Non-Projective Dependency Parsing

Based partly on the Kübler et al (2009) book
NASSLLI short course on Dependency Parsing
Summer 2010

Markus Dickinson & Sandra Kübler
Projectivity (review)

An arc \((w_i, r, w_j) \in A\) is *projective* iff \(w_i \rightarrow^* w_k\) for all:

- \(i < k < j\) when \(i < j\)
- \(j < k < i\) when \(j < i\)
Covington’s Algorithm (review)

How is it that Covington’s algorithm allows for non-projectivity?

- Deterministic incremental parsing in $O(n^2)$ time by trying to link each new word to each preceding one [Covington(2001)]:

\[
\text{PARSE}(x = (w_1, \ldots, w_n))
\]

1. for $i = 1$ up to $n$
2. for $j = i - 1$ down to 1
3. \text{LINK}(w_i, w_j)

\[
\text{LINK}(w_i, w_j) = \begin{cases} 
E \leftarrow E \cup (i, j) & \text{if } w_j \text{ is a dependent of } w_i \\
E \leftarrow E \cup (j, i) & \text{if } w_i \text{ is a dependent of } w_j \\
E \leftarrow E & \text{otherwise}
\end{cases}
\]
Nivre’s Algorithm (review)

Why is Nivre’s algorithm only projective?

- Four parsing actions:

  - **Shift**
    \[
    \begin{array}{c|c|c}
    \text{Shift} & [\ldots]s & [w_i, \ldots]Q \\
    \hline
    [\ldots, w_i]s & [\ldots]Q
    \end{array}
    \]

  - **Reduce**
    \[
    \begin{array}{c|c|c}
    \text{Reduce} & [\ldots, w_i]s & [\ldots]Q \\
    \hline
    \exists w_k : w_k \rightarrow w_i
    \end{array}
    \]

  - **Left-Arc**
    \[
    \begin{array}{c|c|c}
    \text{Left-Arc}_r & [\ldots, w_i]s & [w_j, \ldots]Q \\
    \hline
    \neg \exists w_k : w_k \rightarrow w_i & \quad & \quad & w_j \leftarrow w_j
    \end{array}
    \]

  - **Right-Arc**
    \[
    \begin{array}{c|c|c}
    \text{Right-Arc}_r & [\ldots, w_i]s & [w_j, \ldots]Q \\
    \hline
    \neg \exists w_k : w_k \rightarrow w_j & \quad & \quad & w_i \rightarrow w_j
    \end{array}
    \]
Limitations of some transition-based parsing

Why won’t some transition-based parsers (e.g., Nivre’s) handle non-projective structures?

(root Z nich je jen jedna na kvalitu .)

(out-of) (them) (is) (only) (one) (to) (quality)
Limitations of some transition-based parsing

Why won’t some transition-based parsers (e.g., Nivre’s) handle non-projective structures?

(root (Z nich je jen Z jedna) na kvalitu).

(out-of) (them) (is) (only) (one) (to) (quality)
Allowable transitions

At this stage:

- Z is on top of the stack, and je is on top of the buffer
Allowable transitions

At this stage:

- $Z$ is on top of the stack, and $je$ is on top of the buffer
- We want an arc between root and $je$, leaving $Z$ unattached
Non-Projective Dependency Parsing

- Many parsing algorithms are restricted to projective dependency graphs.
- Is this a problem?
- Statistics from CoNLL-X Shared Task [Buchholz and Marsi(2006)]
  - NPD = Non-projective dependencies
  - NPS = Non-projective sentences

<table>
<thead>
<tr>
<th>Language</th>
<th>%NPD</th>
<th>%NPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>5.4</td>
<td>36.4</td>
</tr>
<tr>
<td>German</td>
<td>2.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>Slovene</td>
<td>1.9</td>
<td>22.2</td>
</tr>
<tr>
<td>Portuguese</td>
<td>1.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Danish</td>
<td>1.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Two Main Approaches

- **Algorithms for non-projective dependency parsing:**
  - Constraint satisfaction methods
  - McDonald’s spanning tree algorithm
    - McDonald et al. (2005)
  - Covington’s algorithm
    - Nivre (2006)

- **Post-processing of projective dependency graphs:**
  - Pseudo-projective parsing
    - Nivre and Nilsson (2005)
  - Corrective modeling
    - Hall and Novák (2005)
  - Approximate non-projective parsing
    - McDonald and Pereira (2006)
## Non-Projective Parsing Algorithms

- **Complexity considerations:**
  - Projective (Proj)
  - Non-projective (NonP)

