UPPSALA UNIVERSITET

Advanced Dependency Parsing

Joakim Nivre

Uppsala University
Linguistics and Philology

Based on tutorials with Ryan McDonald

Plan for the Lecture

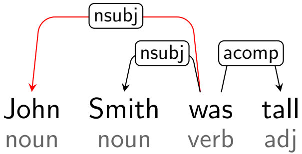
1. Graph-based vs. transition-based dependency parsing
2. Advanced graph-based parsing techniques
   - Higher order models
   - Non-projective parsing
3. Advanced transition-based parsing techniques
   - Beam search
   - Dynamic oracles
   - Non-projective parsing

Graph-Based Parsing

- Basic idea:
  - Define a space of candidate dependency trees for a sentence
  - Learning: Induce a model for scoring an entire dependency tree for a sentence
  - Parsing: Find the highest-scoring dependency tree, given the induced model
- Characteristics:
  - Global learning of a model for optimal dependency trees
  - Exhaustive search during parsing (exact)

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



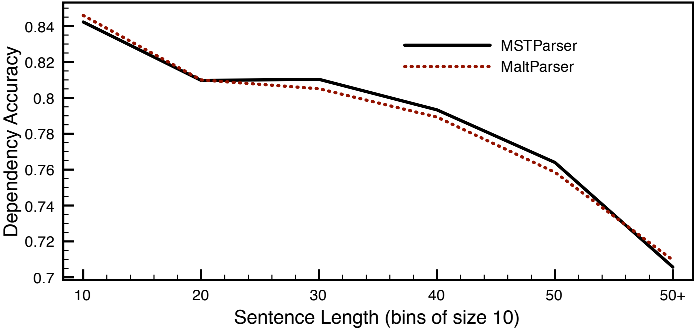
Transition-Based Parsing

- 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 learning of a model for optimal transitions
  - Greedy best-first search (heuristic)

Transition-Based Parsing Trade-Off

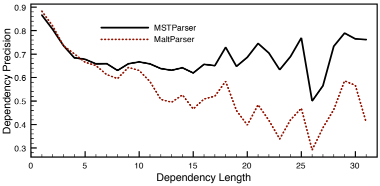
- Advantages:
  - Highly efficient parsing – linear time complexity
  - Rich history-based feature representations – no rigid constraints from parsing algorithm
- Drawback:
  - Sensitive to search errors and error propagation due to greedy inference and local learning

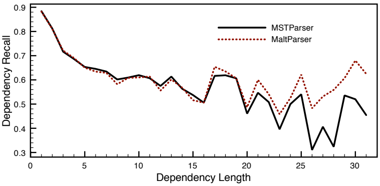
Error Analysis [McDonald and Nivre 2007]



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

Error Analysis [McDonald and Nivre 2007]





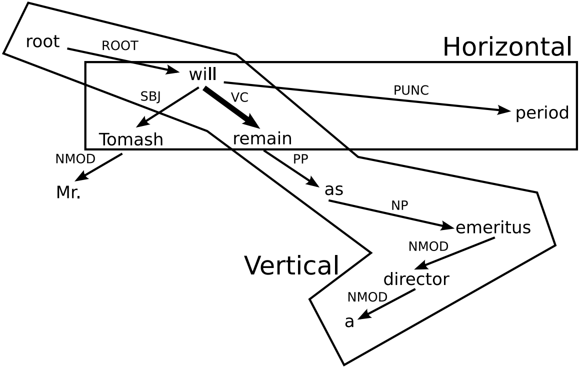
- MaltParser is more precise than MSTParser for short dependencies (1–3 words) but its performance degrades drastically with increasing dependency length ($>$ 10 words)
- MSTParser has more or less constant precision for dependencies longer than 3 words
- Recall is very similar across systems

Plan for the Lecture

1. Graph-based vs. transition-based dependency parsing
2. Advanced graph-based parsing techniques
   - Higher order models
   - Non-projective parsing
3. Advanced transition-based parsing techniques
   - Beam search
   - Dynamic oracles
   - Non-projective parsing
4. Neural networks in dependency parsing

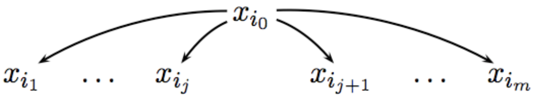
Higher-Order Models

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



2nd-Order Horizontal Projective Parsing

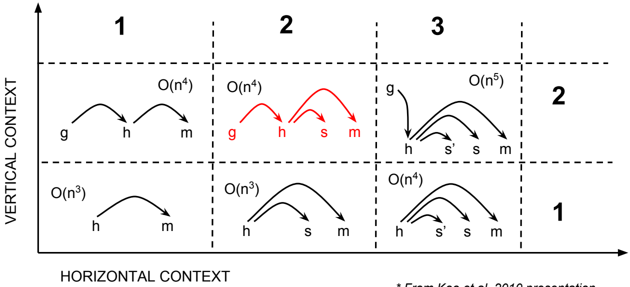
- Score factors by pairs of horizontally adjacent arcs
- Often called sibling dependencies
- $s(i, j, j')$ = score of adjacent arcs $x_i \to x_j$ and $x_i \to x_{j'}$



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

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+)



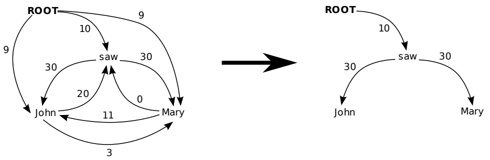
* *From Koo et al. 2010 presentation*

Parsing Algorithms

- Eisner's algorithm can be generalized to higher orders
- But there is a price to pay:
  - Specialized chart items and combination rules
  - Time complexity increases for every added order
  - Anything beyond 2nd-order is too slow in practice
- Remember basic trade-off:
  - Global training and exact inference – local feature scope
  - Increasing feature scope makes exact inference harder
- This has led to research on approximate graph-based parsing

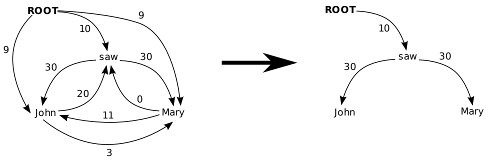
Non-Projective Parsing

- First-order model – equivalent to MST problem
- Chu-Liu-Edmonds' algorithm:
  - Construct a graph with the highest-scoring head for each word
  - If this is a tree, it must be the MST
  - If not, contract a cycle and recurse on smaller graph



Non-Projective Parsing

- First-order model – equivalent to MST problem
- Chu-Liu-Edmonds' algorithm:
  - Construct a graph with the highest-scoring head for each word
  - If this is a tree, it must be the MST
  - If not, contract a cycle and recurse on smaller graph



