Graph-Based and Transition-Based Dependency Parsing

Joakim Nivre

Uppsala University
Linguistics and Philology

Based on previous tutorials with Ryan McDonald

Overview of the Course

1. Introduction to dependency grammar and dependency parsing
2. Graph-based and transition-based dependency parsing
3. Multiword expressions in dependency parsing
4. Practical lab session (MaltParser)

Plan for this Lecture

- Graph-based parsing:
  - Basic concepts
  - Projective parsing
  - Non-projective parsing
- Transition-based parsing
  - Basic concepts
  - Beam search and structured prediction
  - Non-projective parsing
  - Joint morphological and syntactic analysis
- Conclusion and outlook

Graph-Based Parsing

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

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Valid dependency trees for x equivalent to directed spanning trees T of G_x rooted at w_0

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Valid dependency trees for x equivalent to directed spanning trees T of G_x rooted at w_0
- Score of dependency tree T factors by subgraphs $G_1, \ldots, G_m$:
  - $s(T) = \sum_{c=1}^{m} s(G_c)$
  - Each G_c need not be a subtree

Graph-Based Parsing

- For input sentence x define a graph $G_x = (V_x, A_x)$, where
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Valid dependency trees for x equivalent to directed spanning trees T of G_x rooted at w_0
- Score of dependency tree T factors by subgraphs $G_1, \ldots, G_m$:
  - $s(T) = \sum_{c=1}^{m} s(G_c)$
  - Each G_c need not be a subtree
- 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
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq 0 \text{ and } i \neq j \text{ and } l_k \in L\}$
- Valid dependency trees for x equivalent to directed spanning trees T of G_x rooted at w_0
- Score of dependency tree T factors by subgraphs $G_1, \ldots, G_m$:
  - $s(T) = \sum_{c=1}^{m} s(G_c)$
  - Each G_c need not be a subtree
- 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

- We will assume scoring function is a linear classifier
  - $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{c=1}^{m} \mathbf{w} \cdot \mathbf{f}(G_c)$

Learning

- We will assume scoring function is a linear classifier
  - $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{c=1}^{m} \mathbf{w} \cdot \mathbf{f}(G_c)$
- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c

Learning

- We will assume scoring function is a linear classifier
  - $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{c=1}^{m} \mathbf{w} \cdot \mathbf{f}(G_c)$
- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- $\mathbf{w} \in \mathbb{R}^n$ is a corresponding weight vector

Learning

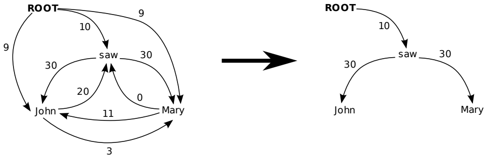
- We will assume scoring function is a linear classifier
  - $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{c=1}^{m} \mathbf{w} \cdot \mathbf{f}(G_c)$
- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- $\mathbf{w} \in \mathbb{R}^n$ is a corresponding weight vector

- We will assume that learning is solved
  - Linear scoring plus inference allows us to use Perceptron, MIRA, etc. to find suitable $\mathbf{w}$

Parameterizing Graph-Based Parsing

First-order (arc-factored) model

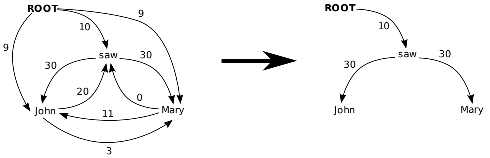
- Scored subgraph G_c is a single arc (i, j, k)
- $s(T) = \sum_{c=1}^{m} s(G_c) = \sum_{(i,j,k) \in T} s(i, j, k)$
- Often we drop k, since it is rarely structurally relevant
  - $s(T) = \sum_{(i,j) \in T} s(i, j)$
  - $s(i, j) = max_k s(i, j, k)$



Parameterizing Graph-Based Parsing

First-order (arc-factored) 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]

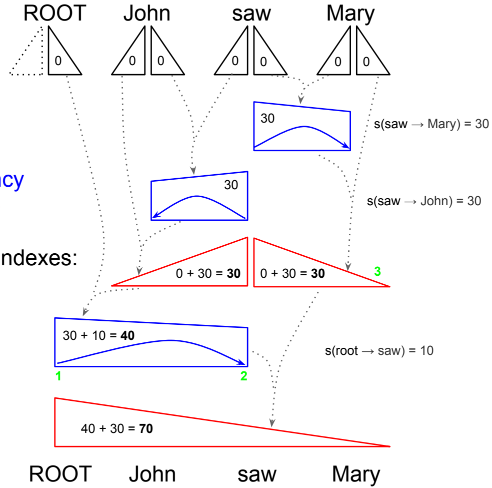


Chart items either:

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)**

Feature Scope

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

Feature Scope

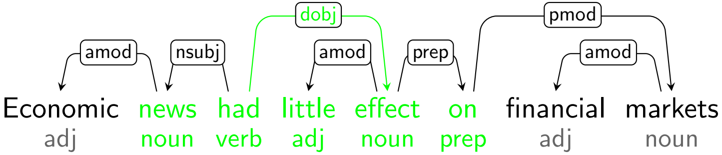
- $\mathbf{f} \in \mathbb{R}^n$ is a feature representation of the subgraph G_c
- For first-order models, G_c is an arc
  - I.e., $G_c = (i, j)$ for a head i and modifier j

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
  - I.e., $G_c = (i, j)$ for a head i and modifier j
- This inherently limits features to a local scope

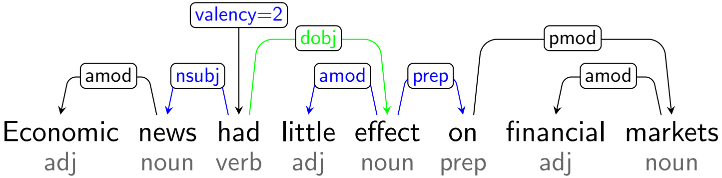
Feature Scope

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

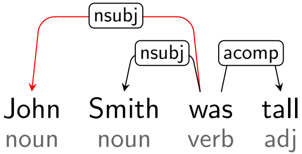
[McDonald and Nivre 2007]

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

Graph-Based Parsing Trade-Off

[McDonald and Nivre 2007]

- 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
    - E.g., First-order models often predict two subjects
    - No parameter exists to discourage this



Graph-Based Parsing Trade-Off

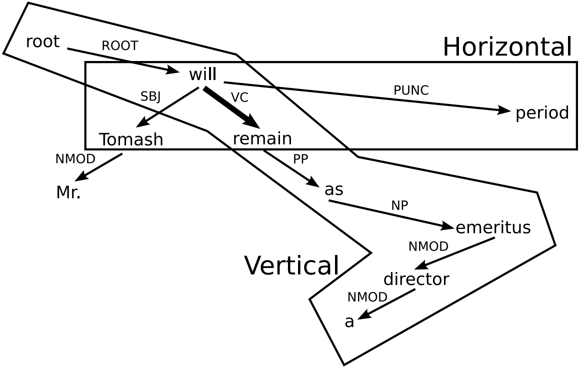
[McDonald and Nivre 2007]

- 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
    - E.g., First-order models often predict two subjects
    - No parameter exists to discourage this

The major question in graph-based parsing in recent years has been how to increase scope of features to larger subgraphs, without making inference intractable.

