Recent Advances in Dependency Parsing

Tutorial, EACL, April 27th, 2014

Ryan McDonald\textsuperscript{1} \hspace{1cm} Joakim Nivre\textsuperscript{2}

\textsuperscript{1}Google Inc., USA/UK
E-mail: ryanmcd@google.com

\textsuperscript{2}Uppsala University, Sweden
E-mail: joakim.nivre@lingfil.uu.se
Overview of the Tutorial

- Introduction to Dependency Parsing (Joakim)
- Graph-based parsing post-2008 (Ryan)
- Transition-based parsing post-2008 (Joakim)
- Summary and final thoughts (Ryan)
Transition-Based Dependency Parsing

Configuration: \((S, B, A)\)
Initial: \([[ ], [0, 1, \ldots, n], \{ \} \])
Terminal: \((S, [ ], A)\)
Shift: \((S, i|B, A) \Rightarrow (S|i, B, A)\)
Reduce: \((S|i, B, A) \Rightarrow (S, B, A)\)
Right-Arc\((k)\): \((S|i,j|B, A) \Rightarrow (S|i,j, B, A \cup \{(i,j,k)\})\)
Left-Arc\((k)\): \((S|i,j|B, A) \Rightarrow (S,j|B, A \cup \{(j,i,k)\})\)
Overview

- Improved learning and inference
  - Beam search and structured prediction
  - Dynamic programming
  - Easy-first parsing
  - Dynamic oracles
- Non-projective parsing
  - Online reordering
  - Multiplanar parsing
- Joint morphological and syntactic analysis
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: $(S, i | B, A) \Rightarrow (S | i, B, A)$
Reduce: $(S | i, B, A) \Rightarrow (S, B, A) \quad h(i, A)$
Right-Arc$(k)$: $(S | i, j | B, A) \Rightarrow (S | i | j, B, A \cup \{(i, j, k)\})$
Left-Arc$(k)$: $(S | i, j | B, A) \Rightarrow (S, j | B, A \cup \{(j, i, k)\}) \quad \neg h(i, A) \land i \neq 0$

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

[ROOT]s [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

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

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

\[
\text{adj noun verb adj noun prep adj noun .}
\]
Example Transition Sequence

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

\begin{align*}
\text{ROOT} & \quad \text{Economic} \\
\text{adj} & \quad \text{news} \\
\text{noun} & \quad \text{had} \\
\text{verb} & \quad \text{little} \\
\text{effect} & \quad \text{on} \\
\text{prep} & \quad \text{financial} \\
\text{adj} & \quad \text{markets} \\
\text{noun} & \quad .
\end{align*}
Example Transition Sequence

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

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

Example Transition Sequence

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

ROOT Economic news had little effect on financial markets .

amod nsubj
adj noun verb adj noun prep adj noun .
Example Transition Sequence

\[ \text{ROOT, had} \rightarrow \text{little, effect, on, financial, markets, .} \]

\[
\begin{array}{c}
\text{root} \\
\text{amod} \\
\text{nsubj}
\end{array}
\]

ROOT Economic news had little effect on financial markets .
Example Transition Sequence

\[
\text{[ROOT, had, little]}_S \quad \text{[effect, on, financial, markets, .]}_B
\]
Example Transition Sequence

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

\[
\text{ROOT, had, effect}_S \quad \text{on, financial, markets, .}_B
\]
Example Transition Sequence

\[[\text{ROOT, had, effect, on}]_S \quad [\text{financial, markets, .}]_B\]
Example Transition Sequence

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

\[
\begin{array}{c}
\text{root} \\
\downarrow \\
\text{amod} \\
\text{nsubj} \\
\text{ROOT} \\
\text{Economic} \\
\text{adj} \\
\text{news} \\
\text{noun} \\
\text{had} \\
\text{verb} \\
\text{little} \\
\text{adj} \\
\text{effect} \\
\text{noun} \\
\text{on} \\
\text{prep} \\
\text{financial} \\
\text{markets} \\
\text{.} \\
\end{array}
\]
Example Transition Sequence

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

Recent Advances in Dependency Parsing
Example Transition Sequence

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

- **ROOT**
- **Economic**
  - adj
- **news**
  - noun
- **had**
  - verb
- **little**
  - adj
- **effect**
  - noun
- **on**
  - prep
- **financial**
  - adj
- **markets**
  - noun
Example Transition Sequence

\[
[\text{ROOT, had, effect, on}]_S \quad [.]_B
\]
Example Transition Sequence

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

- **ROOT**
- **Economic**
  - **adj**
  - **news**
  - **noun**
  - **had**
  - **verb**
- **little**
  - **adj**
  - **effect**
  - **noun**
- **on**
  - **prep**
- **financial**
  - **adj**
  - **markets**
  - **noun**
Example Transition Sequence

\[
\text{[ROOT, had, .]}_S \quad [ \quad ]_B
\]
Arc-Standard Transition System [Nivre 2004]

Configuration: \((S, B, A)\) \([S = \text{Stack}, B = \text{Buffer}, A = \text{Arcs}]\)

Initial: \((\emptyset, [0, 1, \ldots, n], \emptyset)\)

Terminal: \((\emptyset, \emptyset, A)\)

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\((k)\): \((S|i|j, B, A) \Rightarrow (S|j, B, A \cup \{(j, i, k)\})\) for \(i \neq 0\)
Greedy Inference

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

  \[
  \text{Parse}(w_1, \ldots, w_n)
  \]

  1. $c \leftarrow ([], [0, 1, \ldots, n]_B, \{\})$
  2. while $B_c \neq []$
  3. $t \leftarrow o(c)$
  4. $c \leftarrow t(c)$
  5. 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:
  \[ o(c) = \arg\max_t w \cdot f(c, t) \]
- History-based feature representation \( f(c, t) \)
- Weight vector \( w \) learned from treebank data
Feature Representation

