Graph-Based and Transition-Based Dependency Parsing

Joakim Nivre

Uppsala University
Linguistics and Philology

Based on previous tutorials with Ryan McDonald
Overview of the Course

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

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

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

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

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

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

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

$$T^* = \arg\max_{T \in G_x} s(T) = \arg\max_{T \in G_x} \sum_{c=1}^{m} s(G_c)$$
Learning

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

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

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

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

- We will assume that learning is solved
  - Linear scoring plus inference allows us to use Perceptron, MIRA, etc. to find suitable \( w \)
Parameterizing Graph-Based Parsing

First-order (arc-factored) model

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

![Diagram of score computations for a sentence: John saw Mary.]

This search is global: consider all possible trees.
Parameterizing Graph-Based Parsing

**First-order (arc-factored) model**

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

![Diagram of a sentence tree with arrows indicating the arcs and labels for the words and their positions]

- This search is **global**: consider all possible trees
First-Order Projective Parsing

Eisner algorithm

\[ \text{Eisner 1996} \]

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

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

All operations require 3 indexes: \( O(n^3) \)
Feature Scope

\[ f \in \mathbb{R}^n \] is a feature representation of the subgraph \( G_c \)
Feature Scope

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

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

Economic news had little effect on financial markets

adj noun verb adj noun prep adj noun
amod nsubj dobj amod prep pmod amod
Feature Scope

- $f \in \mathbb{R}^n$ is a feature representation of the subgraph $G_c$
- For first-order models, $G_c$ is an arc
  - i.e., $G_c = (i, j)$ for a head $i$ and modifier $j$
- This inherently limits features to a local scope
- For arc (had, effect) below, can have features over properties of arc and context within sentence


**Feature Scope**

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

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

[McDonald and Nivre 2007]

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

John Smith was tall noun noun verb adj

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

[McDonald and Nivre 2007]

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

```
John noun Smith noun was verb tall adj
```

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

[McDonald and Nivre 2007]

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

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

- Two main dimensions of higher-order features
  - Vertical: e.g., “remain” is the grandparent of “emeritus”
  - Horizontal: e.g., “remain” is first child of “will”
2nd-Order Horizontal Projective Parsing

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

\[
\begin{align*}
    s(T) &= \sum_{(i,j):(i,j') \in A} s(i, j, j') \\
    &= \ldots + s(i_0, i_1, i_2) + s(i_0, i_2, i_3) + \ldots + s(i_0, i_{j-1}, i_j) + \\
    &\quad s(i_0, i_{j+1}, i_{j+2}) + \ldots + s(i_0, i_{m-1}, i_m) + \ldots
\end{align*}
\]
2nd-Order Horizontal Projective Parsing

- Add a sibling chart item to get to $O(n^3)$
Higher-Order Projective Parsing

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

* From Koo et al. 2010 presentation
Exact Higher-Order Projective Parsing

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>En-UAS</th>
<th>Zh-UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st order exact</td>
<td>91.8</td>
<td>84.4</td>
</tr>
<tr>
<td>2nd order exact</td>
<td>92.4</td>
<td>86.6</td>
</tr>
<tr>
<td>3rd order exact</td>
<td>93.0</td>
<td>86.8</td>
</tr>
<tr>
<td>4th order exact</td>
<td>93.4</td>
<td>87.4</td>
</tr>
<tr>
<td>struct. pred. casc.</td>
<td>93.1</td>
<td>–</td>
</tr>
<tr>
<td>cube-pruning</td>
<td>93.5</td>
<td>87.9</td>
</tr>
</tbody>
</table>

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

Cube-pruning is 5x faster and structured prediction cascades 10x faster than third-order.
Non-Projective Parsing

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

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

- Exact: integer linear programming (ILP) 
- Inference intractable, but efficient optimizers exist
- Easy to extend by adding labels, grammar constraints, etc.
- Related to constraint dependency grammar
- Approximate inference:
  - \[T^* = \arg\max_{T \in G} (T)\]
- Post-processing: [McDonald and Pereira 2006], [Hall and Novák 2005], [Hall 2007]
- Sampling: [Nakagawa 2007]
- Belief Propagation: [Smith and Eisner 2008]
- Dual Decomposition: [Koo et al. 2010]
- Approximate search space:
  - \[T^* = \arg\max_{T \in G} (T)\]
Higher-Order Non-Projective Parsing

- Exact: integer linear programming (ILP)
  - Inference intractable, but efficient optimizers exist
  - Easy to extend by adding labels, grammar constraints, etc.
  - Related to constraint dependency grammar
Higher-Order Non-Projective Parsing