- This does not generalize to higher orders – no exact algorithm

Plan for the Lecture

1. Graph-based vs. transition-based dependency parsing
2. Advanced graph-based parsing techniques
   - Higher order models
   - Non-projective parsing
3. Advanced transition-based parsing techniques
   - Beam search
   - Dynamic oracles
   - Non-projective parsing
4. Neural networks in dependency parsing

Greedy Search

- Take the single best action at any point (given by oracle o):

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

- Maximally efficient – linear time complexity
- Sensitive to search errors and error propagation

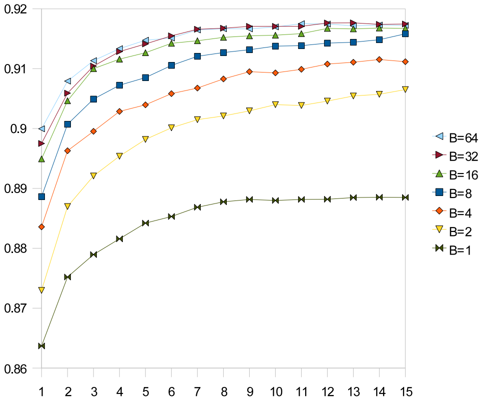
Beam Search

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

  Parse($w_1, \ldots, w_n$)
  1 Beam $\leftarrow \{([\,]_S, [0, 1, \ldots, n]_B, \{\,\})\}$
  2 **while** $\exists c \in$ Beam $[B_c \neq [\,]]$
  3 **foreach** $c \in$ Beam
  4 **foreach** t
  5 Add($t(c)$, NewBeam)
  6 Beam $\leftarrow$ Top(k, NewBeam)
  7 **return** $G = (\{0, 1, \ldots, n\}, A_{\text{Top}(1,\,\text{Beam})})$

- Note:
  - Pruning the beam requires that we score transition sequences
  - Global learning to maximize score of entire sequence

Beam Size

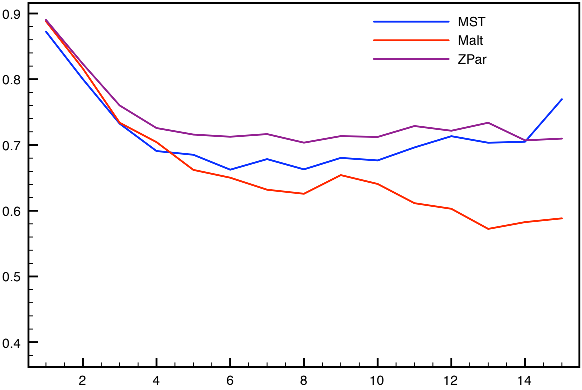


[Zhang and Clark 2008]

The Best of Two Worlds?

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

Precision by Dependency Length



[Zhang and Nivre 2012]

Dynamic Oracles

- Beam search helps because it explores the search space
  - At parsing time, the parser can recover from early bad decisions
  - At training time, the parser can learn to avoid costly mistakes
- Can the parser benefit from exploration only at training time?
  - Yes – but we need dynamic oracles for training
  - Then we can improve greedy parsing for maximum speed

Online Learning with a Conventional Oracle

```
Learn({T_1, ..., T_N})
    w ← 0.0
    for i in 1..K
        for j in 1..N
            c ← ([ ], [0, 1, ..., n_j], { })
            while B_c ≠ [ ]
                t* ← argmax_t w · f(c, t)
                t_o ← o(c, T_i)
                if t* ≠ t_o
                    w ← w + f(c, t_o) − f(c, t*)
                c ← t_o(c)
    return w
```

Online Learning with a Conventional Oracle

Learn($\{T_1, \ldots, T_N\}$)

$\mathbf{w} \leftarrow 0.0$

for i in $1..K$

  for j in $1..N$

    $c \leftarrow ([\,], [0, 1, \ldots, n_j], \{\,\})$

    while $B_c \neq [\,]$

      $t^* \leftarrow \text{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$

      $t_o \leftarrow o(c, T_i)$

      if $t^* \neq t_o$

        $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$

      $c \leftarrow t_o(c)$

return $\mathbf{w}$

- Oracle $o(c, T_i)$ returns the optimal transition for c and T_i

Conventional Oracle for Arc-Eager Parsing

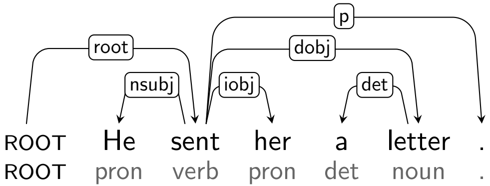
$$o(c, T) = \begin{cases} \text{Left-Arc} & \text{if } \text{top}(S_c) \leftarrow \text{first}(B_c) \text{ in } T \\ \text{Right-Arc} & \text{if } \text{top}(S_c) \rightarrow \text{first}(B_c) \text{ in } T \\ \text{Reduce} & \text{if } \exists v < \text{top}(S_c) : v \leftrightarrow \text{first}(B_c) \text{ in } T \\ \text{Shift} & \text{otherwise} \end{cases}$$

- Correct:
  - Derives T in a configuration sequence $C_{o,T} = c_0, \ldots, c_m$
- Problems:
  - Deterministic: Ignores other derivations of T
  - Incomplete: Valid only for configurations in $C_{o,T}$

Oracle Parse

Transitions:

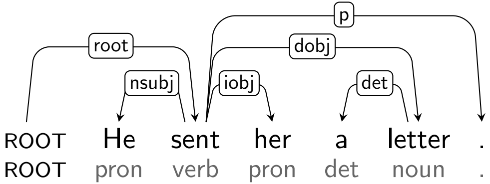
Stack	**Buffer**	**Arcs**
[ ]	[ROOT, He, sent, her, a, letter, .]	



Oracle Parse

Transitions: SH

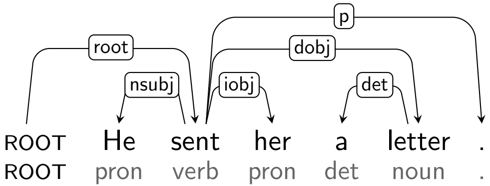
Stack	**Buffer**	**Arcs**
[ROOT]	[He, sent, her, a, letter, .]	



Oracle Parse

Transitions: SH-SH

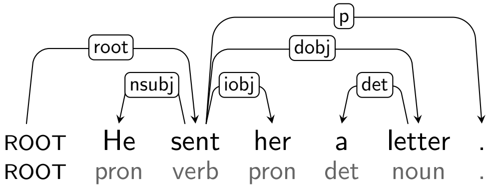
Stack	**Buffer**	**Arcs**
[ROOT, He]	[sent, her, a, letter, .]	