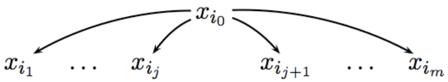
Higher-Order Parsing

- Two main dimensions of higher-order features
  - Vertical: e.g., “remain” is the grandparent of “emeritus”
  - Horizontal: e.g., “remain” is first child of “will”



2nd-Order Horizontal Projective Parsing

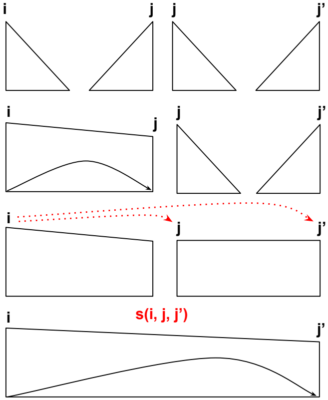
- Score factors by pairs of horizontally adjacent arcs
- Often called sibling dependencies
- $s(i,j,j')$ is the score of creating adjacent arcs $x_i \to x_j$ and $x_i \to x_{j'}$



$$
\begin{aligned}
s(T) &= \sum_{(i,j):(i,j')\in A} s(i,j,j') \\
&= \ldots + s(i_0,i_1,i_2) + s(i_0,i_2,i_3) + \ldots + s(i_0,i_{j-1},i_j) + \\
&\quad s(i_0,i_{j+1},i_{j+2}) + \ldots + s(i_0,i_{m-1},i_m) + \ldots
\end{aligned}
$$

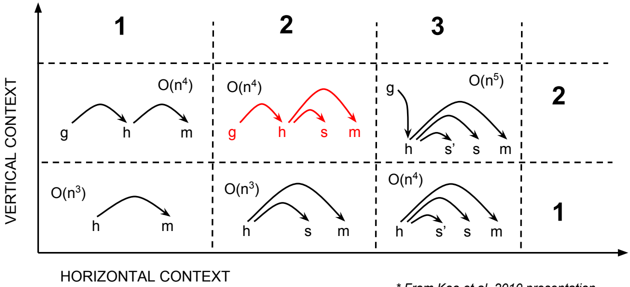
2nd-Order Horizontal Projective Parsing

- Add a sibling chart item to get to $O(n^3)$



Higher-Order Projective Parsing

- People played this game since 2006
  - McDonald and Pereira [2006] (2nd-order sibling)
  - Carreras [2007] (2nd-order sibling and grandparent)
  - Koo and Collins [2010] (3rd-order grand-sibling and tri-sibling)
  - Ma and Zhao [2012] (4th-order grand-tri-sibling+)



** From Koo et al. 2010 presentation*

Exact Higher-Order Projective Parsing

- Can be done via chart augmentation
- But there are drawbacks
  - $O(n^4)$, $O(n^5)$, ... is just too slow
  - Every type of higher order feature requires specialized chart items and combination rules

Exact Higher-Order Projective Parsing

- Can be done via chart augmentation
- But there are drawbacks
  - $O(n^4)$, $O(n^5)$, ... is just too slow
  - Every type of higher order feature requires specialized chart items and combination rules
- Led to research on approximations
  - Bohnet [2010]: feature hashing, parallelization
  - Koo and Collins [2010]: first-order marginal probabilities
  - Bergsma and Cherry [2010]: classifier arc filtering
  - Rush and Petrov [2012]: structured prediction cascades
  - He et al. [2013]: dynamic feature selection
  - Zhang and McDonald [2012], Zhang et al. [2013]: cube-pruning

Projective Parsing Summary

- Can augment chart (dynamic program) to increase scope of features, but comes at complexity cost
- Solution: use pruning approximations

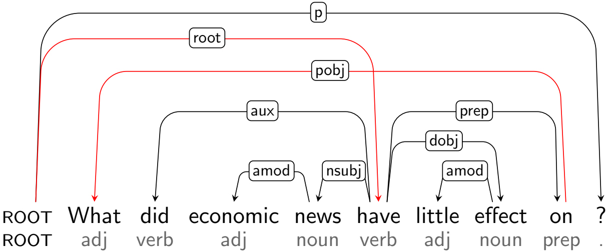
	En-UAS	Zh-UAS
1st order exact	91.8	84.4
2nd order exact	92.4	86.6
3rd order exact*	93.0	86.8
4th order exact†	93.4	87.4
struct. pred. casc.‡	93.1	–
cube-pruning⋆	93.5	87.9

* [Koo and Collins 2010], † [Ma and Zhao 2012], ‡ [Rush and Petrov 2012], ⋆ [Zhang et al. 2013]

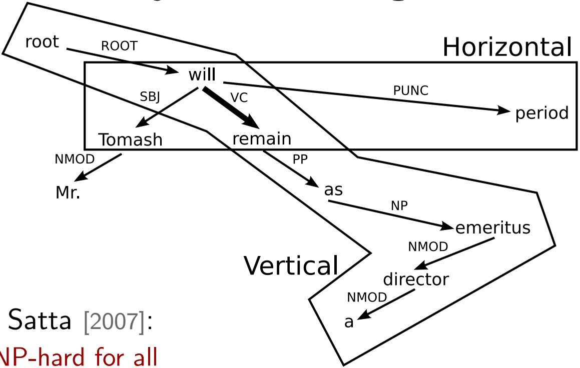
Cube-pruning is 5x faster and structured prediction cascades 10x faster than third-order

Non-Projective Parsing

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



Higher-Order Non-Projective Parsing



- McDonald and Satta [2007]:
  - Parsing is NP-hard for all higher-order features
  - Horizontal, vertical, valency, etc.
  - Even seemingly simple arc features like "Is this the only modifier" result in intractability

Higher-Order Non-Projective Parsing

Higher-Order Non-Projective Parsing

- Exact: integer linear programming (ILP)
  [Riedel and Clarke 2006, Kübler et al. 2009, Martins et al. 2009]
  - Inference intractable, but efficient optimizers exist
  - Easy to extend by adding labels, grammar constraints, etc.
  - Related to constraint dependency grammar

Higher-Order Non-Projective Parsing

- Exact: integer linear programming (ILP)
  [Riedel and Clarke 2006, Kübler et al. 2009, Martins et al. 2009]
  - Inference intractable, but efficient optimizers exist
  - Easy to extend by adding labels, grammar constraints, etc.
  - Related to constraint dependency grammar
- Approximate inference: $T^* = \text{argmax}_{T \in G_x} \; s(T)$
  - Post-processing [McDonald and Pereira 2006], [Hall and Novák 2005], [Hall 2007]
  - Sampling [Nakagawa 2007]
  - Belief Propagation [Smith and Eisner 2008]
  - Dual Decomposition [Koo et al. 2010]

Higher-Order Non-Projective Parsing

- Exact: integer linear programming (ILP)
  [Riedel and Clarke 2006, Kübler et al. 2009, Martins et al. 2009]
  - Inference intractable, but efficient optimizers exist
  - Easy to extend by adding labels, grammar constraints, etc.
  - Related to constraint dependency grammar
- Approximate inference: $T^* = \text{argmax}_{T \in G_x} \; s(T)$
  - Post-processing [McDonald and Pereira 2006], [Hall and Novák 2005], [Hall 2007]
  - Sampling [Nakagawa 2007]
  - Belief Propagation [Smith and Eisner 2008]
  - Dual Decomposition [Koo et al. 2010]
- Approximate search space: $T^* = \text{argmax}_{T \in G_x} \; s(T)$
  - Mildly non-projective structures
    [Bodirsky et al. 2005, Pitler et al. 2012, Pitler et al. 2013]

Transition-Based Dependency Parsing

- The basic idea:
  - Define a transition system for dependency parsing
  - Learn a model for scoring possible transitions
  - Parse by searching for the optimal transition sequence

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	.

