Advanced Transition-Based Parsing Techniques

Joakim Nivre

Uppsala University
Linguistics and Philology

Based on previous tutorials with Ryan McDonald
Overall Plan

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

- Improved learning and inference
  - Beam search and structured prediction
  - Easy-first parsing
  - Dynamic oracles
- Non-projective parsing using online reordering
- Joint morphological and syntactic analysis
Transition-Based Parsing Trade-Off

- **Advantages:**
  - Highly efficient parsing — linear time complexity with constant time oracles and transitions
  - Rich history-based feature representations — no rigid constraints from inference algorithm

- **Drawback:**
  - Sensitive to search errors and error propagation due to greedy inference and local learning

- 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]:

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

- Note:
  - 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

[Graph showing the relationship between beam size (B) 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
Improved Learning and Inference

Precision by Dependency Length

[Graph showing precision by dependency length for MST, Malt, and ZPar, with data from 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]
The Need for Speed

- Beam search helps but slows down the parser
- 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:** \((L, A)\) \[L = \text{List}, A = \text{Arcs}\]

**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. while length($L_c$) > 1
  3. \hspace{1em} $t \leftarrow o(c)$
  4. \hspace{1em} $c \leftarrow t(c)$
  5. 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
Improved Learning and Inference

Parsing Example

(1) ATTACHRIGHT(2)

(2) ATTACHRIGHT(1)

(3) ATTACHRIGHT(1)

(4) ATTACHLEFT(2)

(5) ATTACHLEFT(1)

(6)

Figure 1: Parsing the sentence “a brown fox jumped with joy”. Rounded arcs represent possible actions.

relationally intensive sampling-based methods (Nakagawa, 2007). As a result, these models, while accurate, are slow (O(n^3) for projective, first-order models, higher polynomials for higher-order models, and worse for richer tree-feature models).

We propose a new category of dependency parsing algorithms, inspired by (Shen et al., 2007): non-directional easy-first parsing. This is a greedy, deterministic parsing approach, which relaxes the left-to-right processing order of transition-based parsing algorithms. By doing so, we allow the explicit incorporation of rich structural features derived from both sides of the attachment point, and implicitly take into account the entire previously derived structure of the whole sentence. This extension allows the incorporation of much richer features than those available to transition- and especially to graph-based parsers, and greatly reduces the locality of transition-based algorithm decisions. On the other hand, it is still a greedy, best-first algorithm leading to an efficient implementation.

We present a concrete O(n log n) parsing algorithm, which significantly outperforms state-of-the-art transition-based parsers, while closing the gap to graph-based parsers.

2 Easy-first parsing

When humans comprehend a natural language sentence, they arguably do it in an incremental, left-to-right manner. However, when humans consciously annotate a sentence with syntactic structure, they hardly ever work in fixed left-to-right order. Rather, they start by building several isolated constituents by making easy and local attachment decisions and only then combine these constituents into bigger constituents, jumping back-and-forth over the sentence and proceeding from easy to harder phenomena to analyze. When getting to the harder decisions a lot of structure is already in place, and this structure can be used in deciding a correct attachment.

Our parser follows a similar kind of annotation process: starting from easy attachment decisions, and proceeding to harder and harder ones. When making later decisions, the parser has access to the entire structure built in earlier stages. During the training process, the parser learns its own notion of easy and hard, and learns to defer specific kinds of decisions until more structure is available.