- Features over input tokens relative to $S$ and $B$

Configuration

Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos($S_2$)</td>
<td>ROOT</td>
</tr>
<tr>
<td>pos($S_1$)</td>
<td>verb</td>
</tr>
<tr>
<td>pos($S_0$)</td>
<td>noun</td>
</tr>
<tr>
<td>pos($B_0$)</td>
<td>prep</td>
</tr>
<tr>
<td>pos($B_1$)</td>
<td>adj</td>
</tr>
<tr>
<td>pos($B_2$)</td>
<td>noun</td>
</tr>
</tbody>
</table>
Feature Representation

- Features over input tokens relative to $S$ and $B$

**Configuration**

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

**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 $A$
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**

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

**Features**

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

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

Configuration

Features

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

- 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 transition-based parsing has been how to improve learning and inference, while maintaining high efficiency and rich feature models
Beam Search

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

  Parse($w_1, \ldots, w_n$)
  
  1. $\text{Beam} \leftarrow \{([,]_s,[0,1,\ldots,n]_B,\{\})\}$
  2. While $\exists c \in \text{Beam} [B_c \not= []]$
  3.   For each $c \in \text{Beam}$
  4.       For each $t$
  5.         Add($t(c), \text{NewBeam}$)
  6.   $\text{Beam} \leftarrow \text{Top}(k, \text{NewBeam})$
  7. Return $G = (\{0,1,\ldots,n\}, A_{\text{Top}(1,\text{Beam})})$

- Note:
  - $\text{Score}(c_0, \ldots, c_m) = \sum_{i=1}^{m} w \cdot f(c_{i-1}, t_i)$
  - 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:

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

- 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]
## Even Richer Feature Models

<table>
<thead>
<tr>
<th></th>
<th>ZPar</th>
<th>Malt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>92.18</td>
<td>89.37</td>
</tr>
<tr>
<td>+distance</td>
<td>+0.07</td>
<td>−0.14</td>
</tr>
<tr>
<td>+valency</td>
<td>+0.24</td>
<td>0.00</td>
</tr>
<tr>
<td>+unigrams</td>
<td>+0.40</td>
<td>−0.29</td>
</tr>
<tr>
<td>+third-order</td>
<td>+0.18</td>
<td>0.00</td>
</tr>
<tr>
<td>+label set</td>
<td>+0.07</td>
<td>+0.06</td>
</tr>
<tr>
<td>Extended</td>
<td>93.14</td>
<td>89.00</td>
</tr>
</tbody>
</table>

[Zhang and Nivre 2011, Zhang and Nivre 2012]

- Adding graph-based features may require special techniques
  [Zhang and Clark 2008, Bohnet and Kuhn 2012]
Dynamic Programming

- If beam search reduces search errors, why not exact inference?
- Dynamic programming for transition-based parsers:
  - Using a graph-structured stack [Huang and Sagae 2010]
  - Using push-computations [Kuhlmann et al. 2011]
- Adds constraints on feature representations
Improved Learning and Inference

Deduction System for Arc-Eager Parsing

**Items:** \([i^b, j] \iff (S, i|B, A) \Rightarrow^* (S|i, j|B', A')\)

\[
b = \begin{cases} 
1 & \text{if } [h(i) \in A'] \\
0 & \text{otherwise}
\end{cases}
\]

**Goal:** \([0^0, n + 1]\)

**Axiom:** \([0^0, 1]\)

**Rules:**
- **Shift:** \([i^b, j] \Rightarrow [j^0, j + 1]\)
- **Reduce:** \([i^b, m] \land [m^1, j] \Rightarrow [i^b, j]\)
- **Right-Arc:** \([i^b, j] \Rightarrow [j^1, j + 1]\)
- **Left-Arc:** \([i^b, m] \land [m^0, j] \Rightarrow [i^b, j]\)

[Kuhlmann et al. 2011]
Theory and Practice

- Theoretical results:
  - Arc-eager parsing in $O(n^3)$ time (cf. Eisner)
  - Arc-standard parsing in $O(n^5)$ time (cf. CKY)

- In practice:
  - Results hold only for very simplistic feature models
  - Practical implementations use beam search
  - Benefits from ambiguity packing

[Figure 6: DP searches over a forest of k-best lists with higher oracles, while non-DP only explores b trees allowed in the beam (b = 16 here).]

[Figure 5: Speed comparisons between DP and non-DP, with beam size b = 16 for DP and b = 64 for non-DP. Speed is measured by avg. parsing time (secs) per sentence length.]

[Huang and Sagae 2010]
The Need for Speed

- Beam search helps but slows down the parser
- Dynamic programming in addition constrains feature model
- What can we do to maintain the highest speed?
  - Easy-first parsing – give up left-to-right incremental search
  - Dynamic oracles – learn how to recover from errors
- These two ideas can be combined
Easy-First Non-Directional Parsing

- Process dependencies from easy to hard (not left to right) and from local to global (bottom up) [Goldberg and Elhadad 2010]

**Configuration:**  \((L, A) \quad [L = \text{List}, \ A = \text{Arcs}]\)

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

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

**Attach-Right** \((i, k)\):  
\[
([v_1, \ldots, v_m], A) \Rightarrow ([v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m], A \cup \{(v_{i+1}, v_i, k)\})
\]

**Attach-Left** \((i, k)\):  
\[
([v_1, \ldots, v_m], A) \Rightarrow ([v_1, \ldots, v_i, v_{i+2}, \ldots, v_m], A \cup \{(v_i, v_{i+1}, k)\})
\]
Parsing Algorithm

Given an oracle \( o \) that selects the highest-confidence transition \( o(c) \), parsing is deterministic:

\[
\text{Parse}(w_1, \ldots, w_n)
\]

1. \( c \leftarrow ([0, 1, \ldots, n], \{ \}) \)
2. \( \text{while } \text{length}(L_c) > 1 \)
3. \( t \leftarrow o(c) \)
4. \( c \leftarrow t(c) \)
5. \( \text{return } G = ([0, 1, \ldots, n], A_c) \)