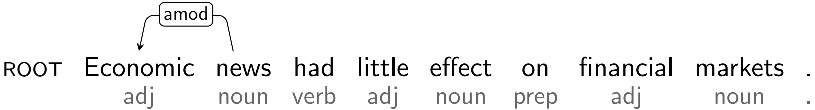
Example Transition Sequence

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

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

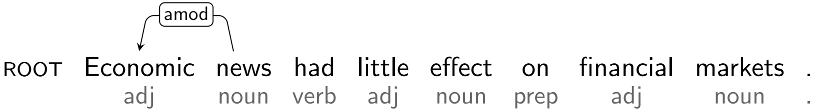
Example Transition Sequence

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



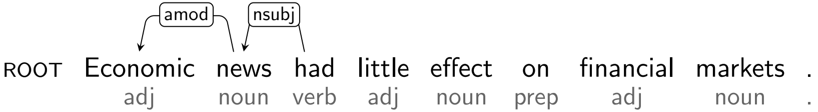
Example Transition Sequence

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



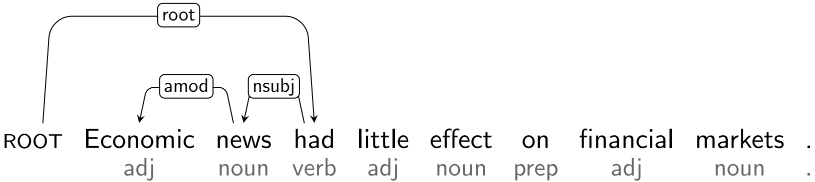
Example Transition Sequence

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



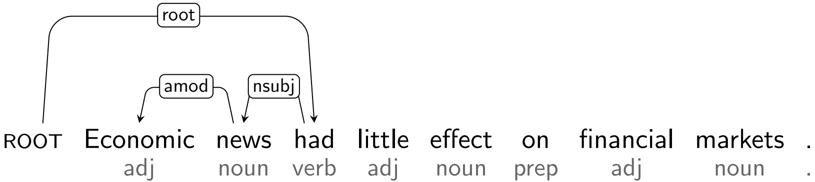
Example Transition Sequence

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



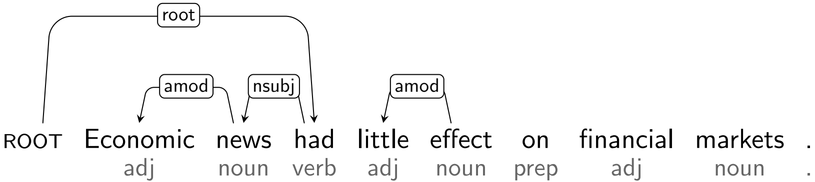
Example Transition Sequence

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



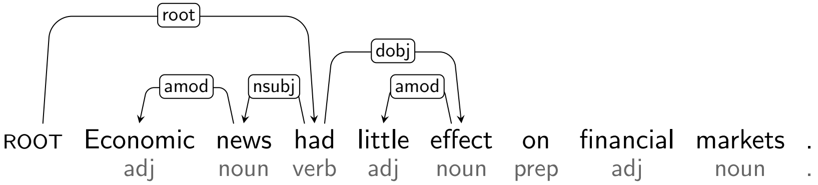
Example Transition Sequence

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



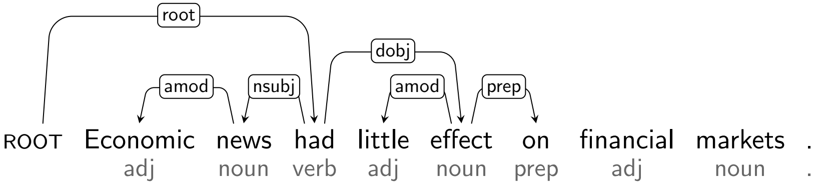
Example Transition Sequence

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



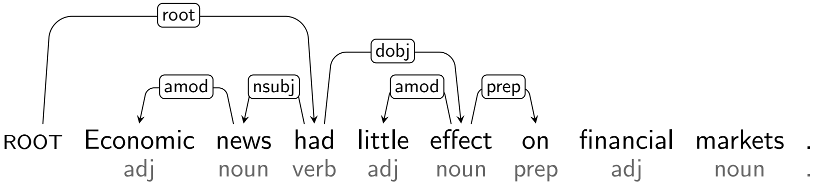
Example Transition Sequence

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



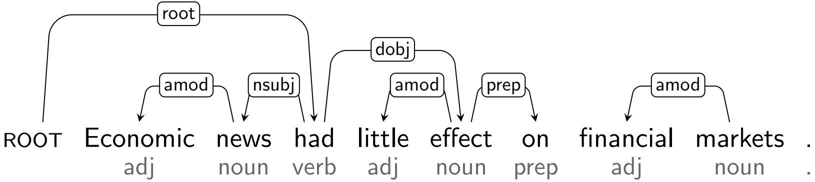
Example Transition Sequence

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



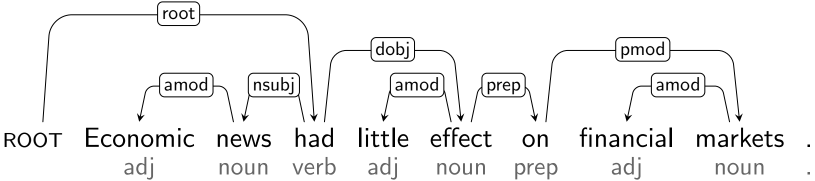
Example Transition Sequence

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



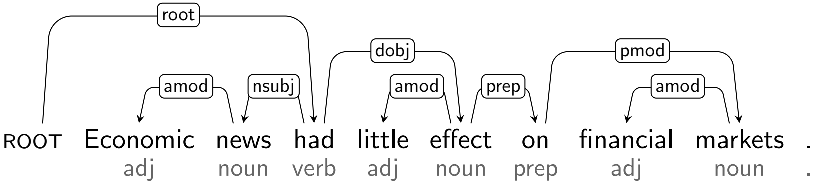
Example Transition Sequence

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



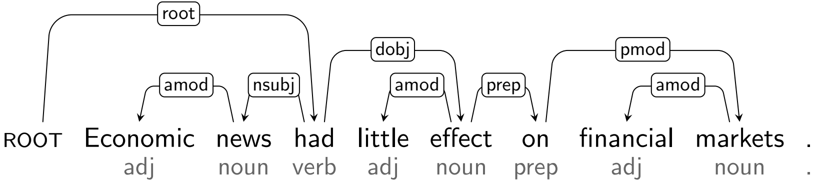
Example Transition Sequence

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



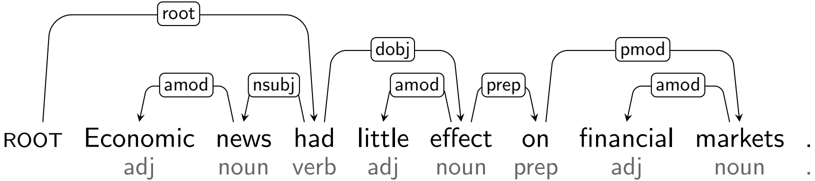
Example Transition Sequence

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



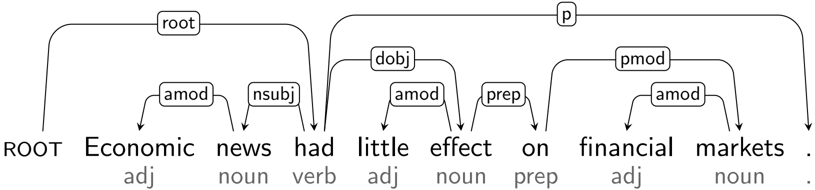
Example Transition Sequence

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



Example Transition Sequence

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



Arc-Standard Transition System [Nivre 2004]

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

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

Terminal: $([0], [\ ], A)$

Shift: $(S, i|B, A) \Rightarrow (S|i, B, A)$

Right-Arc(k): $(S|i|j, B, A) \Rightarrow (S|i, B, A \cup \{(i, j, k)\})$

Left-Arc(k): $(S|i|j, B, A) \Rightarrow (S|j, B, A \cup \{(j, i, k)\})$ $\quad i \neq 0$

Greedy Inference

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

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

- 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:

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

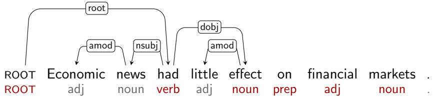
- History-based feature representation $\mathbf{f}(c, t)$
- Weight vector $\mathbf{w}$ learned from treebank data