3 Parsing algorithm

Our (projective) parsing algorithm builds the parse tree bottom up, using two kinds of actions: ATTACHLEFT(i) and ATTACHRIGHT(i). These actions are applied to a list of partial structures p_1,...,p_k, called pending, which is initialized with the n words of the sentence w_1,...,w_n. Each action...
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. \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 t^* \leftarrow \arg\max_t w \cdot f(c, t)
7. \quad \quad t_o \leftarrow o(c, T_i)
8. \quad \quad \textbf{if} t^* \neq t_o
9. \quad \quad \quad \textbf{w} \leftarrow \textbf{w} + f(c, t_o) - f(c, t^*)
10. \quad \quad \quad \textbf{c} \leftarrow t_o(c)
11. \quad \textbf{return w}
Online Learning with a Conventional Oracle

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

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

Advanced Transition-Based Parsing Techniques
Oracle Parse

Transitions: SH

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

Advanced Transition-Based Parsing Techniques 18(37)
Oracle Parse

**Transitions:** SH-RA

**Stack**

[ROOT, He]

**Buffer**

[sent, her, a, letter, .]

**Arcs**

ROOT $\rightarrow$ root sent

Advanced Transition-Based Parsing Techniques
Oracle Parse

Transitions:  SH-RA-LA

Stack

[ROOT]

Buffer

[sent, her, a, letter, .]

Arcs

ROOT → sent
He ← sent

ROOT

root

nsubj

ROOT

He

sent

her

a

letter

.

dobj

p

iobj

det

ROOT

pron

verb

pron

det

noun

.

Advanced Transition-Based Parsing Techniques

18(37)
Oracle Parse

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

**Stack**  
[ROOT, sent]

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

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

```
He
sent
her
a
letter
.
```

```
ROOT
pron
verb
pron
det
noun
.
```
Oracle Parse

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

**Stack**

[ROOT, sent, her]

**Buffer**

[a, letter, .]

**Arcs**

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

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

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

He $\xrightarrow{\text{pron}}$ sent

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

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

ROOT $\xrightarrow{\text{pron}}$ verb

he $\xrightarrow{\text{verb}}$ sent

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

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

Advanced Transition-Based Parsing Techniques
Oracle Parse

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

**Stack**

[ROOT, sent, her, a]

**Buffer**

[letter, .]

**Arcs**

- ROOT $\rightarrow$ sent
- He $\leftrightarrow$ sent
- sent $\rightarrow$ her
- sent $\rightarrow$ a
- sent $\rightarrow$ letter
- sent $\rightarrow$ .
Oracle Parse

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

Stack
[ROOT, sent, her]

Buffer
[letter, .]

Arcs
ROOT root sent
He sbj sent
sent iobj her
a det letter
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 | Arcs
--- | --- | ---
[ROOT, sent, letter] | [.] | ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{sbj}$ sent
sent $\xrightarrow{iobj}$ her
a $\xleftarrow{\text{det}}$ letter
sent $\xrightarrow{\text{dobj}}$ letter

Advanced Transition-Based Parsing Techniques
Oracle Parse


Stack  Buffer  Arcs
[ROOT, sent]  [.]  ROOT \(\xrightarrow{\text{root}}\) sent
He \(\xleftarrow{\text{subj}}\) sent
sent \(\xrightarrow{\text{iobj}}\) her
a \(\xleftarrow{\text{det}}\) letter
sent \(\xrightarrow{\text{dobj}}\) letter

Advanced Transition-Based Parsing Techniques
Oracle Parse

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

**Stack**

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

**Buffer**

\[ \text{[]} \]

**Arcs**

\[
\text{ROOT} \xrightarrow{\text{root}} \text{sent} \\
\text{He} \xleftrightarrow{\text{sbj}} \text{sent} \\
\text{sent} \xrightarrow{iobj} \text{her} \\
\text{a} \xleftrightarrow{\text{det}} \text{letter} \\
\text{sent} \xrightarrow{\text{dobj}} \text{letter} \\
\text{sent} \xrightarrow{p} .
\]
Non-Determinism

              SH-RA-LA-SH-RA

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

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

Advanced Transition-Based Parsing Techniques
Non-Determinism

                  SH-RA-LA-SH-RA-RE

Stack

[ROOT, sent]

Buffer

[a, letter, .]