- Number of possible transitions grows with sentence length
- Parsing in \( O(n \log n) \) time with priority heap
**Parsing Example**

(1) ATTACHRIGHT(2)

```
(2) ATTACHRIGHT(1)

(3) ATTACHRIGHT(1)

(4) ATTACHLEFT(2)

(5) ATTACHLEFT(1)

(6)
```

[Goldberg and Elhadad 2010]
Oracles Revisited

- How do we train the easy-first parser?
- Recall our training procedure for greedy parsers:
  - Reconstruct oracle transition sequence for each sentence
  - Construct training data set \( D = \{(c, t) \mid o(c) = t\} \)
  - Maximize accuracy of local predictions \( o(c) = t \)
- Presupposes a unique optimal transition for each configuration
  - Does not make sense for the easy-first parser
  - Turns out to be a bad idea in general
Online Learning with a Conventional Oracle

Learn($\{T_1, \ldots, T_N\}$)
1. $w \leftarrow 0.0$
2. for $i$ in 1..$K$
3.   for $j$ in 1..$N$
4.     $c \leftarrow ([], [0, 1, \ldots, n_j], \{\})$
5.     while $B_c \neq []$
6.        $t^* \leftarrow \text{argmax}_t w \cdot f(c, t)$
7.        $t_o \leftarrow o(c, T_i)$
8.        if $t^* \neq t_o$
9.           $w \leftarrow w + f(c, t_o) - f(c, t^*)$
10.       $c \leftarrow t_o(c)$
11. return $w$
Online Learning with a Conventional Oracle

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

1. \textbf{w} \leftarrow 0.0
2. \textbf{for} \ i \textbf{ in } 1..K
3. \quad \textbf{for} \ j \textbf{ in } 1..N
4. \quad \quad \textbf{c} \leftarrow ([], [0, 1, \ldots, n_j], \{ \})
5. \quad \textbf{while} \ B_c \neq []
6. \quad \quad \textbf{t}^* \leftarrow \text{argmax}_t \ \textbf{w} \cdot \textbf{f}(\textbf{c}, t)
7. \quad \quad \textbf{t}_o \leftarrow o(\textbf{c}, T_i)
8. \quad \quad \textbf{if} \ t^* \neq t_o
9. \quad \quad \quad \textbf{w} \leftarrow \textbf{w} + \textbf{f}(\textbf{c}, t_o) - \textbf{f}(\textbf{c}, t^*)
10. \quad \quad \quad \textbf{c} \leftarrow \textbf{t}_o(c)
11. \quad \textbf{return} \ \textbf{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, .]

Recent Advances in Dependency Parsing 30(54)
Oracle Parse

Transitions:  SH-RA

Stack
[ROOT, He]

Buffer
[sent, her, a, letter, .]

Arcs
ROOT → sent

Recent Advances in Dependency Parsing
**Oracle Parse**

**Transitions:** SH-RA-LA

**Stack**

- [ROOT]

**Buffer**

- [sent, her, a, letter, .]

**Arcs**

- ROOT $\rightarrow$ sent
- He $\leftrightarrow$ sent

```
ROOT
  ↓
root
  ↓
nsubj
  ↓
He
  ↓
pron
  ↓
ROOT

  ↓
verb
  ↓
iobj
  ↓
her
  ↓
det
  ↓
a
  ↓
dobj
  ↓
letter
  ↓
p
  ↓
.
```
Oracle Parse

Transitions: SH-RA-LA-SH

Stack
[ROOT, sent]

Buffer
[her, a, letter, .]

Arcs
ROOT $\rightarrow$ sent
He $\leftarrow$ sent

Recent Advances in Dependency Parsing 30(54)
Oracle Parse

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

**Stack**

\[
\text{[ROOT, sent, her]}
\]

**Buffer**

\[
\text{[a, letter, .]}
\]

**Arcs**

\[
\text{ROOT} \xrightarrow{\text{root}} \text{sent}
\]

\[
\text{He} \xleftarrow{\text{sbj}} \text{sent}
\]

\[
\text{sent} \xrightarrow{\text{iobj}} \text{her}
\]
Oracle Parse

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

Stack  Buffer
[ROOT, sent, her, a]  [letter, .]

Arcs
ROOT$\rightarrow$ sent
He$\leftarrow$sent
sent$\rightarrow$her

He sent her a letter.

Root

nsubj

ROOT

verb

sent

ROOT

pron

sent

ROOT

det

a

n

noun
Oracle Parse

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

Stack
[ROOT, sent, her]

Buffer
[letter, .]

Arcs
ROOT → sent
He ← sent
sent → her
a ← letter

ROOT
He
sent
her
a
letter

ROOT
pron
verb
pron
det
noun
Oracle Parse

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

Stack
[ROOT, sent]

Buffer
[letter, .]

Arcs
ROOT → sent
He ← sent
sent → her
a ← letter

He sent her a letter.
Oracle Parse

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

Stack

Buffer

[ROOT, sent, letter] [.]

Arcs

ROOT \rightarrow sent
He \leftarrow sent
sent \rightarrow iobj her
a \leftarrow det letter
sent \rightarrow dobj letter
Oracle Parse

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

**Stack**

[ROOT, sent]

**Buffer**

[.]

**Arcs**

ROOT $\rightarrow$ sent

He $\leftarrow$ sent

sent $\leftarrow$ her

a $\leftarrow$ letter

sent $\rightarrow$ letter
Oracle Parse

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

**Stack**

[ROOT, sent, .]

**Buffer**

[ ]

**Arcs**

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

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

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

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

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

sent $\xrightarrow{\text{p}}$ .
Non-Determinism

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

Stack
[ROOT, sent, her]

Buffer
[a, letter, .]

Arcs
ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{\text{sbj}}$ sent
sent $\xrightarrow{\text{iobj}}$ her

He sent her a letter.