Oracle Parse

Transitions: SH-SH-LA

Stack	**Buffer**	**Arcs**
[ROOT]	[sent, her, a, letter, .]	He $\xleftarrow{\text{sbj}}$ sent



Oracle Parse

Transitions: SH-SH-LA-RA

Stack

[ROOT, sent]

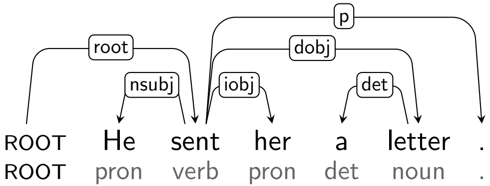
Buffer

[her, a, letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent



Oracle Parse

Transitions: SH-SH-LA-RA-RA

Stack

[ROOT, sent, her]

Buffer

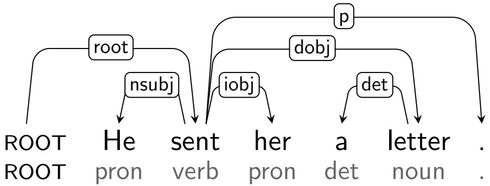
[a, letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Oracle Parse

Transitions: SH-SH-LA-RA-RA-SH

Stack

[ROOT, sent, her, a]

Buffer

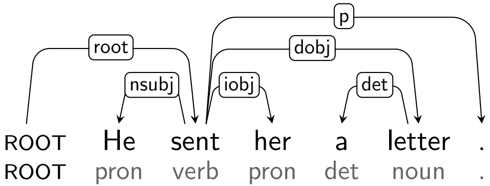
[letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Oracle Parse

Transitions: SH-SH-LA-RA-RA-SH-LA

Stack

[ROOT, sent, her]

Buffer

[letter, .]

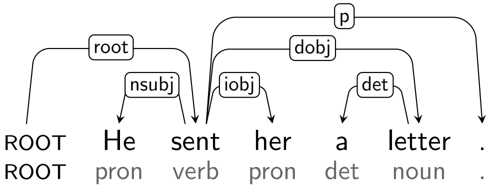
Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter



Oracle Parse

Transitions: SH-SH-LA-RA-RA-SH-LA-RE

Stack

[ROOT, sent]

Buffer

[letter, .]

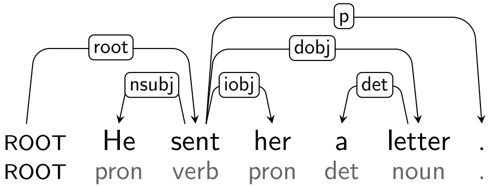
Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter



Oracle Parse

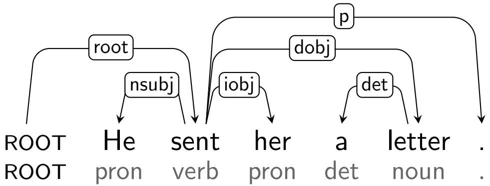
Transitions: SH-SH-LA-RA-RA-SH-LA-RE-RA

Stack

[ROOT, sent, letter]

Buffer

[.]



Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter

Oracle Parse

Transitions: SH-SH-LA-RA-RA-SH-LA-RE-RA-RE

Stack

[ROOT, sent]

Buffer

[.]

Arcs

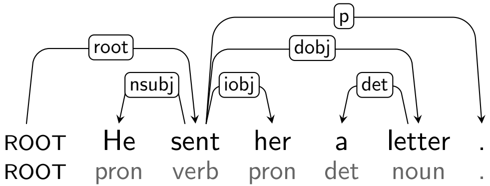
He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



Oracle Parse

Transitions: SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Stack

[ROOT, sent, .]

Buffer

[ ]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

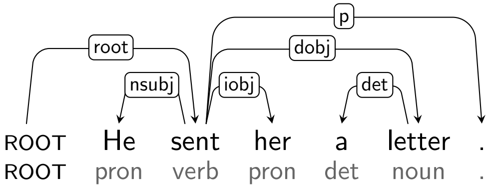
ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter

sent $\xrightarrow{\text{p}}$.



Non-Determinisim

Transitions: SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-RA

Stack

[ROOT, sent, her]

Buffer

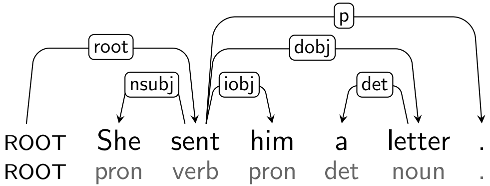
[a, letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

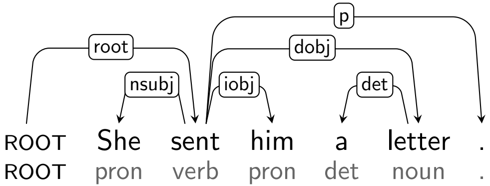
sent $\xrightarrow{\text{iobj}}$ her



Non-Determinisim

Transitions: SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-RA-RE

Stack	Buffer	Arcs
[ROOT, sent]	[a, letter, .]	He $\xleftarrow{\text{sbj}}$ sent ROOT $\xrightarrow{\text{root}}$ sent sent $\xrightarrow{\text{iobj}}$ her



Non-Determinisim

Transitions: SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-RA-RE-SH

Stack

[ROOT, sent, a]

Buffer

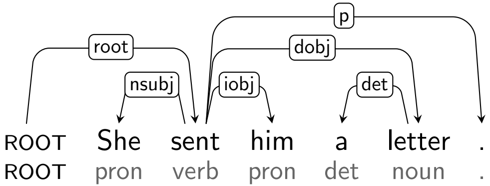
[letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Non-Determinisim

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-RA-RE-SH-LA

Stack

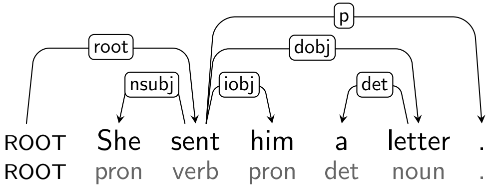
[ROOT, sent]

Buffer

[letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent
ROOT $\xrightarrow{\text{root}}$ sent
sent $\xrightarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ letter



Non-Determinisim

Transitions:

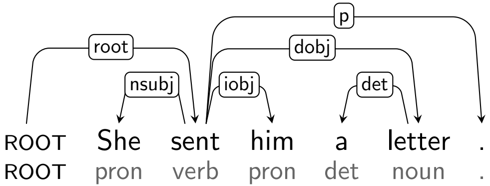
SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-RA-RE-SH-LA-RA

Stack

[ROOT, sent, letter]

Buffer

[.]