Arcs

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

ROOT

ROOT

She

sent

him

a

letter

. 

pron

verb

pron

det
	noun

. 

root

nsubj

iobj

dobj

det

p
Non-Determinism

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

Stack
[ROOT, sent, a]

Buffer
[letter, .]

Arcs
ROOT → sent
He ← sent
sent → her

ROOT
She
sent
him
a
letter
.
pron
verb
pron
det
noun

Advanced Transition-Based Parsing Techniques
Non-Determinism

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

Stack
[ROOT, sent]

Buffer
[letter, .]

Arcs
ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{\text{subj}}$ sent
sent $\xleftarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ 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  
Buffer  

Arcs  
ROOT $\rightarrow$ sent  
He $\leftrightarrow$ sent  
a $\leftarrow$ det letter  
sent $\rightarrow$ dobj letter
Non-Determinism

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

Stack          Buffer          Arcs
[ROOT, sent]   [.]               ROOT $\xrightarrow{\text{root}}$ sent
               \hspace{1cm}      He $\xleftarrow{\text{sbj}}$ sent
               \hspace{1cm}      sent $\xleftarrow{\text{iobj}}$ her
               \hspace{1cm}      a $\xleftarrow{\text{det}}$ letter
               \hspace{1cm}      sent $\xrightarrow{\text{dobj}}$ letter

Advanced Transition-Based Parsing Techniques
Non-Determinism

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

Stack
[ROOT, sent, .]

Buffer
[ ]

Arcs
ROOT → sent
He ← sent
sent → her
a ← letter
sent → letter
sent → .
Non-Optimality

Transitions:

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

SH-RA-LA-SH

Stack

[ROOT, sent]

Buffer

[her, a, letter, .]

Arcs

ROOT → sent

He ← sent

She sent him a letter.

ROOT

P

Root

nsubj

ROOT

She

pron

sent

verb

ROOT

him

pron

a

det

letter

noun
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 $\rightarrow$ sent
He $\leftarrow$ sent
Non-Optimality


Stack
[ROOT, sent, her, a]

Buffer
[letter, .]

Arcs
ROOT → sent
He ← sent

She sent him a letter.

Advanced Transition-Based Parsing Techniques
Non-Optimality

**Transitions:**

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

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

**Stack**

[ROOT, sent, her]

**Buffer**

[letter, .]

**Arcs**

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

He $\xleftarrow{sbj}$ sent

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

She sent him a letter.

Advanced Transition-Based Parsing Techniques 20(37)
Non-Optimality

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


Stack

[ROOT, sent, her, letter] [.]

Buffer

ROOT She sent him a letter .

Arcs

ROOT $\rightarrow$ sent

He $\leftarrow$ sent a $\leftarrow$ letter
Non-Optimality


Stack  
[ROOT, sent, letter, .]  

Buffer  
[ ]  

Arcs  
ROOT → sent  
He → sent  
a ← letter  

She sent him a letter.
Non-Optimality

Transitions:

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


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

Stack

[ROOT, sent, her]

Buffer

[letter, .]

Arcs

ROOT $\rightarrow$ sent

He $\leftarrow$ sent

a $\leftarrow$ letter

ROOT

pron

She

sent

him

a

letter

.
Non-Optimality

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

Stack
[ROOT, sent]

Buffer
[letter, .]

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

Advanced Transition-Based Parsing Techniques
Non-Optimality

Transitions:

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

Stack

Buffer

Arcs

[ROOT, sent, letter]

[.]

She \textit{sent} him a letter.

He \textit{sent} a letter

her \textit{?} letter

sent \textit{dobj} letter

ROOT \textit{root} sent

He \textit{subj} sent

a \textit{det} letter

her \textit{?} letter

sent \textit{dobj} letter
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-RE

**Stack**

[ROOT, sent]

**Buffer**

[.]