ROOT $\xrightarrow{\text{pron}}$ pron
sent $\xrightarrow{\text{verb}}$ verb
him $\xrightarrow{\text{pron}}$ pron
a $\xrightarrow{\text{det}}$ det
letter $\xrightarrow{\text{noun}}$ noun
Non-Determinism

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

**Stack**
[ROOT, sent]

**Buffer**
[a, letter, .]

**Arcs**
ROOT → sent
He ← sent
sent → her

She sent him a letter.

Recent Advances in Dependency Parsing
Non-Determinism

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

Stack: [ROOT, sent, a]  Buffer: [letter, .]

Arcs:
- ROOT → sent
- He ← sent
- sent → her

She sent him a letter.

Recent Advances in Dependency Parsing
Non-Determinism

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

Stack
[ROOT, sent]

Buffer
[letter, .]

Arcs
ROOT \xrightarrow{\text{root}} \text{sent}
He \xleftarrow{\text{sbj}} \text{sent}
\text{sent} \xrightarrow{\text{iobj}} \text{her}
a \xleftarrow{\text{det}} \text{letter}
Non-Determinism

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

Stack
[ROOT, sent, letter] [.]

Buffer

Arcs
ROOT → sent
He ← sent
sent → her
a ← letter
sent → letter

ROOT
pron
verb
pron
det
noun

She sent him a letter.

Recent Advances in Dependency Parsing
Non-Determinism

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

Stack
[ROOT, sent]

Buffer
[.]

Arcs
ROOT → sent
He → sent
sent → her
a → letter
sent → letter
Non-Determinism

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

Stack
- [ROOT, sent, .]

Buffer
- [ ]

Arcs
- \( \text{ROOT} \xrightarrow{\text{root}} \text{sent} \)
- He \( \xleftarrow{\text{sbj}} \text{sent} \)
- \( \text{sent} \xrightarrow{\text{iobj}} \text{her} \)
- a \( \xleftarrow{\text{det}} \text{letter} \)
- \( \text{sent} \xrightarrow{\text{dobj}} \text{letter} \)
- \( \text{sent} \xrightarrow{p} . \)
Non-Optimality

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

Transitions: SH-RA-LA-SH

Stack
[ROOT, sent]

Buffer
[her, a, letter, .]

Arcs
ROOT → sent
He ← sent

Recent Advances in Dependency Parsing 32(54)
Non-Optimality

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

Transitions: SH-RA-LA-SH-SH

Stack
[ROOT, sent, her]

Buffer
[a, letter, .]

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

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

p

root

nsubj

ROOT She pron sent verb

iobj

him pron

dobj

da det noun

sent
Non-Optimality

Improved Learning and Inference


Stack: [ROOT, sent, her, a]

Buffer: [letter, .]

Arcs:
ROOT \rightarrow \text{sent}
He \leftrightarrow \text{sent}

Recent Advances in Dependency Parsing 32(54)
Non-Optimality

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

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

Stack

Buffer

Arcs

He $\text{sbj}$ sent
a $\text{det}$ letter

She $\text{nsubj}$ sent $\text{iobj}$ him $\text{dobj}$ a $\text{det}$ letter $\text{p}$
Non-Optimality

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


Stack

Buffer

Arcs

ROOT → sent
He ⇐ subj sent
a ⇐ det letter
Non-Optimality

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


Stack  Buffer  Arcs

[ROOT, sent, letter, .]  [ ]  ROOT → sent
He ← sent
He sent
She sent him
She sent a
She sent a letter

Recent Advances in Dependency Parsing
Non-Optimality

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

**Stack**
- [ROOT, sent, her]

**Buffer**
- [letter, .]

**Arcs**
- \(\text{ROOT} \rightarrow \text{sent}\)
- \(\text{He} \leftarrow \text{sent}\)
- \(\text{a} \leftarrow \text{letter}\)

**Diagram:**
- **Stack:** [ROOT, sent, her]
- **Buffer:** [letter, .]
- **Arcs:**
  - \(\text{ROOT} \rightarrow \text{sent}\)
  - \(\text{He} \leftarrow \text{sent}\)
  - \(\text{a} \leftarrow \text{letter}\)
Non-Optimality

Transitions:

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


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

Stack

Buffer

Arcs

[ROOT, sent]

[letter, .]

ROOT → sent

He → subj sent

ROOT → sent

a → det letter

ROOT → sent

her → ? letter

ROOT

root

nsubj

ROOT

She

pron

sent

verb

ROOT

him

pron

a

det

letter

.}

noun

.}

Recent Advances in Dependency Parsing
Non-Optimality

Transitions:

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

Stack

[ROOT, sent, letter]

Buffer

[.]

Arcs

ROOT $\rightarrow$ sent
He $\leftrightarrow$ sent
a $\leftrightarrow$ letter
her $\leftrightarrow$ letter
sent $\rightarrow$ letter
Non-Optimality


[3/6]

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

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

Stack

[ROOT, sent]

Buffer

[.]

Arcs

ROOT → sent

He → sent

a → letter

her → letter

sent → letter
Non-Optimality

Transitions:

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


Stack

[ROOT, sent, .]

Buffer

[ ]

Arcs

ROOT $\rightarrow$ sent

He $\leftrightarrow$ sent

a $\leftrightarrow$ letter

her $\leftrightarrow$ letter

sent $\rightarrow$ letter

sent $\rightarrow$. 

She sent him a letter.