Arcs

He $\xleftarrow{\text{sbj}}$ sent
ROOT $\xrightarrow{\text{root}}$ sent
sent $\xrightarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ letter
sent $\xrightarrow{\text{dobj}}$ letter

Non-Determinisim

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-RA-RE-SH-LA-RA-RE

Stack

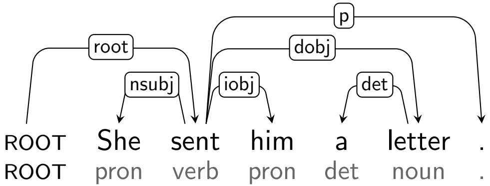
[ROOT, sent]

Buffer

[.]

Arcs

He $\xleftarrow{\text{sbj}}$ sent
ROOT $\xrightarrow{\text{root}}$ sent
sent $\xrightarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ letter
sent $\xrightarrow{\text{dobj}}$ letter



Non-Determinisim

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

SH-SH-LA-RA-RA-RE-SH-LA-RA-RE-RA

Stack

[ROOT, sent, .]

Buffer

[ ]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

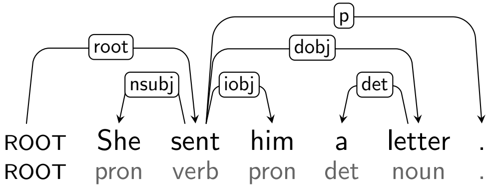
ROOT $\xrightarrow{\text{root}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter

sent $\xrightarrow{\text{p}}$.



Non-Optimality

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-SH-LA-RA

Stack

[ROOT, sent]

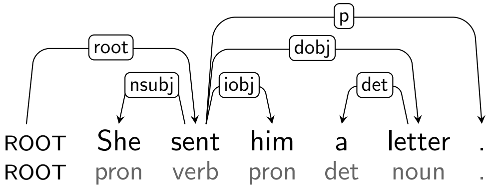
Buffer

[her, a, letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent



Non-Optimality

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-SH-LA-RA-SH

Stack

[ROOT, sent, her]

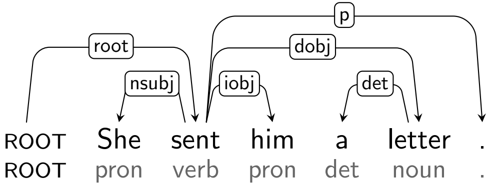
Buffer

[a, letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent



Non-Optimality

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-SH-LA-RA-SH-SH

Stack

[ROOT, sent, her, a]

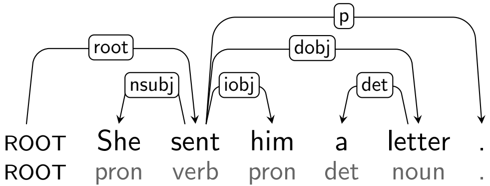
Buffer

[letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent



Non-Optimality

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-SH-LA-RA-SH-SH-LA

Stack

[ROOT, sent, her]

Buffer

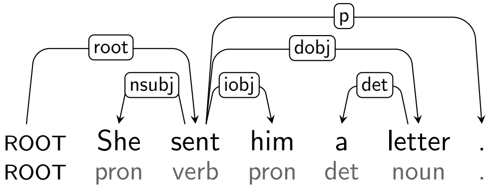
[letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-SH-LA-RA-SH-SH-LA-SH

Stack

[ROOT, sent, her, letter]

Buffer

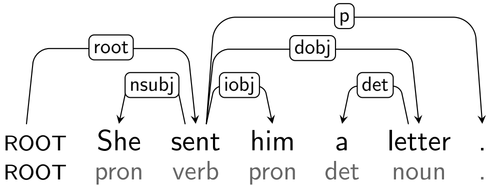
[.]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-SH-LA-RA-SH-SH-LA-SH-SH [3/6]

Stack

[ROOT, sent, letter, .]

Buffer

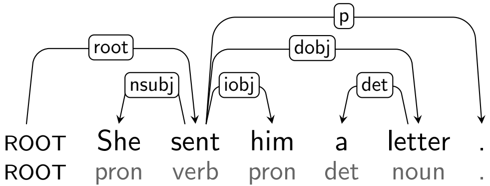
[ ]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA
SH-SH-LA-RA-SH-SH-LA-SH-SH [3/6]
SH-RA-LA-SH-SH-SH-LA

Stack

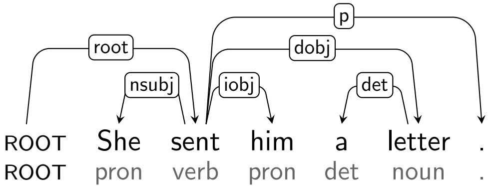
[ROOT, sent, her]

Buffer

[letter, .]

Arcs

He $\xleftarrow{\text{sbj}}$ sent
ROOT $\xrightarrow{\text{root}}$ sent
a $\xleftarrow{\text{det}}$ letter



Non-Optimality

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

SH-SH-LA-RA-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA

Stack

[ROOT, sent]

Buffer

[letter, .]

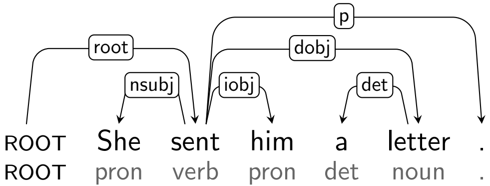
Arcs

He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{?}$ letter



Non-Optimality

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

SH-SH-LA-RA-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA-RA

Stack

[ROOT, sent, letter]

Buffer

[.]

Arcs

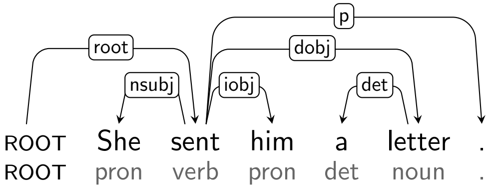
He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{?}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



Non-Optimality

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

SH-SH-LA-RA-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA-RA-RE

Stack

[ROOT, sent]

Buffer

[.]

Arcs

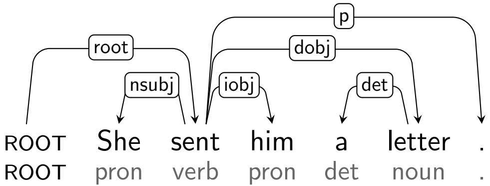
He $\xleftarrow{\text{sbj}}$ sent

ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{?}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



Non-Optimality

Transitions:

SH-SH-LA-RA-RA-SH-LA-RE-RA-RE-RA

SH-SH-LA-RA-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA-RA-RE-RA [5/6]

Stack

[ROOT, sent, .]

Buffer

[ ]

Arcs

He $\xleftarrow{\text{sbj}}$ sent

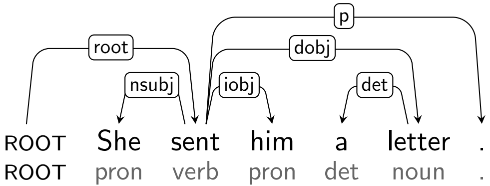
ROOT $\xrightarrow{\text{root}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{?}$ letter

sent $\xrightarrow{\text{dobj}}$ letter

sent $\xrightarrow{\text{p}}$.