<table>
<thead>
<tr>
<th>Problem/Algorithm</th>
<th>Proj</th>
<th>NonP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete grammar parsing</td>
<td>$P$</td>
<td>NP hard</td>
</tr>
<tr>
<td>[Gaifman(1965), Neuhaus and Bröker(1997)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deterministic parsing</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[Nivre(2003), Covington(2001)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First order spanning tree</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[McDonald et al.(2005)McDonald, Pereira, Riberov and Hajič]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N$th order spanning tree ($N &gt; 1$)</td>
<td>$P$</td>
<td>NP hard</td>
</tr>
<tr>
<td>[McDonald and Pereira(2006)]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Changing the transition system

One way to allow for a limited amount of non-projectivity is to change the definitions of the transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-Left(_r)</td>
<td>((\sigma</td>
</tr>
<tr>
<td>NP-Right(_r)</td>
<td>((\sigma</td>
</tr>
</tbody>
</table>

These definitions apply to the second word on the stack

- Top word on the stack is treated as a context node
- Queue is now allowed to have items re-inserted on it
Changing the transition system

One way to allow for a limited amount of non-projectivity is to change the definitions of the transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-Left$_r$</td>
<td>$(\sigma</td>
</tr>
<tr>
<td>NP-Right$_r$</td>
<td>$(\sigma</td>
</tr>
</tbody>
</table>

These definitions apply to the second word on the stack

- Top word on the stack is treated as a context node
- Queue is now allowed to have items re-inserted on it

Will this work for *A hearing is scheduled on the issue today*? (see handout based on [Nivre(2009)])
Changing the transition system (2)

Another approach is to define configurations with two stacks instead of one

Idea:

- A No-arc transition allows a node to be passed from the stack to an auxiliary stack without having been given a head
- Later, this node can return to the stack and be in a dependency relation
  - i.e., save the node for later analysis

See [Nivre(2008)] (sec. 5.1) for more details
Changing the transition system (3)

Another way to alter the transition system is to add a transition Swap

<table>
<thead>
<tr>
<th>Transition</th>
<th>Stack</th>
<th>Buffer</th>
<th>Arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap</td>
<td>[root\textsubscript{0}, ..., scheduled\textsubscript{4}, on\textsubscript{5}]</td>
<td>[the\textsubscript{6}, ..., \textsubscript{9}]</td>
<td></td>
</tr>
<tr>
<td>Swap</td>
<td>[root\textsubscript{0}, ..., is\textsubscript{3}, on\textsubscript{5}]</td>
<td>[scheduled\textsubscript{4}, ..., \textsubscript{9}]</td>
<td></td>
</tr>
<tr>
<td>Swap</td>
<td>[root\textsubscript{0}, ..., hearing\textsubscript{3}, on\textsubscript{5}]</td>
<td>[is\textsubscript{3}, ..., \textsubscript{9}]</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Consider: A hearing is scheduled on the issue today.

See [Nivre(2009)] for more details.
Post-Processing

Two-step approach:

1. Derive the best projective approximation of the correct (possibly) non-projective dependency graph.
2. Improve the approximation by replacing projective arcs by (possibly) non-projective arcs.

Rationale:

- Most “naturally occurring” dependency graphs are primarily projective, with only a few non-projective arcs.

Approaches:

- Pseudo-projective parsing [Nivre and Nilsson(2005)]
- Corrective modeling [Hall and Novák(2005)]
- Approximate non-projective parsing [McDonald and Pereira(2006)]
Pseudo-Projective Parsing

1. Projectivize dependency trees in the training set while encoding information about necessary transformations in augmented arc labels.
2. Train a projective parser on the transformed training set.
3. Parse new sentences using the projective parser.
4. Deprojectivize the output of the projective parser, using heuristic transformations guided by augmented arc labels.
Pseudo-Projective Parsing

- Projectivize training data:
  - Projective head nearest permissible ancestor of real head
  - Arc label extended with dependency type of real head

```
root
Z nich je jen jedna na kvalitu.
```

```
Pred
Atr

AuxK

AuxP

Atr

Sb

AuxP

Adv

na kvalitu

je jen jedna

(out-of) (is) (only) (one)

(je) (jena)

(only) (one)

(je)

(z)

(out-of) (them)

(root)
```
Pseudo-Projective Parsing

- Projectivize training data:
  - Projective head nearest permissible ancestor of real head
  - Arc label extended with dependency type of real head

```
 root
 Z nich je jen jedna na kvalitu.
```

Pred

AuxK

AuxP

Atr

Sb

AuxZ

Adv

(out-of) (them) (is) (only) (one) (to) (quality)
Pseudo-Projective Parsing

- Deprojectivize parser output:
  - Top-down, breadth-first search for real head
  - Search constrained by extended arc label

Root: Z nich je jedna na kvalitu.