Recent Advances in Dependency Parsing
Dynamic Oracles

- **Optimality:**
  - A transition is optimal if the best tree remains reachable
  - Best tree = arg\ min_{T', \mathcal{L}(T, T')}

- **Oracle:**
  - Boolean function \( o(c, t, T) = \text{true} \) 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?
Reachability for Arcs and Trees

- **Arc reachability:**
  - An arc $w_i \rightarrow w_j$ is reachable in $c$ iff $w_i \rightarrow w_j \in A_c$, or $w_i \in S_c \cup B_c$ and $w_j \in B_c$ (same for $w_i \leftarrow w_j$)

- **Tree reachability:**
  - A (projective) tree $T$ is reachable in $c$ iff every arc in $T$ is reachable in $c$

- **Arc-decomposable systems** [Goldberg and Nivre 2013]:
  - Tree reachability reduces to arc reachability
  - Holds for some transition systems but not all
    - Arc-eager and easy-first are arc-decomposable
    - Arc-standard is **not** decomposable
Oracles for Arc-Decomposable Systems

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

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

**Arc-Eager**

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

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

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

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

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\})

1. \( w \leftarrow 0.0 \)
2. \textbf{for} \( i \) in 1..K
3. \hspace{1em} \textbf{for} \( j \) in 1..N
4. \hspace{2em} \( c \leftarrow ([], [w_1, \ldots, w_n], B, \{\}) \)
5. \hspace{1em} \textbf{while} \( B_c \neq [\] \)
6. \hspace{2em} \( t^* \leftarrow \arg\max_t w \cdot f(c, t) \)
7. \hspace{2em} \( t_o \leftarrow \arg\max_{t \in \{t \mid o(c, t, T_i)\}} w \cdot f(c, t) \)
8. \hspace{2em} \textbf{if} \( t^* \neq t_o \)
9. \hspace{2em} \hspace{1em} \( w \leftarrow w + f(c, t_o) - f(c, t^*) \)
10. \hspace{2em} \hspace{1em} \( c \leftarrow \text{choice}(t_o(c), t^*(c)) \)
11. \textbf{return} \( w \)
Online Learning with a Dynamic Oracle

\[
\text{Learn}(\{T_1, \ldots, T_N\})
\]

1. \( w \leftarrow 0.0 \)
2. \( \text{for } i \text{ in } 1..K \)
3. \( \text{for } j \text{ in } 1..N \)
4. \( c \leftarrow ([S, [w_1, \ldots, w_n]_B, \{\}) \)
5. \( \text{while } B_c \neq [ ] \)
6. \( t^* \leftarrow \text{argmax}_t w \cdot f(c, t) \)
7. \( t_o \leftarrow \text{argmax}_{t \in \{t \mid o(c, t, T_i)\}} w \cdot f(c, t) \)
8. \( \text{if } t^* \neq t_o \)
9. \( w \leftarrow w + f(c, t_o) - f(c, t^*) \)
10. \( c \leftarrow \text{choice}(t_o(c), t^*(c)) \)
11. \( \text{return } w \)

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

[Goldberg and Nivre 2012]
Ambiguity and Exploration

- Lessons from dynamic oracles:
  - Do not hide spurious ambiguity from the parser – exploit it
  - Let the parser explore the consequences of its own mistakes

- Related work:
  - Bootstrapping [Choi and Palmer 2011]
  - Selectional branching [Choi and McCallum 2013]
  - Non-monotonic parsing [Honnibal et al. 2013]
  - Dynamic parsing strategy [Sartorio et al. 2013]
Summary: Learning and Inference

- Beam search and structured prediction:
  - Explores a larger search space at training and parsing time
  - Can be combined with dynamic programming
- Dynamic oracles:
  - Explores a larger search space only at training time
  - Can be combined with selectional branching and with flexible transition systems (easy-first, dynamic, non-monotonic)
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 – $O(n)$
- Also harder to construct dynamic oracles
  - Conjecture: arc-decomposability presupposes projectivity
Previous Approaches

- **Pseudo-projective parsing** [Nivre and Nilsson 2005]
  - Preprocess training data, post-process parser output
  - Approximate encoding with incomplete coverage
  - Relatively high precision but low recall

- **Extended arc transitions** [Attardi 2006]
  - Transitions that add arcs between non-adjacent subtrees
  - Upper bound on arc degree (limited to local relations)
  - Exact dynamic programming algorithm [Cohen et al. 2011]

- **List-based algorithms** [Covington 2001, Nivre 2007]
  - Consider all word pairs instead of adjacent subtrees
  - Increases parsing complexity (and training time)
  - Improved accuracy and efficiency by adding “projective transitions” [Choi and Palmer 2011]
Novel Approaches

▶ Online reordering [Nivre 2009, Nivre et al. 2009]:
  ▶ Reorder words during parsing to make tree projective
  ▶ Add a special transition for swapping adjacent words
  ▶ Quadratic time in the worst case but linear in the best case

▶ Multiplanar parsing [Gómez-Rodríguez and Nivre 2010]:
  ▶ Factor dependency trees into $k$ planes without crossing arcs
  ▶ Use $k$ stacks to parse each plane separately
  ▶ Linear time parsing with constant $k$
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 $T = (V, A, \prec)$, let the projective order $\prec_p$ be the order defined by an inorder traversal of $T$ with respect to $\prec$ [Veselá et al. 2004]

**Example:**

```
0 det hearing noun
1 det is verb
2 det scheduled verb
3 on prep the det
4 issue noun
5 today adv
6 .
```
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 $T = (V, A, \prec)$, let the projective order $\prec_p$ be the order defined by an inorder traversal of $T$ with respect to $\prec$ [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]

\[
\begin{array}{c}
\text{ROOT} \\
0
\end{array} \quad \begin{array}{c}
\text{A} \\
1
\end{array} \quad \begin{array}{c}
\text{hearing} \\
2
\end{array} \quad \begin{array}{c}
is \\
3
\end{array} \quad \begin{array}{c}
scheduled \\
4
\end{array} \quad \begin{array}{c}
on \\
5
\end{array} \quad \begin{array}{c}
\text{the} \\
6
\end{array} \quad \begin{array}{c}
\text{issue} \\
7
\end{array} \quad \begin{array}{c}
today \\
8
\end{array} \quad \begin{array}{c}
\text{.} \\
9
\end{array}
\]
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 \textit{projective order} $<_p$ be the order defined by an \textit{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 $T = (V, A, \prec)$, let the projective order $\prec_P$ be the order defined by an inorder traversal of $T$ with respect to $\prec$ [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]

```
ROOT  hearing is scheduled on the issue today.

ROOT  det noun verb verb prep det noun adv .
```