Dynamic Oracles

- Optimality:
  - A transition is optimal if the best tree remains reachable
  - Best tree $= \text{argmin}_{T'} \mathcal{L}(T, T')$
- Oracle:
  - Boolean function $o(c, t, T) =$ **true** if t is optimal for c and T
  - Non-deterministic: More than one transition can be optimal
  - Complete: Correct for all configurations
- New problem:
  - How do we know which trees are reachable?
  - Easy for some transition systems (called arc-decomposable)

Oracles for Arc-Decomposable Systems

$$o(c,t,T) = \begin{cases} \textbf{true} & \text{if } [\mathcal{R}(c) - \mathcal{R}(t(c))] \cap T = \emptyset \\ \textbf{false} & \text{otherwise} \end{cases}$$

$$\text{where } \mathcal{R}(c) \equiv \{a \mid a \text{ is an arc reachable in } c\}$$

Arc-Eager

$$o(c, \text{LA}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in B_c : s \leftrightarrow w \in T \text{ (except } s \leftarrow b) \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{RA}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in S_c : w \leftrightarrow b \in T \text{ (except } s \rightarrow b) \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{RE}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in B_c : s \rightarrow w \in T \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{SH}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in S_c : w \leftrightarrow b \in T \\ \textbf{true} & \text{otherwise} \end{cases}$$

Notation: s = node on top of the stack S
b = first node in the buffer B

Online Learning with a Dynamic Oracle

Learn($\{T_1, \ldots, T_N\}$)

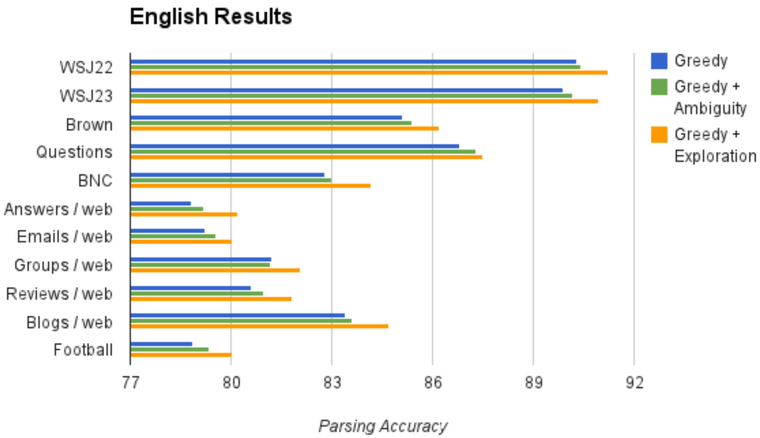
- $\mathbf{w} \leftarrow 0.0$
- **for** i in $1..K$
  - **for** j in $1..N$
    - $c \leftarrow ([\,]_S, [w_1, \ldots, w_{n_j}]_B, \{\,\})$
    - **while** $B_c \neq [\,]$
      - $t^* \leftarrow \mathrm{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$
      - $t_o \leftarrow \mathrm{argmax}_{t \in \{t | o(c,t,T_i)\}}\, \mathbf{w} \cdot \mathbf{f}(c, t)$
      - **if** $t^* \neq t_o$
        - $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
      - $c \leftarrow \mathrm{choice}(t_o(c), t^*(c))$
- **return w**

Online Learning with a Dynamic Oracle

Learn($\{T_1, \ldots, T_N\}$)

$\mathbf{w} \leftarrow 0.0$
for i in $1..K$
    for j in $1..N$
        $c \leftarrow ([\,]_S, [w_1, \ldots, w_{n_j}]_B, \{\,\})$
        while $B_c \neq [\,]$
            $t^* \leftarrow \text{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$
            $t_o \leftarrow \text{argmax}_{t \in \{t | o(c,t,T_i)\}}\, \mathbf{w} \cdot \mathbf{f}(c, t)$
            if $t^* \neq t_o$
                $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
            $c \leftarrow \text{choice}(t_o(c), t^*(c))$
return w

- Ambiguity: use model score to break ties
- Exploration: follow model prediction even if not optimal



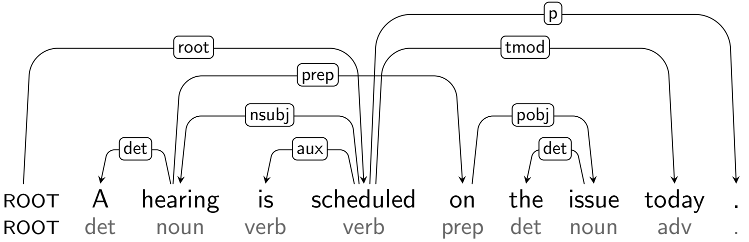
[Goldberg and Nivre 2012]

Non-Projective Parsing

- Standard transition systems only derive projective trees
- Approaches to non-projective transition-based parsing:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Non-adjacent arc transitions [Covington 2001, Attardi 2006, Nivre 2007]
  - Online reordering [Nivre 2009, Nivre et al. 2009]

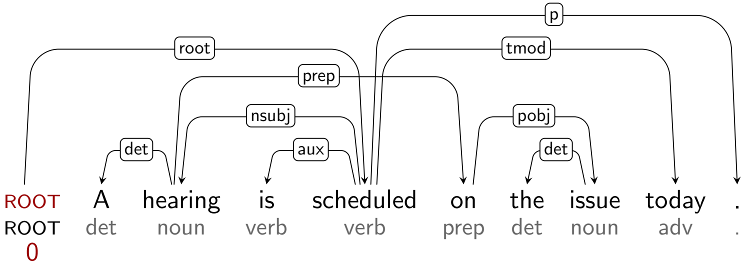
Projectivity and Word Order

- 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]



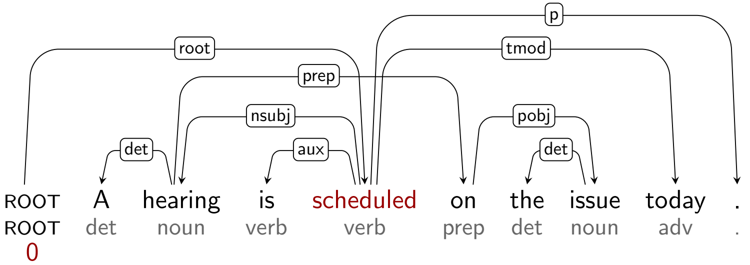
Projectivity and Word Order

- 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]