Feature Representation

- Features over input tokens relative to S and B

Configuration

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



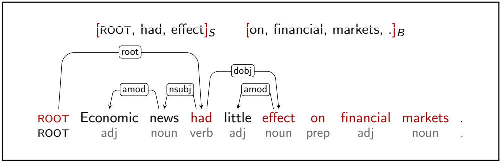
Features

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

Feature Representation

- Features over input tokens relative to S and B

Configuration



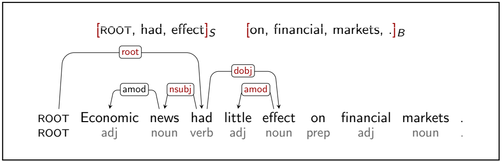
Features

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

Feature Representation

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

Configuration



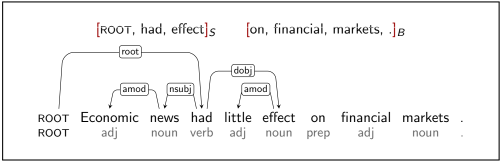
Features

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

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



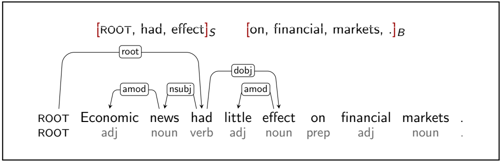
Features

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

Feature Representation

- 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

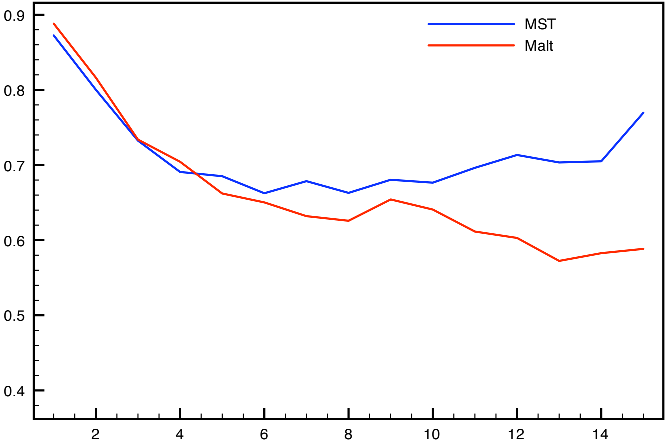
Greedy, Local, Transition-Based Parsing

- 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
- The major question in recent research on transition-based parsing has been how to improve learning and inference, while maintaining high efficiency and rich feature models

Empirical Analysis

- CoNLL 2006 shared task [Buchholz and Marsi 2006]:
  - MaltParser [Nivre et al. 2006] – deterministic, local learning
  - MSTParser [McDonald et al. 2006] – exact, global learning
  - Same average parsing accuracy over 13 languages
- Comparative error analysis [McDonald and Nivre 2007]:
  - MaltParser more accurate on short dependencies and disambiguation of core grammatical functions
  - MSTParser more accurate on long dependencies and dependencies near the root of the tree
- Hypothesized explanation for MaltParser results:
  - Rich features counteracted by error propagation

Precision by Dependency Length



Beam Search

- Maintain the k best hypotheses [Johansson and Nugues 2006]:

$$
\begin{array}{l}
\text{Parse}(w_1, \ldots, w_n) \\
\quad \text{Beam} \leftarrow \{([\,]_S, [0, 1, \ldots, n]_B, \{\,\})\} \\
\quad \textbf{while } \exists c \in \text{Beam } [B_c \neq [\,]] \\
\quad\quad \textbf{foreach } c \in \text{Beam} \\
\quad\quad\quad \textbf{foreach } t \\
\quad\quad\quad\quad \text{Add}(t(c), \text{NewBeam}) \\
\quad\quad \text{Beam} \leftarrow \text{Top}(k, \text{NewBeam}) \\
\quad \textbf{return } G = (\{0, 1, \ldots, n\}, A_{\text{Top}(1, \text{Beam})})
\end{array}
$$

- Note:
  - $\text{Score}(c_0, \ldots, c_m) = \sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{f}(c_{i-1}, t_i)$
  - Simple combination of locally normalized classifier scores
  - Marginal gains in accuracy

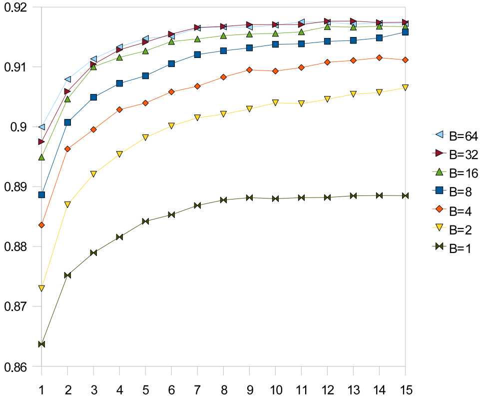
Structured Prediction

- Parsing as structured prediction [Zhang and Clark 2008]:
  - Minimize loss over entire transition sequence
  - Use beam search to find highest-scoring sequence
- Factored feature representations:

$$\mathbf{f}(c_0, \ldots, c_m) = \sum_{i=1}^{m} \mathbf{f}(c_{i-1}, t_i)$$

- Online learning from oracle transition sequences:
  - Structured perceptron [Collins 2002]
  - Early update [Collins and Roark 2004]
  - Max-violation update [Huang et al. 2012]

Beam Size and Training Iterations

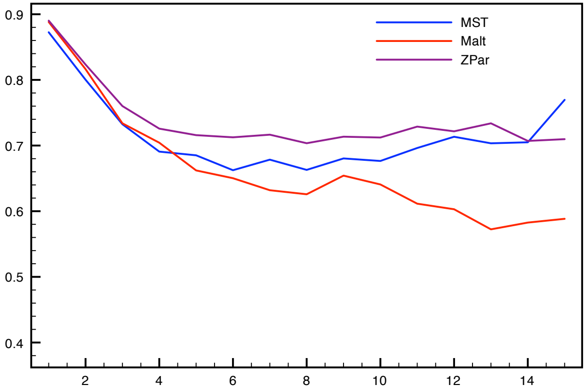


[Zhang and Clark 2008]

The Best of Two Worlds?

- Like graph-based dependency parsing (MSTParser):
  - Global learning – minimize loss over entire sentence
  - Non-greedy search – accuracy increases with beam size
- Like (old school) transition-based parsing (MaltParser):
  - Highly efficient – complexity still linear for fixed beam size
  - Rich features – no constraints from parsing algorithm

Precision by Dependency Length



[Zhang and Nivre 2012]

Non-Projective Parsing

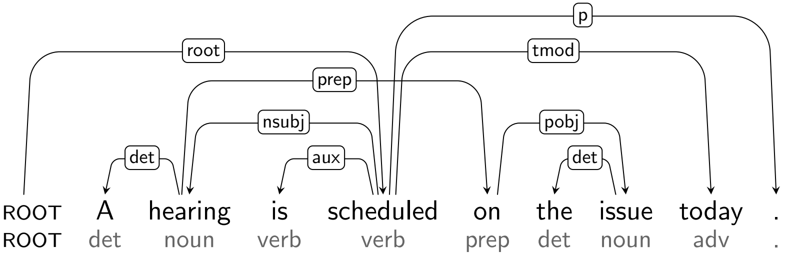
- So far only projective parsing models
- Non-projective parsing harder even with greedy inference
  - Non-projective: $n(n-1)$ arcs to consider – $O(n^2)$
  - Projective: at most $2(n-1)$ arcs to consider – $O(n)$
- Approaches:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Extended arc transitions [Attardi 2006]
  - List-based algorithms [Covington 2001, Nivre 2007]
  - Online reordering [Nivre 2009, Nivre et al. 2009]:

Non-Projective Parsing

- So far only projective parsing models
- Non-projective parsing harder even with greedy inference
  - Non-projective: $n(n-1)$ arcs to consider – $O(n^2)$
  - Projective: at most $2(n-1)$ arcs to consider – $O(n)$
- Approaches:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Extended arc transitions [Attardi 2006]
  - List-based algorithms [Covington 2001, Nivre 2007]
  - Online reordering [Nivre 2009, Nivre et al. 2009]:

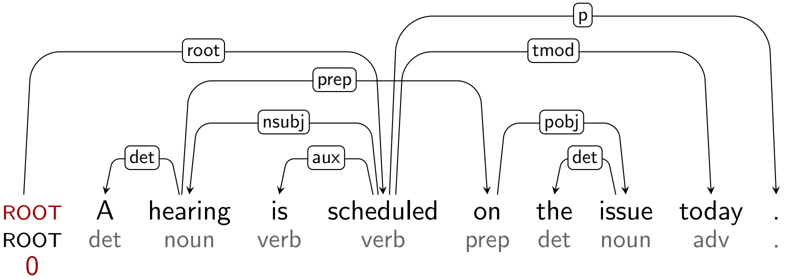
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



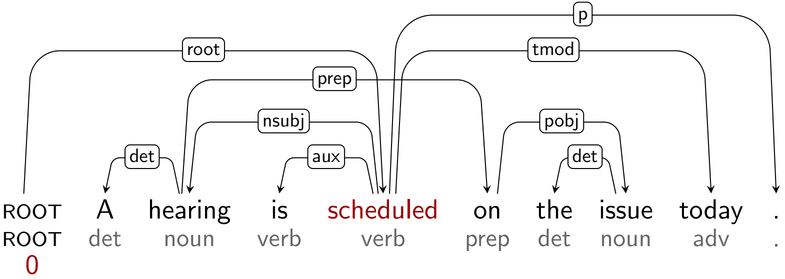
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



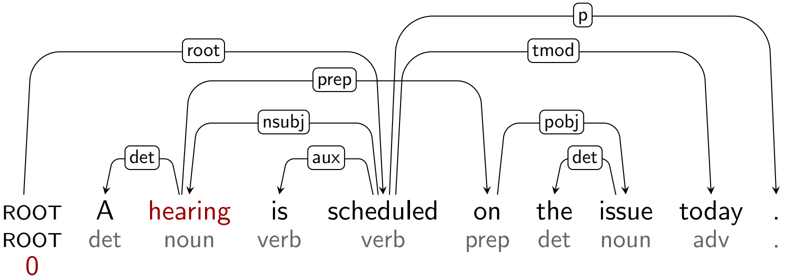
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



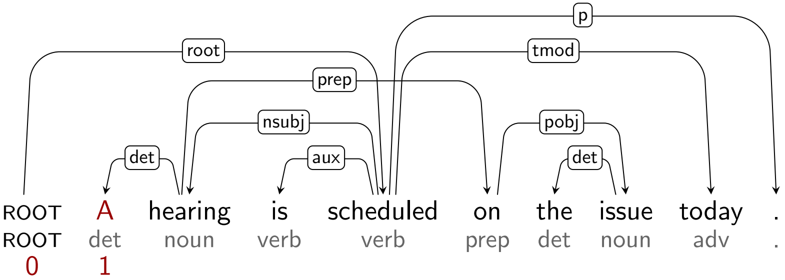
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



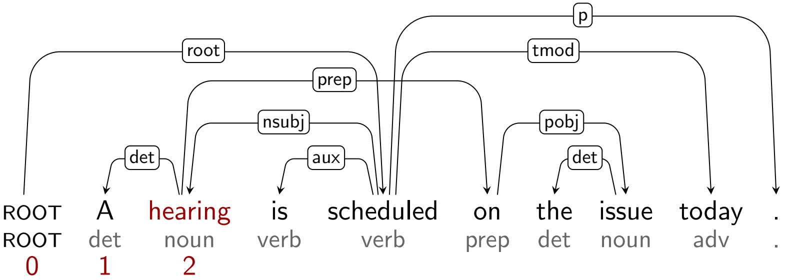
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



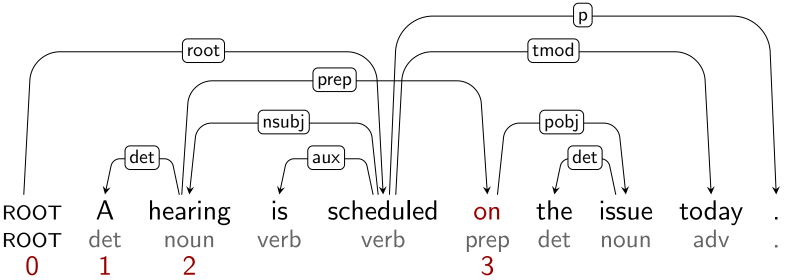
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



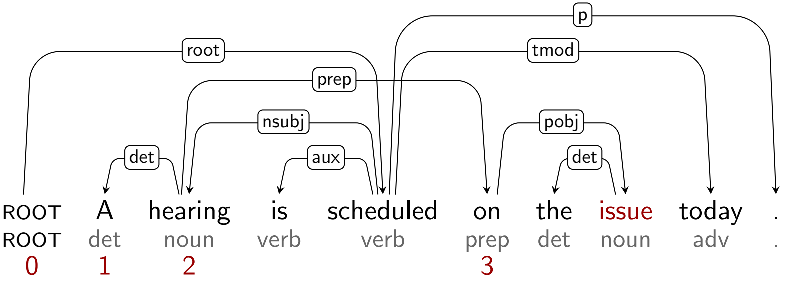
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



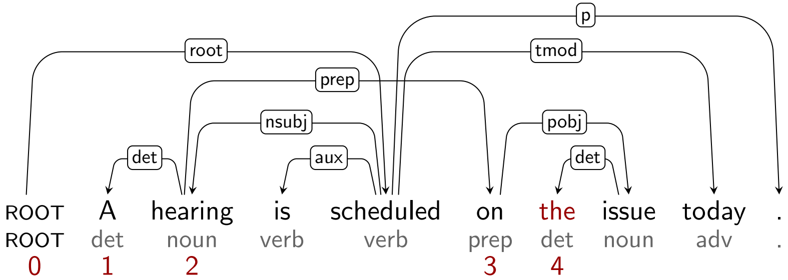
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



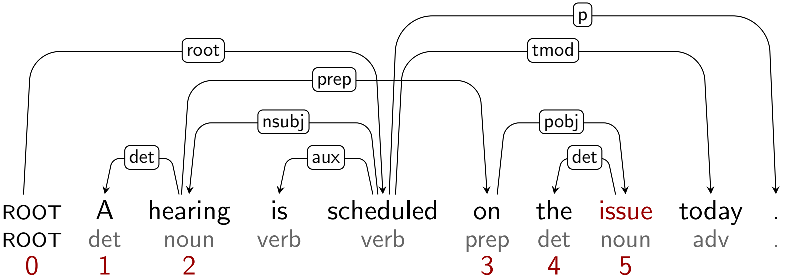
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



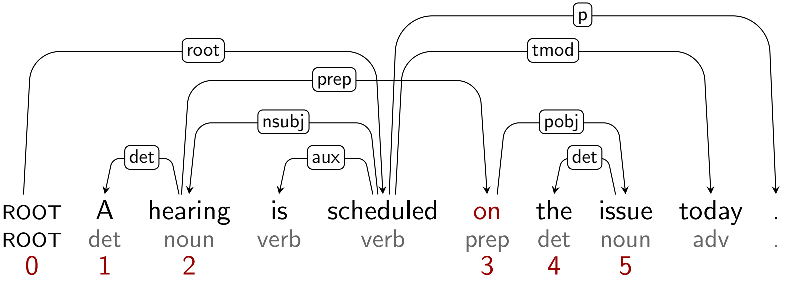
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