0 1 2 6 7 3 4 5 8 9
root

Recent Advances in Dependency Parsing
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 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 \textit{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($k$): $(S|i|j, B, A) \Rightarrow (S|j, B, A \cup \{(j, i, k)\}) \quad i \neq 0$

Swap: $(S|i|j, B, A) \Rightarrow (S|j, i|B, A) \quad 0 < i < j$
Non-Projective Parsing

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

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

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

[ROOT]₅ [A, hearing, is, scheduled, on, the, issue, today, .]₆

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

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

\[
\begin{array}{ccccccccc}
\text{ROOT} & \text{A} & \text{hearing} & \text{is} & \text{scheduled} & \text{on} & \text{the} & \text{issue} & \text{today} . \\
\text{ROOT} & \text{det} & \text{noun} & \text{verb} & \text{verb} & \text{prep} & \text{det} & \text{noun} & \text{adv} . \\
\end{array}
\]
Example Transition Sequence

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

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

\[
[\text{ROOT, hearing}]_S \quad [\text{is, scheduled, on, the, issue, today, .}]_B
\]
Example Transition Sequence

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

ROOT \quad A \quad \text{hearing} \quad \text{is} \quad \text{scheduled} \quad \text{on} \quad \text{the} \quad \text{issue} \quad \text{today} \quad .

ROOT \quad \text{det} \quad \text{noun} \quad \text{verb} \quad \text{verb} \quad \text{prep} \quad \text{det} \quad \text{noun} \quad \text{adv} \quad .
Example Transition Sequence

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

\[
\begin{array}{c}
\text{ROOT} \quad \text{A} \quad \text{hearing} \quad \text{is} \quad \text{scheduled} \quad \text{on} \quad \text{the} \quad \text{issue} \quad \text{today} \quad .
\end{array}
\]
Example Transition Sequence

\[ [\text{ROOT, hearing, scheduled}]_S \quad [\text{on, the, issue, today, .}]_B \]
Example Transition Sequence

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

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

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

\[
\begin{array}{ccccccccc}
\text{ROOT} & \text{A} & \text{hearing} & \text{is} & \text{scheduled} & \text{on} & \text{the} & \text{issue} & \text{today} & . \\
\text{ROOT} & \text{det} & \text{noun} & \text{verb} & \text{verb} & \text{prep} & \text{det} & \text{noun} & \text{adv} & . \\
\end{array}
\]
Example Transition Sequence

\[ \text{ROOT, hearing, scheduled, on, the, issue}_{S} \quad \text{today, .}_{B} \]

\[ \text{ROOT} \quad \text{A} \quad \text{hearing} \quad \text{is} \quad \text{scheduled} \quad \text{on} \quad \text{the} \quad \text{issue} \quad \text{today} \quad . \]

\[ \text{ROOT} \quad \text{det} \quad \text{noun} \quad \text{verb} \quad \text{verb} \quad \text{prep} \quad \text{det} \quad \text{noun} \quad \text{adv} \quad . \]
Example Transition Sequence

\[[\text{ROOT, hearing, scheduled, on, issue}]_S \ [\text{today, .}]_B\]
Example Transition Sequence

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

\[
\begin{array}{cccccc}
\text{ROOT} & \text{det} & \text{A} & \text{hearing} & \text{is} & \text{scheduled} \\
\text{ROOT} & \text{det} & \text{noun} & \text{verb} & \text{verb} & \text{on} \\
\text{ROOT} & \text{det} & \text{the} & \text{issue} & \text{today} & \text{adv} \\
\end{array}
\]
Example Transition Sequence

\[
\text{[ROOT, hearing, on]}_S \quad \text{[scheduled, today, .]}_B
\]
Example Transition Sequence

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

[ROOT, hearing]_S [scheduled, today, .]_B
Example Transition Sequence

\[ \text{ROOT, hearing, scheduled}_S \quad \text{today, .}_B \]
Example Transition Sequence

\[ [\text{ROOT, scheduled}]_S \quad [\text{today, .}]_B \]
Example Transition Sequence

\[
[\text{ROOT, scheduled, today}]_S \quad [.]_B
\]
Example Transition Sequence

\[ \text{[ROOT, scheduled]}_S \quad [.]_B \]
Example Transition Sequence

[ROOT, scheduled, .]_S [ ]_B

ROOT

A det hearing noun is verb scheduled verb on prep the det issue noun today adv .
Example Transition Sequence

\[ [\text{ROOT, scheduled}]_S \quad [\ ]_B \]
Example Transition Sequence

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

Recent Advances in Dependency Parsing
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

<table>
<thead>
<tr>
<th></th>
<th>Czech</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAS UAS</td>
<td>LAS UAS</td>
</tr>
<tr>
<td>Projective</td>
<td>80.8 86.3</td>
<td>86.2 88.5</td>
</tr>
<tr>
<td>Reordering</td>
<td>83.9 89.1</td>
<td>88.7 90.9</td>
</tr>
</tbody>
</table>

[Bohnet and Nivre 2012]
Multiplanarity

- Multiplanarity is based on the notion of planarity:
  - A dependency graph is planar if it has no crossing arcs
  - A dependency graph is k-planar if it can be decomposed into (at most) k planar graphs [Yli-Jyrä 2003]

- In most treebanks, well over 99% of the trees are at most 2-planar [Gómez-Rodríguez and Nivre 2010]

- We can parse k-planar graphs in linear time using k stacks
1-Planar Transition System

Configuration: \((S, B, A) \quad [S = \text{Stack}, \ B = \text{Buffer}, \ A = \text{Arcs}]\)

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

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

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

Reduce: \((S|i, B, A) \Rightarrow (S, B, A)\)