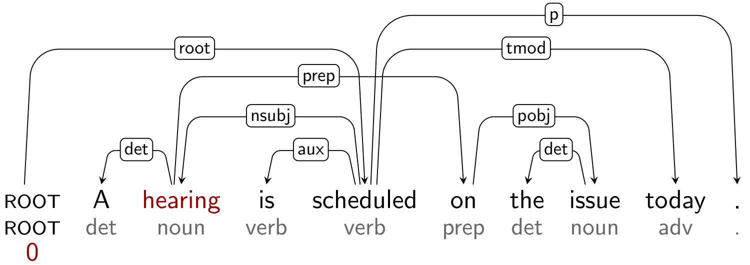
Projectivity and Word Order

- 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]



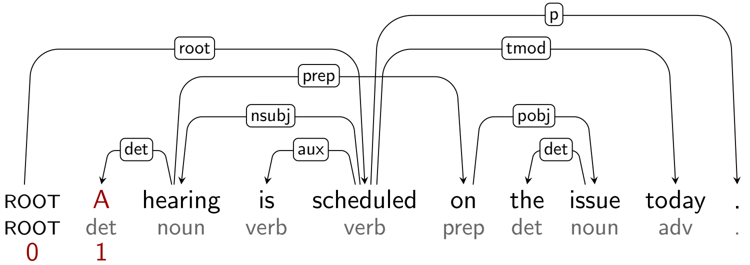
Projectivity and Word Order

- 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]



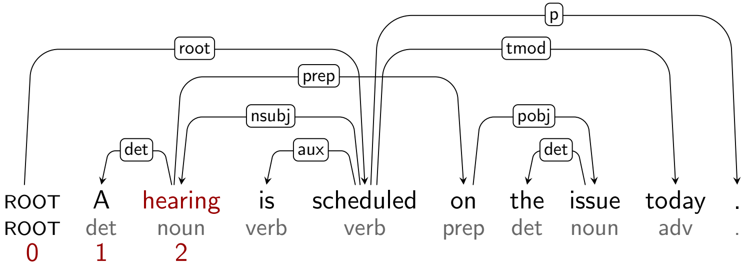
Projectivity and Word Order

- 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]



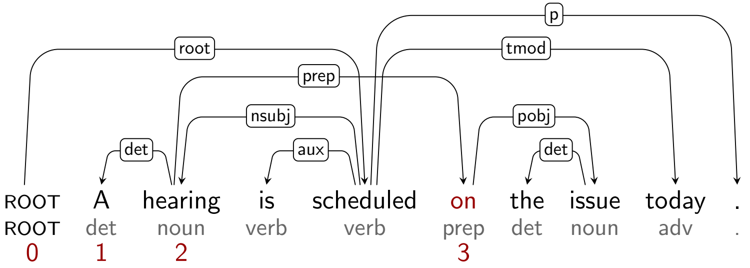
Projectivity and Word Order

- 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]



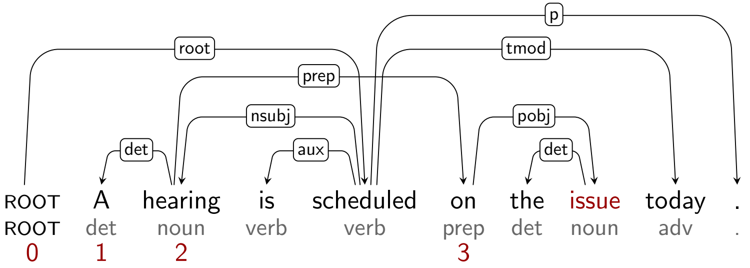
Projectivity and Word Order

- 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]



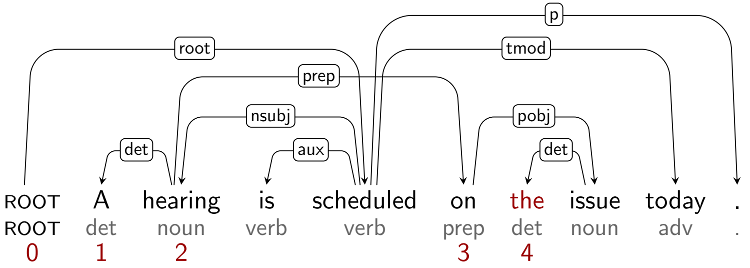
Projectivity and Word Order

- 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]



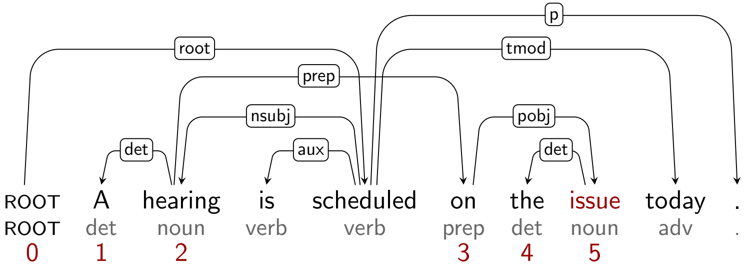
Projectivity and Word Order

- 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]



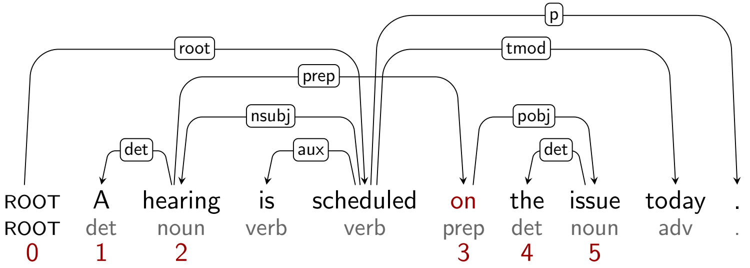
Projectivity and Word Order

- 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]



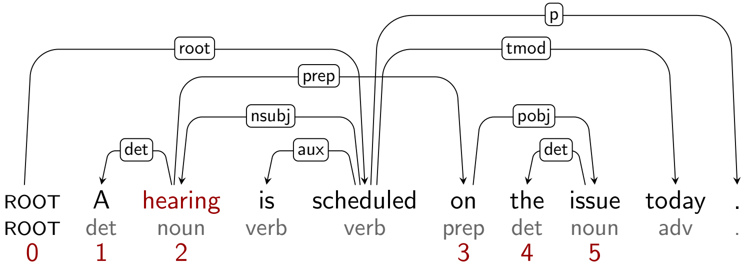
Projectivity and Word Order

- 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]



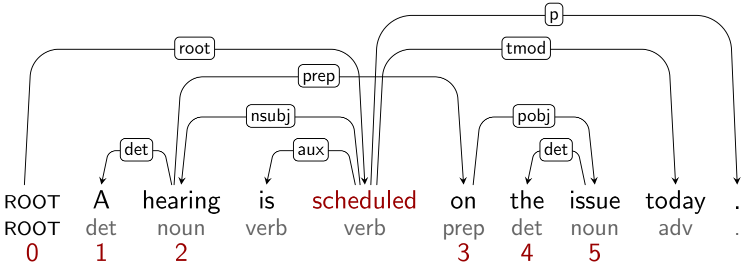
Projectivity and Word Order

- 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]