- Exact: integer linear programming (ILP)
  - Inference intractable, but efficient optimizers exist
  - Easy to extend by adding labels, grammar constraints, etc.
  - Related to constraint dependency grammar

- Approximate inference: $T^* = \arg\max_{T \in G_x} s(T)$
  - Post-processing [McDonald and Pereira 2006],
    [Hall and Novák 2005], [Hall 2007]
  - Sampling [Nakagawa 2007]
  - Belief Propagation [Smith and Eisner 2008]
  - Dual Decomposition [Koo et al. 2010]
Higher-Order Non-Projective Parsing

- **Exact:** integer linear programming (ILP)
    - Inference intractable, but efficient optimizers exist
    - Easy to extend by adding labels, grammar constraints, etc.
    - Related to constraint dependency grammar

- **Approximate inference:** \( T^* = \arg\max_{T \in G_x} s(T) \)
  - Post-processing [McDonald and Pereira 2006],
    - [Hall and Novák 2005], [Hall 2007]
  - Sampling [Nakagawa 2007]
  - Belief Propagation [Smith and Eisner 2008]
  - Dual Decomposition [Koo et al. 2010]

- **Approximate search space:** \( T^* = \arg\max_{T \in G_x} s(T) \)
  - Mildly non-projective structures
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: \((S, B, A)\) \[S = \text{Stack}, \ B = \text{Buffer}, \ A = \text{Arcs}\]

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

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

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

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

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

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

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

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

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

ROOT \quad Economic \quad news \quad had \quad little \quad effect \quad on \quad financial \quad markets \quad .

adj \quad noun \quad verb \quad adj \quad noun \quad prep \quad adj \quad noun \quad .
Example Transition Sequence

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

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

ROOT \quad Economic \quad news \quad had \quad little \quad effect \quad on \quad financial \quad markets \quad .

adj \quad noun \quad verb \quad adj \quad noun \quad prep \quad adj \quad noun \quad .
Example Transition Sequence

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

\[
\begin{array}{cccccccccccc}
{\text{root}} & {\text{amod}} & {\text{nsubj}} & {\text{ROOT}} & {\text{Economic}} & {\text{news}} & {\text{had}} & {\text{little}} & {\text{effect}} & {\text{on}} & {\text{financial}} & {\text{markets}} & {\text{.}} \\
\end{array}
\]
Example Transition Sequence

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

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

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

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

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

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

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

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

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

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

\[[\text{ROOT, had}]_S \quad [.\quad B]\]
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: \([(\;], [0, 1, \ldots, n], \{ \; \})\)

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

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

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

Left-Arc\(k\): \((S\mid i\mid j, B, A) \Rightarrow (S\mid j, B, A \cup \{(j, i, k)\})\) \(i \neq 0\)
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. $c \leftarrow ([S, [0, \ldots, n]]_B, \{ \})$
2. while $B_c \neq []$
3. $t \leftarrow o(c)$
4. $c \leftarrow t(c)$
5. return $G = ([0, \ldots, n], A_c)$

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

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

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

**Configuration**

```
[ROOT, had, effect]_S  [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 $A$

Configuration

Features

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

- 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

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

- Feature representation unconstrained by parsing algorithm
Local Learning

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

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

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

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

- **CoNLL 2006 shared task** [Buchholz and Marsi 2006]:
  - **MaltParser** [Nivre et al. 2006] – deterministic, local learning
  - **MSTParser** [McDonald et al. 2006] – exact, global learning
  - Same average parsing accuracy over 13 languages

- **Comparative error analysis** [McDonald and Nivre 2007]:
  - **MaltParser** more accurate on short dependencies and disambiguation of core grammatical functions
  - **MSTParser** more accurate on long dependencies and dependencies near the root of the tree

- **Hypothesized explanation for MaltParser results:**
  - Rich features counteracted by error propagation
Precision by Dependency Length

Graph showing the precision of MST and Malt parsing methods by dependency length.
Beam Search

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

  \[ \text{Parse}(w_1, \ldots, w_n) \]
  \[
  1. \quad \text{Beam} \leftarrow \{([ ]_S, [0, 1, \ldots, n]_B, \{ \}) \}
  
  2. \quad \textbf{while} \ \exists c \in \text{Beam} \ [B_c \neq [ ]] \]
  
  3. \quad \textbf{foreach} \ c \in \text{Beam} 
  
  4. \quad \textbf{foreach} \ t 
  
  5. \quad \text{Add}(t(c), \text{NewBeam}) 
  
  6. \quad \text{Beam} \leftarrow \text{Top}(k, \text{NewBeam}) 
  
  7. \quad \textbf{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]
Non-Projective Parsing

- So far only projective parsing models
- Non-projective parsing harder even with greedy inference
  - Non-projective: \( n(n - 1) \) arcs to consider – \( O(n^2) \)
  - Projective: at most \( 2(n - 1) \) arcs to consider – \( O(n) \)

- Approaches:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Extended arc transitions [Attardi 2006]
  - List-based algorithms [Covington 2001, Nivre 2007]
  - Online reordering [Nivre 2009, Nivre et al. 2009]:
Non-Projective Parsing

- So far only projective parsing models
- Non-projective parsing harder even with greedy inference
  - Non-projective: \( n(n - 1) \) arcs to consider – \( O(n^2) \)
  - Projective: at most \( 2(n - 1) \) arcs to consider – \( O(n) \)
- Approaches:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Extended arc transitions [Attardi 2006]
  - List-based algorithms [Covington 2001, Nivre 2007]
  - Online reordering [Nivre 2009, Nivre et al. 2009]:
Non-Projective 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 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
 ROOT det
 A hearing
  noun
 is verb
  verb
 scheduled
  verb
 on prep
 the det
 issue noun
 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]

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