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

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

▶ Similar to the arc-eager system except:
  ▶ **Reduce** does not require popped node to have a head
  ▶ **Left-Arc/Right-Arc** do not affect \(S\) or \(B\)
2-Planar Transition System

Configuration: $(S_1, S_2, B, A) \quad [S_1 = \text{Stack 1, } S_2 = \text{Stack 2}]$

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

Terminal: $(S_1, S_2, [\ ], A)$

Shift: $(S_1, S_2, i|B, A) \Rightarrow (S_1|i, S_2|i, B, A)$

Reduce: $(S_1|i, S_2, B, A) \Rightarrow (S_1, S_2, B, A)$

Right-Arc$(k)$: $(S_1|i, S_2, j|B, A) \Rightarrow (S_1|i, S_2, j|B, A \cup \{(i, j, k)\}) \quad \neg h(j, A)$

Left-Arc$(k)$: $(S_1|i, S_2, j|B, A) \Rightarrow (S_1|i, S_2, j|B, A \cup \{(j, i, k)\}) \quad \neg h(i, A) \land i \neq 0$

Switch: $(S_1, S_2, B, A) \Rightarrow (S_2, S_1, B, A)$

- Similar to 1-planar system except:
  - Shift pushes a node to both stacks
  - Left-Arc/Right-Arc/Reduce only affect $S_1$
  - Switch swaps $S_1$ and $S_2$
Example Transition Sequence

\[ S_1 \quad [\text{ROOT, A, hearing, is, scheduled, on, the, issue, today, . }]_B \]

\[ S_2 \]

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

\[ [\text{ROOT}]_{S_1} \quad [\text{A, hearing, is, scheduled, on, the, issue, today, .}]_{B} \]

\[ [\text{ROOT}]_{S_2} \]

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

\[
[\text{ROOT, A}]_{S_1} \quad [\text{hearing, is, scheduled, on, the, issue, today, .}]_B
\]

\[
[\text{ROOT, A}]_{S_2}
\]
Example Transition Sequence

\[
\begin{align*}
\text{ROOT}, A & \quad _{S_1} \quad \text{[hearing, is, scheduled, on, the, issue, today, .]}_B \\
\text{ROOT}, A & \quad _{S_2}
\end{align*}
\]
Non-Projective Parsing

Example Transition Sequence

\[ \text{[ROOT]}_{S_1} \quad \text{[hearing, is, scheduled, on, the, issue, today, .]}_{B} \]

\[ \text{[ROOT, A]}_{S_2} \]

\[
\begin{array}{cccccccc}
\text{ROOT} & \text{A} & \text{hearing} & \text{is} & \text{scheduled} & \text{on} & \text{the} & \text{issue} & \text{today} \ . \\
\text{ROOT} & \text{det} & \text{noun} & \text{verb} & \text{verb} & \text{prep} & \text{det} & \text{noun} & \text{adv} \ . \\
\end{array}
\]
Example Transition Sequence

\[
\begin{align*}
\text{[ROOT, hearing]}_{S_1} & \quad \text{[is, scheduled, on, the, issue, today, .]}_{B} \\
\text{[ROOT, A, hearing]}_{S_2} & \quad \\
\end{align*}
\]

\[
\begin{align*}
\text{ROOT} & \quad \text{A} \quad \text{hearing} \quad \text{is} \quad \text{scheduled} \quad \text{on} \quad \text{the} \quad \text{issue} \quad \text{today} \quad . \\
\text{ROOT} & \quad \text{det} \quad \text{noun} \quad \text{verb} \quad \text{verb} \quad \text{prep} \quad \text{det} \quad \text{noun} \quad \text{adv} \quad .
\end{align*}
\]
Example Transition Sequence

\[ \text{ROOT, hearing, is} \]_{S_1} \quad \text{scheduled, on, the, issue, today, .} \]_{B}

\[ \text{ROOT, A, hearing, is} \]_{S_2}

\[
\begin{array}{ccccccc}
\text{ROOT} & \text{A} & \text{hearing} & \text{is} & \text{scheduled} & \text{on} & \text{the} & \text{issue} & \text{today} & \text{.} \\
\text{ROOT} & \text{det} & \text{noun} & \text{verb} & \text{verb} & \text{prep} & \text{det} & \text{noun} & \text{adv} & \text{.}
\end{array}
\]
Example Transition Sequence

\[ [\text{ROOT, hearing, is}]_{S_1} \quad [\text{scheduled, on, the, issue, today, .}]_{B} \]

\[ [\text{ROOT, A, hearing, is}]_{S_2} \]
Example Transition Sequence

\[ [\text{ROOT, hearing}]_{S_1} \quad [\text{scheduled, on, the, issue, today, .}]_{B} \]
\[ [\text{ROOT, A, hearing, is}]_{S_2} \]

Recent Advances in Dependency Parsing 50(54)
Example Transition Sequence

\[ [\text{ROOT, hearing}]_{S_1} \quad [\text{scheduled, on, the, issue, today, .}]_{B} \]

\[ [\text{ROOT, A, hearing, is}]_{S_2} \]
Example Transition Sequence

$[\text{ROOT}]_{S_1}$ $[\text{scheduled, on, the, issue, today, .}]_{B}$

$[\text{ROOT, A, hearing, is}]_{S_2}$

$\text{ROOT}$ $\text{det}$ $\text{noun}$ $\text{verb}$ $\text{verb}$ $\text{prep}$ $\text{det}$ $\text{noun}$ $\text{adv}$ $\text{.}$

Recent Advances in Dependency Parsing
Example Transition Sequence

\[ \text{ROOT}_{S_1} [ \text{scheduled, on, the, issue, today, .} ]_{B} \]

\[ \text{ROOT, A, hearing, is}_{S_2} \]
Example Transition Sequence

\[ [\text{ROOT, scheduled}]_{S_1} \quad [\text{on, the, issue, today, .}]_{B} \]

\[ [\text{ROOT, A, hearing, is, scheduled}]_{S_2} \]