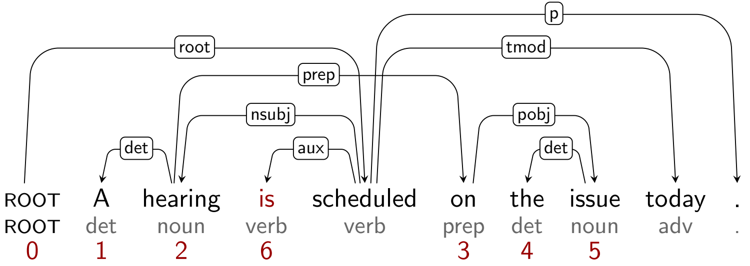
Projectivity and Word Order

- 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]



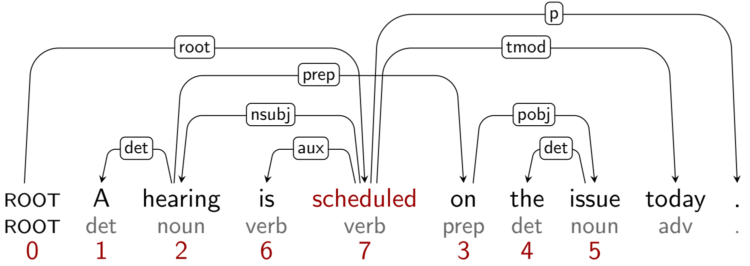
Projectivity and Word Order

- 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]



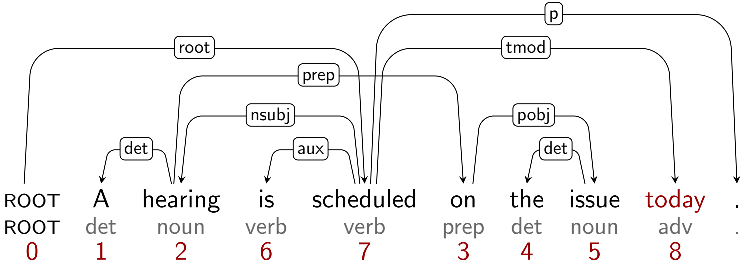
Projectivity and Word Order

- 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]



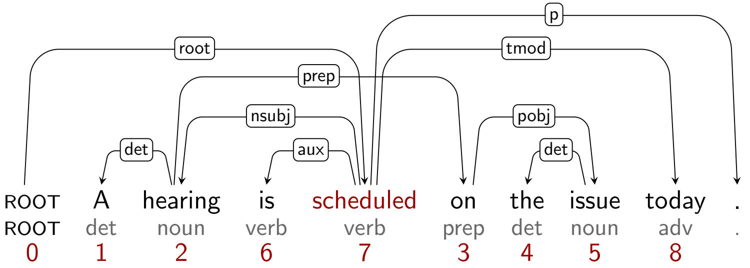
Projectivity and Word Order

- 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]



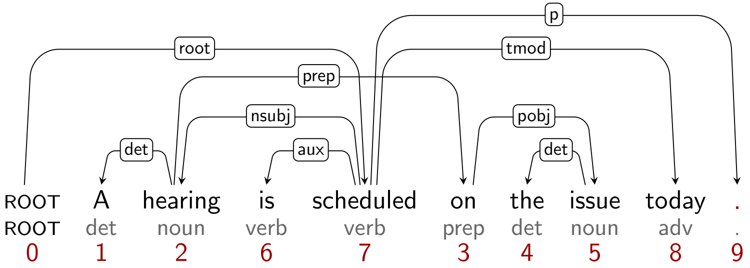
Projectivity and Word Order

- 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:	$([\ ], [w_0, w_1, \ldots, w_n], \{\ \})$	$(w_0 = \text{ROOT})$		
Terminal:	$([0], [\ ], A)$			
Shift:	$(S, w_i \vert B, A)$	$\Rightarrow$	$(S \vert w_i, B, A)$	
Right-Arc(l):	$(S \vert w_i \vert w_j, B, A)$	$\Rightarrow$	$(S \vert w_i, B, A \cup \{(w_i, l, w_j)\})$	
Left-Arc(l):	$(S \vert w_i \vert w_j, B, A)$	$\Rightarrow$	$(S \vert w_j, B, A \cup \{(w_j, l, w_i)\})$	$i \neq 0$
Swap:	$(S \vert w_i \vert w_j, B, A)$	$\Rightarrow$	$(S \vert w_j, w_i \vert B, A)$	$0 < i < j$

Transition System for Online Reordering

Configuration:	(S, B, A)	$[S = \text{Stack},\ B = \text{Buffer},\ A = \text{Arcs}]$		
Initial:	$([\ ], [w_0, w_1, \ldots, w_n], \{\ \})$	$(w_0 = \text{ROOT})$		
Terminal:	$([0], [\ ], A)$			
Shift:	$(S, w_i\vert B, A)$	$\Rightarrow$	$(S\vert w_i, B, A)$	
Right-Arc(l):	$(S\vert w_i\vert w_j, B, A)$	$\Rightarrow$	$(S\vert w_i, B, A \cup \{(w_i, l, w_j)\})$	
Left-Arc(l):	$(S\vert w_i\vert w_j, B, A)$	$\Rightarrow$	$(S\vert w_j, B, A \cup \{(w_j, l, w_i)\})$	$i \neq 0$
Swap:	$(S\vert w_i\vert w_j, B, A)$	$\Rightarrow$	$(S\vert w_j, w_i\vert B, A)$	$0 < i < j$

- 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

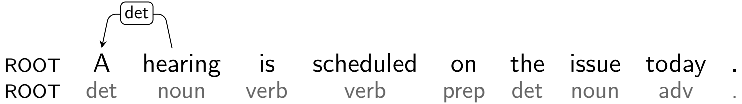
[ROOT, hearing]$_S$ [is, scheduled, on, the, issue, today, .]$_B$

det

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

Example Transition Sequence

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



Example Transition Sequence

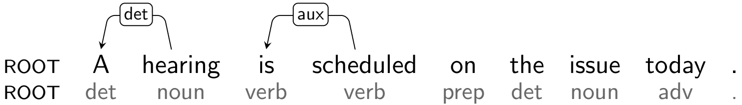
$[$ROOT, hearing, is, scheduled$]_S$ $[$on, the, issue, today, .$]_B$

det

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

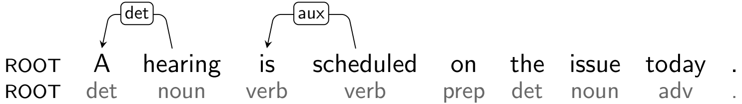
Example Transition Sequence

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



Example Transition Sequence

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



Example Transition Sequence

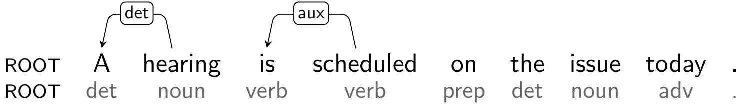
$[\text{ROOT, hearing, scheduled, on, the}]_S$ $[\text{issue, today, .}]_B$

