and Zdeněk Žabokrtský. 2005.
  Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 171–178.