**Arcs**

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

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

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

her $\xleftarrow{?}$ letter

sent $\xrightarrow{\text{dobj}}$ 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 \( \rightarrow \).
Dynamic Oracles

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

- **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**

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

**Notation:**
- \( s \) = node on top of the stack \( S \)
- \( b \) = first node in the buffer \( B \)
Online Learning with a Dynamic Oracle

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

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

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

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]
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]

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

```
ROOT det hearing is scheduled on the det noun today .
```

```
root

prep

nsubj

aux

ROOT

A det noun

is verb

scheduled verb

on prep

the det noun

today adv

p

tmod

pobj

det
```
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]

```
ROOT 0
  ▼ det   hearing is scheduled on the det issue today .
  |      ▼ nsubj
  |       aux
  ▼ prep
  |      ▼ tmod
  |       p
  ▼ nsubj
  ▼ prep
  ▼ pobj
  ▼ det

root
  ▼ det
  |     ▼ nsubj
  |      ▼ prep
  ▼ pobj
  ▼ det
```
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
  det A hearing is scheduled on the det noun today.

root
  det
  aux
  nsubj
  prep tmod
  pobj

p
```

Advanced Transition-Based Parsing Techniques
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]

```
ROOT

0

A

det

1

hearing

nsubj

det

2

is

aux

verb

verb

scheduled

pobj

on

prep

3

the

det

4

issue

noun

5

today

adv

. 
```
Projectivity and Word Order

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

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

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

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

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

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

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

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

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

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

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

**Swap:** \((S|i|j, B, A) \Rightarrow (S|j, i|B, A)\) \(0 < i < j\)
Transition System for Online Reordering

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

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

Terminal: \(([0], [\text{ }], A)\)

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

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

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

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

Transition-based parsing with two interleaved processes:
1. Sort words into projective order \(<_p\)
2. Build tree \(T\) by connecting adjacent subtrees

\(T\) is projective with respect to \(<_p\) but not (necessarily) \(<\)
Example Transition Sequence

[ ]_{s} [ROOT, A, hearing, is, scheduled, on, the, issue, today, .]_{B}

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

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

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

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

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

\begin{align*}
\text{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 . \\
\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

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

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

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

Non-Projective Parsing

Advanced Transition-Based Parsing Techniques
Example Transition Sequence

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[ROOT]_S [ ]_B

ROOT det noun verb verb prep det noun adv.

root

prep

nsubj

aux

tmod

pobj

det

det

ROOT det hearing is scheduled on the det issue today adv.
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</td>
<td>UAS</td>
</tr>
<tr>
<td>Projective</td>
<td>80.8</td>
<td>86.3</td>
</tr>
<tr>
<td>Reordering</td>
<td>83.9</td>
<td>89.1</td>
</tr>
</tbody>
</table>

[Bohnet and Nivre 2012]
Morphology and Syntax

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

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

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

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

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

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

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

Swap: \((S|i|j, B, M, A)\) \Rightarrow \((S|j, i|B, M, A)\) \[0 < i < j\]
Transition System for Morphology and Syntax

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

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

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

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

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

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

\textbf{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>Pipeline</td>
<td>PM</td>
<td>93.0</td>
<td>88.8</td>
<td>89.1</td>
<td>96.1</td>
</tr>
<tr>
<td></td>
<td>LAS</td>
<td>83.1</td>
<td>79.9</td>
<td>91.8</td>
<td>88.4</td>
</tr>
<tr>
<td>Joint</td>
<td>PM</td>
<td>94.4</td>
<td>91.6</td>
<td>91.2</td>
<td>97.4</td>
</tr>
<tr>
<td></td>
<td>LAS</td>
<td>83.5</td>
<td>82.5</td>
<td>92.1</td>
<td>89.1</td>
</tr>
</tbody>
</table>

[Bohnet et al. 2013]
Coming Up Next

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


- Jinho D. Choi and Andrew McCallum. 2013.


References and Further Reading


▶ Yue Zhang and Stephen Clark. 2008.

▶ Yue Zhang and Joakim Nivre. 2011.

▶ Yue Zhang and Joakim Nivre. 2012.