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

det aux

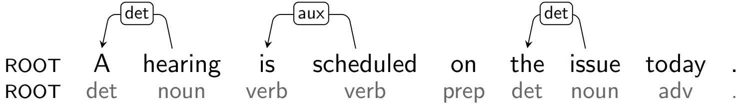
Example Transition Sequence

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



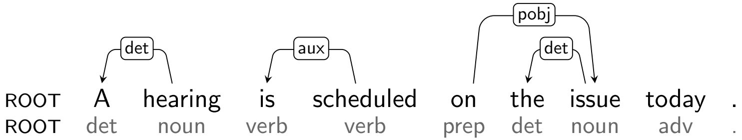
Example Transition Sequence

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



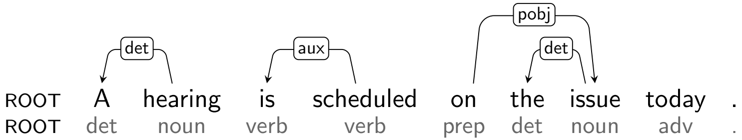
Example Transition Sequence

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



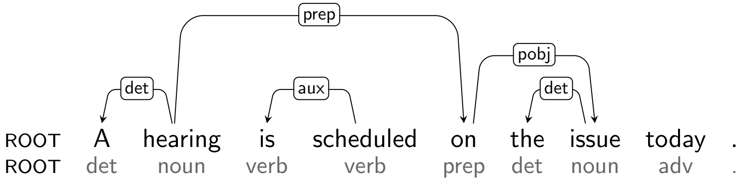
Example Transition Sequence

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



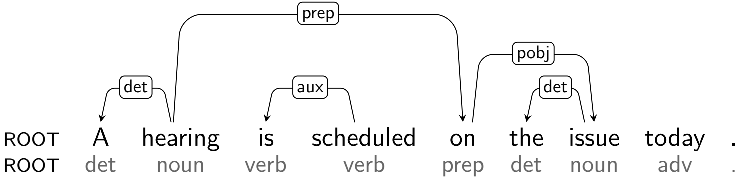
Example Transition Sequence

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



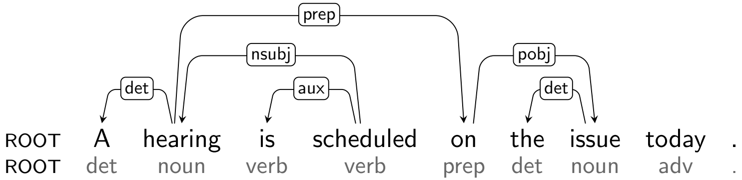
Example Transition Sequence

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



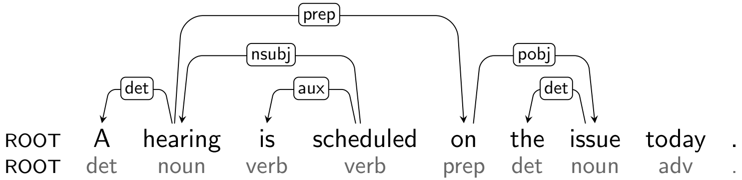
Example Transition Sequence

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



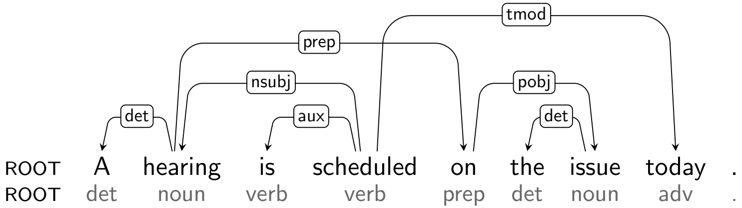
Example Transition Sequence

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



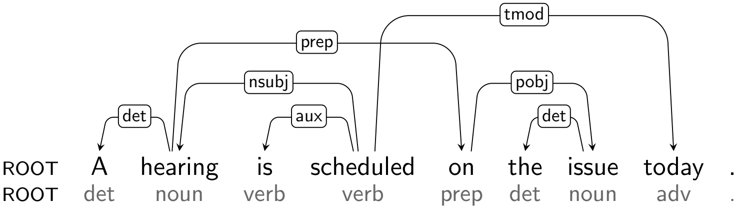
Example Transition Sequence

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



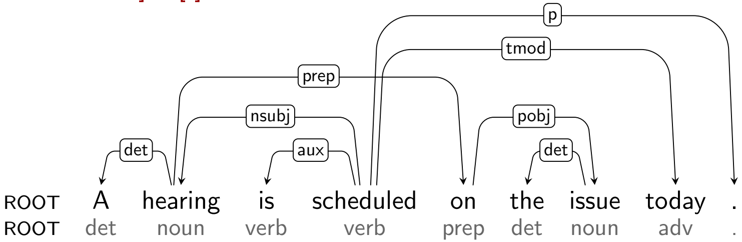
Example Transition Sequence

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



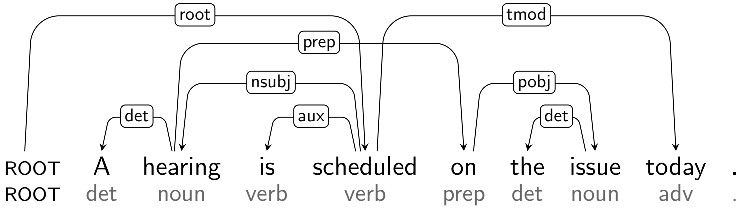
Example Transition Sequence

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



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

	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]

Conclusion

- Graph-based and transition-based parsing have complementary strengths and weaknesses
- Many recent developments can be understood in this light:
  - **Graph-based:** Increase feature scope (higher order models) while keeping learning and inference tractable
  - **Transition-based:** Improve learning and inference (beam search, dynamic oracles) without sacrificing efficiency
- Convergence: global learning, rich features, heuristic search
- And then there is this thing called deep learning ...

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.
- 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.
- 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.
- Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.
- Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976.

- 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.
- Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *Proceedings of the Conference on Computational Linguistics (COLING)*, pages 785–796.
- 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.
- 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, 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. 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.
- 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.

- 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.