Recent Advances in Dependency Parsing 50(54)
Example Transition Sequence

\[ \text{ROOT, A, hearing, is, scheduled}_{S_1} \quad \text{on, the, issue, today, .}_{B} \]

\[ \text{ROOT, scheduled}_{S_2} \]
Example Transition Sequence

\[ \text{ROOT, A, hearing, is}]_{S_1} \quad \text{[on, the, issue, today, .]}_{B} \]

\[ \text{ROOT, scheduled}]_{S_2} \]
Example Transition Sequence

\[\text{[ROOT, A, hearing]}_{S_1} \quad \text{[on, the, issue, today, .]}_{B}\]

\[\text{[ROOT, scheduled]}_{S_2}\]
Example Transition Sequence

\[ [\text{ROOT, A, hearing}]_{S_1} \quad [\text{on, the, issue, today, .}]_{B} \]
\[ [\text{ROOT, scheduled}]_{S_2} \]
Example Transition Sequence

\[ \text{ROOT, A, hearing, on}_S^1 \quad \text{the, issue, today, .}_B \]

\[ \text{ROOT, scheduled, on}_S^2 \]

Recent Advances in Dependency Parsing 50(54)
Example Transition Sequence

\[ [\text{ROOT, A, hearing, on, the}]_{S_1} \quad [\text{issue, today, .}]_{B} \]

\[ [\text{ROOT, scheduled, on, the}]_{S_2} \]
Example Transition Sequence

\[\text{ROOT, A, hearing, on, the}\]_{s_1} \quad \text{[issue, today, .]}_{B}

\[\text{ROOT, scheduled, on, the}\]_{s_2}
Example Transition Sequence

\[
\text{[ROOT, A, hearing, on]}_{S_1} \quad \text{[issue, today, .]}_{B}
\]

\[
\text{[ROOT, scheduled, on, the]}_{S_2}
\]
Example Transition Sequence

$[\text{ROOT, A, hearing, on}]_{S_1}$  $[\text{issue, today, .}]_{B}$

$[\text{ROOT, scheduled, on, the}]_{S_2}$
Example Transition Sequence

\[\text{ROOT, A, hearing, on, issue}\]_{S_1} \quad \text{[today, .]}_{B}

\[\text{ROOT, scheduled, on, the, issue}\]_{S_2}
Example Transition Sequence

\[
\begin{align*}
\text{ROOT, scheduled, on, the, issue}_{S_1} & \quad \text{today, .}_{B} \\
\text{ROOT, A, hearing, on, issue}_{S_2} & \\
\end{align*}
\]
Example Transition Sequence

[ROOT, scheduled, on, the]_{S_1} \quad \text{[today, .]}_{B}

[ROOT, A, hearing, on, issue]_{S_2}
Example Transition Sequence

\[
[\text{ROOT, scheduled, on}]_{S_1} \quad [\text{today, .}]_{B}
\]

\[
[\text{ROOT, A, hearing, on, issue}]_{S_2}
\]
Example Transition Sequence

\[
\text{[ROOT, scheduled]}_{S_1} \quad \text{[today, .]}_B
\]

\[
\text{[ROOT, A, hearing, on, issue]}_{S_2}
\]
Example Transition Sequence

\[
[\text{ROOT, scheduled}]_{S_1} \quad [\text{today, .}]_{B}
\]

\[
[\text{ROOT, A, hearing, on, issue}]_{S_2}
\]
Example Transition Sequence

\[
\text{[ROOT, scheduled, today]}_{S_1} \quad \text{[.]}_{B}
\]

\[
\text{[ROOT, A, hearing, on, issue, today]}_{S_2}
\]
Example Transition Sequence

\([\text{ROOT, scheduled}]_{S_1} \) \(\text{.} \) \(\text{[}.\]_B\)

\([\text{ROOT, A, hearing, on, issue, today}]_{S_2}\)
Example Transition Sequence

\[[\text{ROOT, scheduled}]_{S_1} \quad [\text{.}]_{B}\]

\[[\text{ROOT, A, hearing, on, issue, today}]_{S_2}\]
Example Transition Sequence

\[ [\text{ROOT, scheduled, . }]_{S_1} [ ]_{B} \]

\[ [\text{ROOT, A, hearing, on, issue, today, . }]_{S_2} \]
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: \((S, B, M, A) \quad [M = \text{Morphology}]\)

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\((k)\): \((S|i|j, B, M, A) \Rightarrow (S|j, B, M, A \cup \{(j, i, k)\})\) \quad i \neq 0

Swap: \((S|i|j, B, M, A) \Rightarrow (S|j, i|B, M, A)\) \quad 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\((k)\): \((S|i|j, B, M, A)\) \(\Rightarrow\) \((S|j, B, M, A \cup \{(j, i, k)\})\) \(\quad i \neq 0\)

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

- 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

<table>
<thead>
<tr>
<th></th>
<th>Czech</th>
<th>Finnish</th>
<th>German</th>
<th>Hungarian</th>
<th>Russian</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PM</td>
<td>LAS</td>
<td>PM</td>
<td>LAS</td>
<td>PM</td>
</tr>
<tr>
<td>Pipeline</td>
<td>93.0</td>
<td>83.1</td>
<td>88.8</td>
<td>79.9</td>
<td>89.1</td>
</tr>
<tr>
<td>Joint</td>
<td>94.4</td>
<td>83.5</td>
<td>91.6</td>
<td>82.5</td>
<td>91.2</td>
</tr>
</tbody>
</table>

[Bohnet et al. 2013]
Summary

- Transition-based parsing:
  - Efficient parsing using heuristic inference
  - Unconstrained history-based feature models
- Recent advances in synergy:
  - Beam search and structured prediction
  - Easy-first parsing and dynamic oracles
  - Online reordering for non-projective trees
  - Joint morphological and syntactic analysis
References and Further Reading


References and Further Reading