```
root

A hearing is scheduled on the issue today .

ROOT 0
```
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]
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.
  0   1 hearing is scheduled on the det issue today .
  2
```
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]

Graph-based and Transition-based Dependency Parsing 36(44)
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]

![Dependency Tree]

ROOT A hearing is scheduled on the issue today.

ROOT det noun verb verb prep det noun 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]

![Dependency Tree Example]

- Root
- A
- hearing
- is
- scheduled
- on
- the
- issue
- today
- .
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 0
 A det 1
 hearing noun 2
 is verb 6
 scheduled verb
 on prep 3
 det 4
 the det noun 5
 issue noun 6
 today adv

root
↓
det
↓
nsubj
↓
prep
↓
aux
↓
tmod
↓
pobj
↓
det
↓
```

Graph-Based and Transition-Based Dependency Parsing 36(44)
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].

![Dependency Tree Example]

ROOT  | A  | hearing | is | scheduled | on | the | issue | today |
0     | det| noun    | verb| verb      | prep| det| noun  | 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]
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: \((S|i|j, B, A) \Rightarrow (S|j, i|B, A)\quad 0 < i < j\)
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 \(\prec_p\)
  2. Build tree \(T\) by connecting adjacent subtrees

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

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

\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

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

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

\[
\begin{array}{cccccccccc}
\text{ROOT} & \text{A} & \text{hearing} & \text{is} & \text{scheduled} & \text{on} & \text{the} & \text{issue} & \text{today} & .
\end{array}
\]

\[
\begin{array}{cccccccccc}
\text{ROOT} & \text{det} & \text{noun} & \text{verb} & \text{verb} & \text{prep} & \text{det} & \text{noun} & \text{adv} & .
\end{array}
\]
Example Transition Sequence

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

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

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

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

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

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

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

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

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

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

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

Graph-Based and Transition-Based Dependency Parsing
Example Transition Sequence

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

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

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

\[[\text{ROOT, scheduled}]_S \quad \text{}_B\]
Non-Projective Parsing

Example Transition Sequence

[ROOT]_S [ ]_B

Graph-Based and Transition-Based Dependency Parsing
Non-Projective 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</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: \((S, B, M, A)\) \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-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:
  - $PM = \text{morphology (postag + features)}$
  - $LAS = \text{labeled attachment score}$

<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>96.1</td>
</tr>
<tr>
<td>Joint</td>
<td>94.4</td>
<td>83.5</td>
<td>91.6</td>
<td>82.5</td>
<td>97.4</td>
</tr>
</tbody>
</table>

[Bohnet et al. 2013]
Where do we stand?

Graph-based Parsers
- Global Inference
- Global Learning
- Local Feature Scope

Transition-based Parsers
- Local Inference
- Local Learning
- Global Feature Scope

2008
- higher-order chart parsing
- pruning
- ILP
- dual decomp
- mildly non-projective
- etc.

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

2014
- beam search
- perceptron
- dynamic oracles
- dynamic programming
- more features
- etc.

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

Evaluated on overlapping 9 languages in studies
Coming Up Next

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


References and Further Reading


Methods in Natural Language Processing and the Conference on Computational Natural Language Learning (EMNLP-CoNLL).


References and Further Reading


References and Further Reading