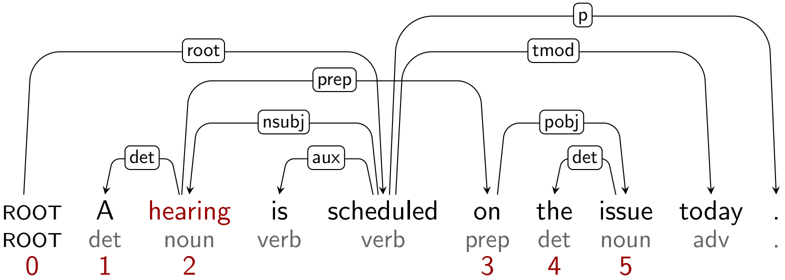
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



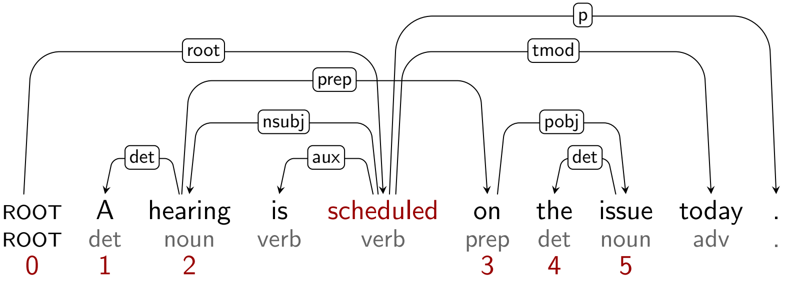
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



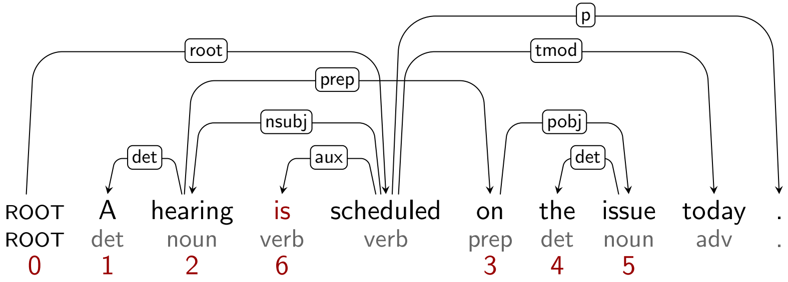
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



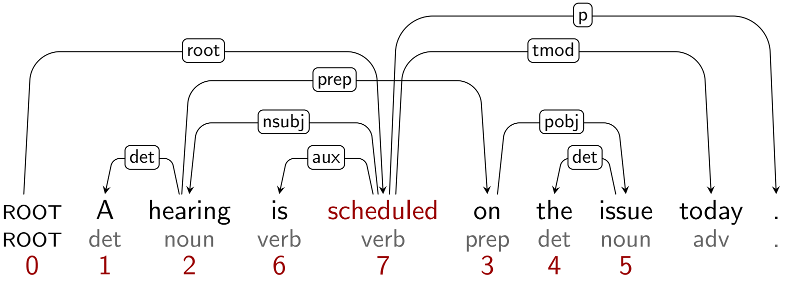
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



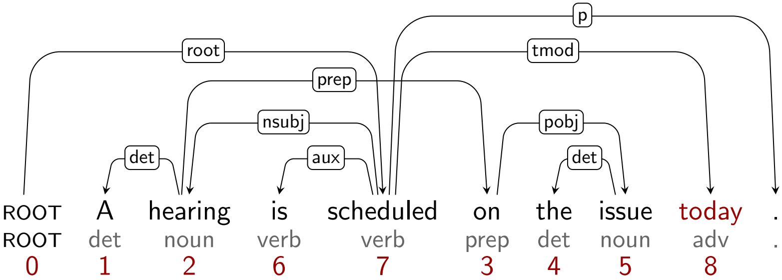
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



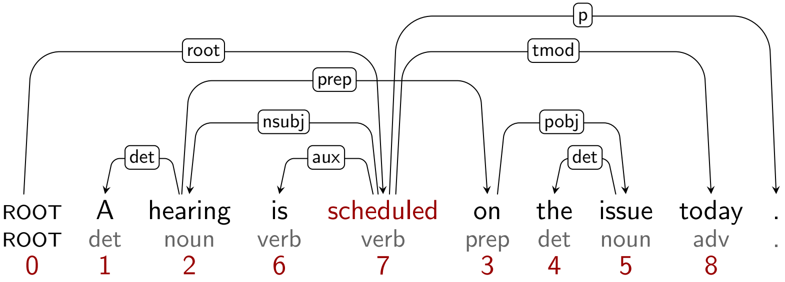
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



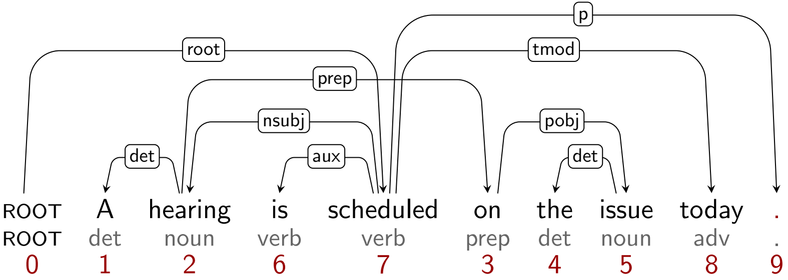
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



Transition System for Online Reordering

Configuration:	(S, B, A)	$[S = \text{Stack}, B = \text{Buffer}, A = \text{Arcs}]$		
Initial:	$([\,], [0, 1, \ldots, n], \{\,\})$			
Terminal:	$([0], [\,], A)$			
Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$	
Right-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|i, B, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, B, A \cup \{(j, i, k)\})$	$i \neq 0$
Swap:	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, i\|B, A)$	$0 < i < j$

Transition System for Online Reordering

Configuration:	(S, B, A)		$[S = \text{Stack}, B = \text{Buffer}, A = \text{Arcs}]$	
Initial:	$([\,], [0, 1, \ldots, n], \{\,\})$			
Terminal:	$([0], [\,], A)$			
Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$	
Right-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|i, B, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, B, A \cup \{(j, i, k)\})$	$i \neq 0$
Swap:	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, i\|B, A)$	$0 < i < j$

- Transition-based parsing with two interleaved processes:
  1. Sort words into projective order $<_p$
  2. Build tree T by connecting adjacent subtrees
- T is projective with respect to $<_p$ but not (necessarily) $<$

Example Transition Sequence

$[\ ]_S$ [ROOT, A, hearing, is, scheduled, on, the, issue, today, .$]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{A, hearing, is, scheduled, on, the, issue, today, .}]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

Example Transition Sequence

$[\text{ROOT, A}]_S$ $[\text{hearing, is, scheduled, on, the, issue, today, .}]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

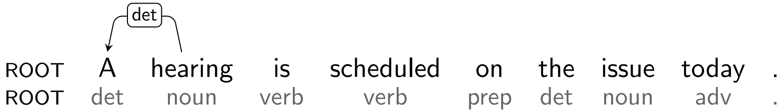
Example Transition Sequence

$[$ROOT, A, hearing$]_S$ $[$is, scheduled, on, the, issue, today, .$]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