(out-of) (them) (is) (only) (one) (to) (quality)
Pseudo-Projective Parsing

- Deprojectivize parser output:
  - Top-down, breadth-first search for real head
  - Search constrained by extended arc label

```
root
Z nich je jen jedna
(out-of) (them) (is) (only) (one) (to) (quality)
```

AuxK

```
AuxP

Pred

AuxP↑Sb

Atr

Sb

AuxZ

Adv

na kvalitu

(jen) (one) (to) (quality)
```
Pseudo-Projective Parsing

Advantages:

- Allows sentences with arbitrary non-projective structures to be parsed in linear time (if projectivization & deprojectivization are linear)
- The system is sound, as long as the base parser is

Disadvantages:

- Increase in the number of distinct dependency labels
Corrective Modeling

- Conditional probability model
  \[ P(h_i' | w_i, N(h_i)) \]
  for correcting the head \( h_i \) of word \( w_i \) to \( h_i' \), restricted to the local neighborhood \( N(h_i) \) of \( h_i \)

- Model trained on parser output and gold standard parses (MaxEnt estimation)

- Post-processing:
  - For every word \( w_i \), replace \( h_i \) by \( \arg\max_{h_i'} P(h_i' | w_i, N(h_i)) \).
Corrective Modeling

[Attardi and Ciaramita(2007)]:

- Revise a base parse tree, by applying tree revision rules
  - Revision rules are combinations of atomic revisions, e.g., $u = \text{move the dependency up one parent}$
  - e.g., $uu$ specifies moving up twice; $-1$ specifies moving to the left one token; etc.
- Can use global parse features to determine revisions

[Attardi and Dell’Orletta(2009)]:

- Use a second shift-reduce parser, run in the reverse direction and with additional features from the parse tree, to revise trees
- Use a voting method to select the best set of dependents from among 3 parsers
Motivating Graph-Based Parsing

In some sense, it would be best to start with a methodology that accounts for non-projectivity from the start.

**Graph-based parsing** explicitly parameterizes over the substructures of a dependency tree:

- Consider all possible arcs of a dependency forest at once.
- Use graph-theoretic algorithms to reduce this to a tree.
  - Such algorithms do not distinguish projective or non-projective arcs.
The score of a dependency tree $y$ for input sentence $x$ is

$$\sum_{(i,k,j) \in y} s(i, k, j)$$

where $k$ and $j$ are adjacent, same-side children of $i$ in $y$.

The highest scoring projective dependency tree can be computed exactly in $O(n^3)$ time using Eisner’s algorithm.

The highest scoring non-projective dependency tree can be approximated with a greedy post-processing procedure:

- While improving the global score of the dependency tree, replace an arc $h_i \rightarrow w_i$ by $h'_i \rightarrow w_i$, greedily selecting the substitution that gives the greatest improvement.
State of the Art – Czech

Evaluation:

- Prague Dependency Treebank (PDT)
- Unlabeled accuracy per word (W) and per sentence (S)
  - Non-projective spanning tree parsing
    [McDonald et al. (2005) McDonald, Pereira, Ribarov and Hajič]
  - Corrective modeling on top of the Charniak parser
    [Hall and Novák (2005)]
  - Approximate non-projective parsing on top of a second-order projective spanning tree parser [McDonald and Pereira (2006)]
  - Pseudo-projective parsing on top of a deterministic classifier-based parser

<table>
<thead>
<tr>
<th>Parser</th>
<th>W</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald and Pereira</td>
<td>85.2</td>
<td>35.9</td>
</tr>
<tr>
<td>Hall and Novák</td>
<td>85.1</td>
<td>—</td>
</tr>
<tr>
<td>Nilsson et al.</td>
<td>84.6</td>
<td>37.7</td>
</tr>
<tr>
<td>McDonald et al.</td>
<td>84.4</td>
<td>32.3</td>
</tr>
<tr>
<td>Charniak</td>
<td>84.4</td>
<td>—</td>
</tr>
</tbody>
</table>


Gaifman, Haim (1965). Dependency systems and phrase-structure systems.


In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL) and the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. pp. 337–343.

In *Proceedings of COLING-ACL*.