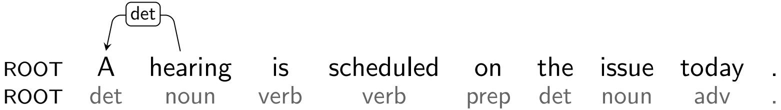
Example Transition Sequence

[ROOT, hearing]$_S$ [is, scheduled, on, the, issue, today, .]$_B$



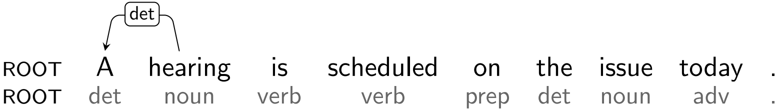
Example Transition Sequence

[ROOT, hearing, is]$_S$ [scheduled, on, the, issue, today, .]$_B$



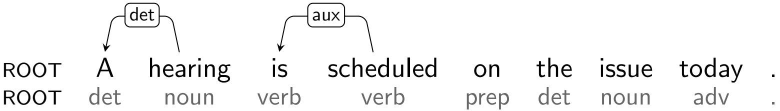
Example Transition Sequence

$[\text{ROOT, hearing, is, scheduled}]_S$ $[\text{on, the, issue, today, .}]_B$



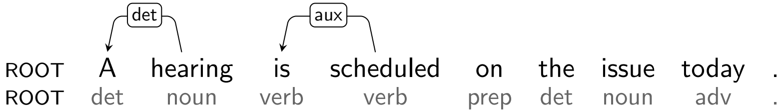
Example Transition Sequence

[ROOT, hearing, scheduled]$_S$ [on, the, issue, today, .]$_B$



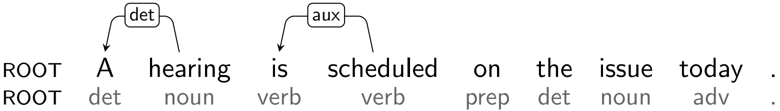
Example Transition Sequence

$[\text{ROOT, hearing, scheduled, on}]_S \ \ [\text{the, issue, today, .}]_B$



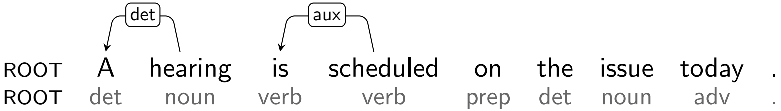
Example Transition Sequence

[ROOT, hearing, scheduled, on, the]$_S$ [issue, today, .]$_B$



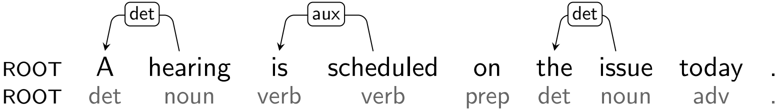
Example Transition Sequence

[ROOT, hearing, scheduled, on, the, issue]$_S$ [today, .]$_B$



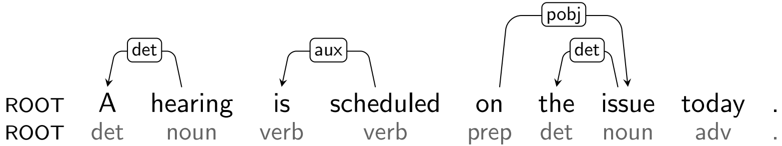
Example Transition Sequence

[ROOT, hearing, scheduled, on, issue]$_S$ [today, .]$_B$



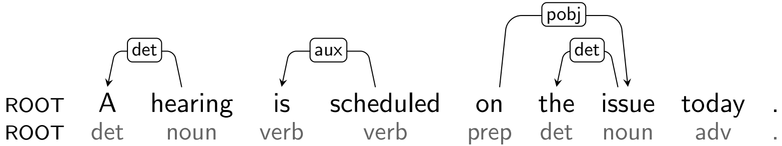
Example Transition Sequence

$[$ROOT, hearing, scheduled, on$]_S$ $[$today, .$]_B$



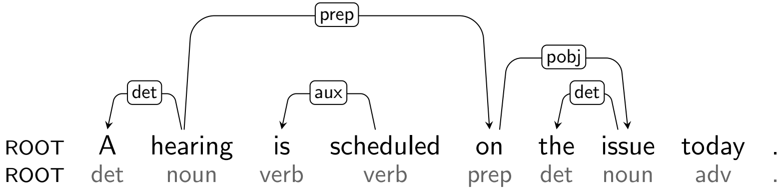
Example Transition Sequence

$[\text{ROOT, hearing, on}]_S$ $[\text{scheduled, today, .}]_B$



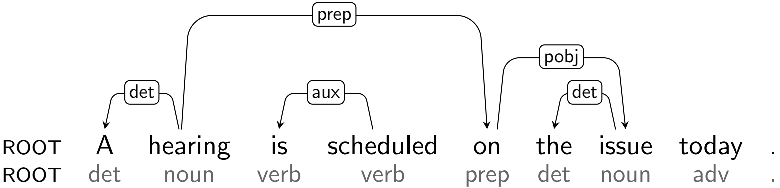
Example Transition Sequence

$[\text{ROOT, hearing}]_S$ $[\text{scheduled, today, .}]_B$



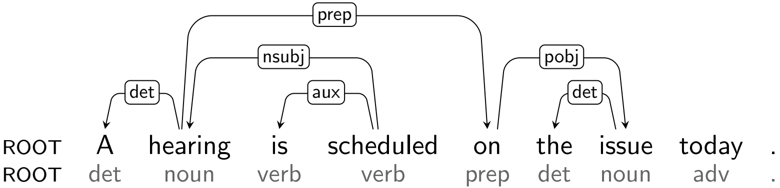
Example Transition Sequence

[ROOT, hearing, scheduled]$_S$ [today, .]$_B$



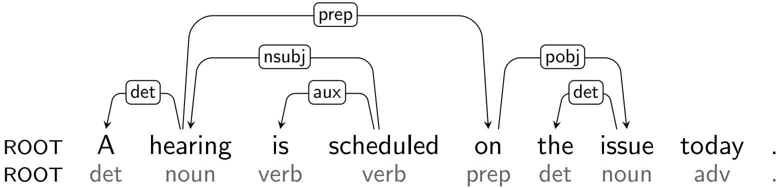
Example Transition Sequence

[ROOT, scheduled]$_S$ [today, .]$_B$



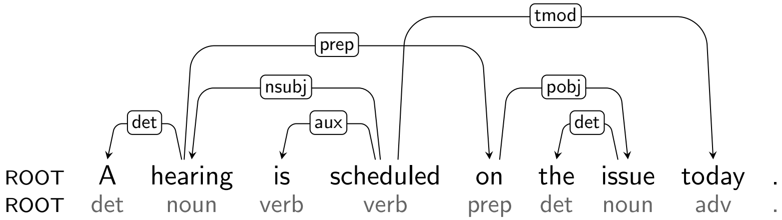
Example Transition Sequence

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



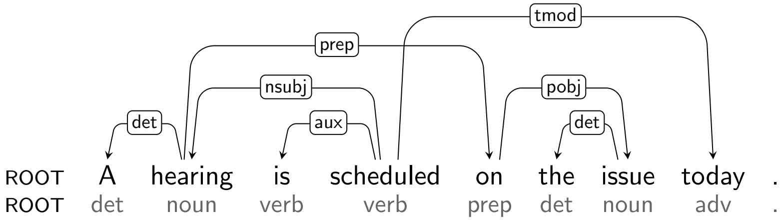
Example Transition Sequence

[ROOT, scheduled]$_S$ [.]$_B$



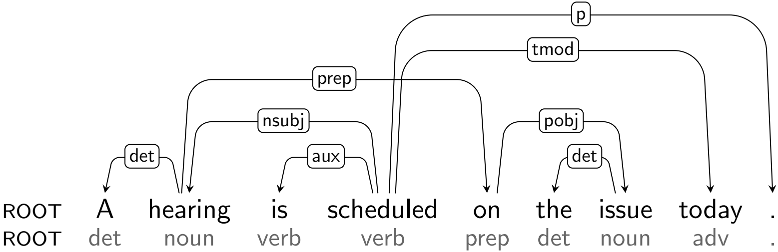
Example Transition Sequence

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



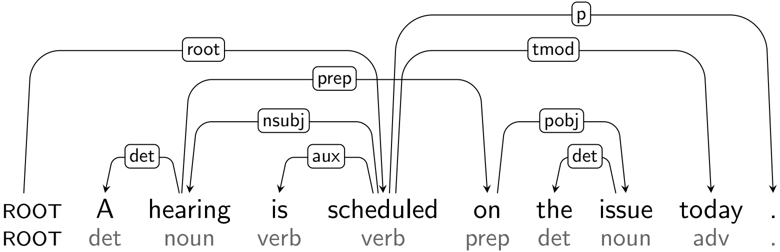
Example Transition Sequence

[ROOT, scheduled]$_S$ [ ]$_B$



Example Transition Sequence

$[\text{ROOT}]_S \quad [\ ]_B$



Analysis

- Correctness:
  - Sound and complete for the class of non-projective trees
- Complexity for greedy or beam search parsing:
  - Quadratic running time in the worst case
  - Linear running time in the average case
- Works well with beam search and structured prediction

	Czech		German	
	LAS	UAS	LAS	UAS
Projective	80.8	86.3	86.2	88.5
Reordering	83.9	89.1	88.7	90.9

[Bohnet and Nivre 2012]

Morphology and Syntax

- Morphological analysis in dependency parsing:
  - Crucially assumed as input, not predicted by the parser
  - Pipeline approach may lead to error propagation
  - Most PCFG-based parsers at least predict their own tags
- Recent interest in joint models for morphology and syntax:
  - Graph-based [McDonald 2006, Lee et al. 2011, Li et al. 2011]
  - Transition-based [Hatori et al. 2011, Bohnet and Nivre 2012]
- Can improve both morphology and syntax

Transition System for Morphology and Syntax

Configuration:	(S, B, M, A) $[M = \text{Morphology}]$			
Initial:	$([\ ], [0, 1, \ldots, n], \{\ \}, \{\ \})$			
Terminal:	$([0], [\ ], M, A)$			
Shift(p):	$(S, i\|B, M, A)$	$\Rightarrow$	$(S\|i, B, M \cup \{(i, m)\}, A)$	
Right-Arc(k):	$(S\|i\|j, B, M, A)$	$\Rightarrow$	$(S\|i, B, M, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\|i\|j, B, M, A)$	$\Rightarrow$	$(S\|j, B, M, A \cup \{(j, i, k)\})$	$i \neq 0$
Swap:	$(S\|i\|j, B, M, A)$	$\Rightarrow$	$(S\|j, i\|B, M, A)$	$0 < i < j$

Transition System for Morphology and Syntax

Configuration: (S, B, M, A) $[M = \text{Morphology}]$

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

Terminal: $([0], [\ ], M, A)$

Shift(p): $(S, i|B, M, A) \Rightarrow (S|i, B, M \cup \{(i, m)\}, A)$

Right-Arc(k): $(S|i|j, B, M, A) \Rightarrow (S|i, B, M, A \cup \{(i, j, k)\})$

Left-Arc(k): $(S|i|j, B, M, A) \Rightarrow (S|j, B, M, A \cup \{(j, i, k)\})$ $i \neq 0$

Swap: $(S|i|j, B, M, A) \Rightarrow (S|j, i|B, M, A)$ $0 < i < j$

- Transition-based parsing with three interleaved processes:
  - Assign morphology when words are shifted onto the stack
  - Optionally sort words into projective order $<_p$
  - Build dependency tree T by connecting adjacent subtrees

Parsing Richly Inflected Languages

- Full morphological analysis: lemma + postag + features
  - Beam search and structured predication
  - Parser selects from k best tags + features
  - Rule-based morphology provides additional features
- Evaluation metrics:
  - PM = morphology (postag + features)
  - LAS = labeled attachment score

	Czech		Finnish		German		Hungarian		Russian	
	PM	LAS	PM	LAS	PM	LAS	PM	LAS	PM	LAS
Pipeline	93.0	83.1	88.8	79.9	89.1	91.8	96.1	88.4	92.6	87.4
Joint	94.4	83.5	91.6	82.5	91.2	92.1	97.4	89.1	95.1	88.0

[Bohnet et al. 2013]

Where do we stand?

LAS: 83.8 v. 83.6
[McDonald & Nivre 2007]

2008

	Graph-based Parsers	**Transition-based Parsers**
2008	Global Inference Global Learning Local Feature Scope	Local Inference Local Learnng Global Feature Scope
↓	higher-order chart parsing pruning ILP dual decomp mildly non-projective etc.	beam search perceptron dynamic oracles dynamic programming more features etc.
2014	Global Inference Global Learnng Global Feature Scope	Global Inference Global Learnng Global Feature Scope

LAS: 85.8 v. 85.5
[Zhang et al. 2013]

Coming Up Next

1. Introduction to dependency grammar and dependency parsing
2. Graph-based and transition-based dependency parsing
3. Multiword expressions in dependency parsing
4. Practical lab session (MaltParser)

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.
- Shane Bergsma and Colin Cherry. 2010. Fast and accurate arc filtering for dependency parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 53–61.
- Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*.
- Bernd Bohnet and Jonas Kuhn. 2012. The best of both worlds – a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chpater of the Association for Computational Linguistics (EACL)*, pages 77–87.
- Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465.

- Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.
- Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97. Association for Computational Linguistics.
- Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164.
- Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 957–961.
- Y. J. Chu and T. J. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.
- Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 112–119.

- Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.
- 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.
- Keith Hall and Václav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 42–52. Association for Computational Linguistics.
- Keith Hall. 2007. K-best spanning tree parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint pos tagging and dependency parsing in chinese. In *Proceedings of*

5th International Joint Conference on Natural Language Processing (IJCNLP), pages 1216–1224.

- He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic feature selection for dependency parsing. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.
- Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151.
- Richard Johansson and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 206–210.
- Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics.
- Terry Koo, Alexander M Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298. Association for Computational Linguistics.

- Sandra Kübler, Joakim Nivre, and Ryan McDonald. 2009. *Dependency Parsing*. Morgan & Claypool Publishers.
- John Lee, Jason Naradowsky, and David A. Smith. 2011. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 885–894.
- Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for chinese pos tagging and dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1180–1191.
- Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *Proceedings of the Conference on Computational Linguistics (COLING)*, pages 785–796.
- André FT Martins, Noah A Smith, and Eric P Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 342–350. Association for Computational Linguistics.
- 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 and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88.
- Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 122–131.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. 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.
- Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 216–220.
- Ryan McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing. University of Pennsylvania*. Ph.D. thesis, PhD Thesis.

- Tetsuji Nakagawa. 2007. Multilingual dependency parsing using global features. In *EMNLP-CoNLL*, pages 952–956.
- Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.
- Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.
- Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76.
- 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. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings*

of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL), pages 50–57.

- 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. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 351–359.
- Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2012. Dynamic programming for higher order parsing of gap-minding trees. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 478–488. Association for Computational Linguistics.
- Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *Transactions of the Association for Computational Linguistics (TACL)*.
- Sebastian Riedel and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on*

Empirical Methods in Natural Language Processing, pages 129–137. Association for Computational Linguistics.

- Alexander M Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507. Association for Computational Linguistics.
- David A Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 145–156. Association for Computational Linguistics.
- Katerina Veselá, Havelka Jiri, and Eva Hajicová. 2004. Condition of projectivity in the underlying dependency structures. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 289–295.
- Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–571.
- Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on*

Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 320–331. Association for Computational Linguistics.

- Yue Zhang and Joakim Nivre. 2011. Transition-based parsing with rich non-local features. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 188–193.
- Yue Zhang and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 1391–1400.
- Liang Zhang, Huang, Kai Zhao, and Ryan McDonald. 2013. Online learning for inexact hypergraph search. In *Proceedings of Empirical Methods in Natural Language Processing*.