Dependency Parsing

CMSC 723 / LING 723 / INST 725

Marine Carpuat

Fig credits: Joakim Nivre, Dan Jurafsky & James Martin
Dependency Parsing

• Formalizing dependency trees

• Transition-based dependency parsing
  • Shift-reduce parsing
  • Transition system
  • Oracle
  • Learning/predicting parsing actions
Dependency Grammars

• Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies
## Dependency Relations

<table>
<thead>
<tr>
<th>Argument Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>nominal subject</td>
</tr>
<tr>
<td>csubj</td>
<td>clausal subject</td>
</tr>
<tr>
<td>dobj</td>
<td>direct object</td>
</tr>
<tr>
<td>iobj</td>
<td>indirect object</td>
</tr>
<tr>
<td>pobj</td>
<td>object of preposition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifier Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmod</td>
<td>temporal modifier</td>
</tr>
<tr>
<td>appos</td>
<td>appositional modifier</td>
</tr>
<tr>
<td>det</td>
<td>determiner</td>
</tr>
<tr>
<td>prep</td>
<td>prepositional modifier</td>
</tr>
<tr>
<td>Relation</td>
<td>Examples with head and dependent</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>NSUBJ</td>
<td>United <em>canceled</em> the flight.</td>
</tr>
<tr>
<td>DOBJ</td>
<td>United <em>diverted</em> the <em>flight</em> to Reno.</td>
</tr>
<tr>
<td></td>
<td>We <em>booked</em> her the first <em>flight</em> to Miami.</td>
</tr>
<tr>
<td>IOBJ</td>
<td>We <em>booked</em> <em>her</em> the flight to Miami.</td>
</tr>
<tr>
<td>NMOD</td>
<td>We took the <em>morning</em> <em>flight</em>.</td>
</tr>
<tr>
<td>AMOD</td>
<td>Book the <em>cheapest</em> <em>flight</em>.</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Before the storm JetBlue canceled 1000 <em>flights</em>.</td>
</tr>
<tr>
<td>APPOS</td>
<td>United, a <em>unit</em> of UAL, matched the fares.</td>
</tr>
<tr>
<td>DET</td>
<td>The <em>flight</em> was canceled.</td>
</tr>
<tr>
<td></td>
<td><em>Which</em> flight was delayed?</td>
</tr>
<tr>
<td>CONJ</td>
<td>We <em>flew</em> to Denver and <em>drove</em> to Steamboat.</td>
</tr>
<tr>
<td>CC</td>
<td>We flew to Denver <em>and</em> <em>drove</em> to Steamboat.</td>
</tr>
<tr>
<td>CASE</td>
<td>Book the flight <em>through</em> Houston.</td>
</tr>
</tbody>
</table>

**Figure 14.3** Examples of core Universal Dependency relations.
Example Dependency Parse

They hid the letter on the shelf

Compare with constituent parse…

What's the relation?
Dependency formalisms

- Most general form: a graph $G = (V, A)$
  - $V$ vertices: usually one per word in sentence
  - $A$ arcs (set of ordered pairs of vertices): head-dependent relations between elements in $V$
- Restricting to **trees** provide computational advantages
  - Single designated ROOT node that has no incoming arcs
  - Except for ROOT, each vertex has exactly one incoming arc
  - Unique path from ROOT to each vertex in $V$

- Each word has a single head
- Dependency structure is connected
- There is a single root node from which there is a unique path to each word
Economic news had little effect on financial markets.
What did economic news have little effect on?
Projectivity

• **Arc** from head to dependent is *projective*
  • If there is a path from head to every word between head and dependent

• **Dependency tree** is *projective*
  • If all arcs are projective
  • Or equivalently, if it can be drawn with no crossing edges

• Projective trees make computation easier
• But most theoretical frameworks do not assume projectivity
  • Need to capture long-distance dependencies, free word order
Data-driven dependency parsing

**Goal:** learn a good predictor of dependency graphs

  Input: sentence
  Output: dependency graph/tree $G = (V,A)$

Can be framed as a structured prediction task

- very large output space
- with interdependent labels

2 dominant approaches: transition-based parsing and graph-based parsing
Transition-based dependency parsing

- Builds on shift-reduce parsing [Aho & Ullman, 1927]

- **Configuration**
  - **Stack**
  - **Input buffer** of words
  - Set of dependency relations

- **Goal of parsing**
  - find a final configuration where
  - all words accounted for
  - Relations form dependency tree

---

**Figure 14.5** Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.
Transition operators

- Transitions: produce a new configuration given current configuration

- Parsing is the task of
  - Finding a sequence of transitions
  - That leads from start state to desired goal state

- Start state
  - Stack initialized with ROOT node
  - Input buffer initialized with words in sentence
  - Dependency relation set = empty

- End state
  - Stack and word lists are empty
  - Set of dependency relations = final parse
Arc Standard Transition System

- Defines 3 transition operators [Covington, 2001; Nivre 2003]

- **LEFT-ARC:**
  - create head-dependent rel. between word at top of stack and 2nd word (under top)
  - remove 2nd word from stack

- **RIGHT-ARC:**
  - Create head-dependent rel. between word on 2nd word on stack and word on top
  - Remove word at top of stack

- **SHIFT**
  - Remove word at head of input buffer
  - Push it on the stack
Arc standard transition systems

• Preconditions
  • ROOT cannot have incoming arcs
  • LEFT-ARC cannot be applied when ROOT is the 2\textsuperscript{nd} element in stack
  • LEFT-ARC and RIGHT-ARC require 2 elements in stack to be applied
Transition-based Dependency Parser

- Assume an oracle
- Parsing complexity
  - Linear in sentence length!
- Greedy algorithm
  - Unlike Viterbi for POS tagging

**Figure 14.6** A generic transition-based dependency parser

```plaintext
function DEPENDENCYPARSE(words) returns dependency tree

state ← {[root], [words], []} ; initial configuration
while state not final
    t ← ORACLE(state) ; choose a transition operator to apply
    state ← APPLY(t, state) ; apply it, creating a new state

return state
```
### Transition-Based Parsing Illustrated

**Figure 14.7** Trace of a transition-based parse.
Where to we get an oracle?

• Multiclass classification problem
  • Input: current parsing state (e.g., current and previous configurations)
  • Output: one transition among all possible transitions
  • Q: size of output space?

• Supervised classifiers can be used
  • E.g., perceptron

• Open questions
  • What are good features for this task?
  • Where do we get training examples?
Generating Training Examples

- What we have in a treebank
- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

![Diagram](image_url)

**Figure 14.8** Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.
Generating training examples

• Approach: simulate parsing to generate reference tree

• Given
  • A current config with stack S, dependency relations Rc
  • A reference parse (V,Rp)

• Do

\[
\text{LEFTARC}(r): \text{ if } (S_1 \ r \ S_2) \in R_p \\
\text{RIGHTARC}(r): \text{ if }(S_2 \ r \ S_1) \in R_p \text{ and } \forall r', w. s. t. (S_1 \ r' \ w) \in R_p \text{ then } (S_1 \ r' \ w) \in R_c \\
\text{SHIFT}: \text{ otherwise}
\]
Let’s try it out

**LEFTARC(r):** if \((S_1 \ r \ S_2) \in R_p\)

**RIGHTARC(r):** if \((S_2 \ r \ S_1) \in R_p\) and \(\forall r', w \ s.t. (S_1 \ r' \ w) \in R_p\) then \((S_1 \ r' \ w) \in R_c\)

**SHIFT:** otherwise

```
root
  └── dobj
      └── det
          └── Book
  └── nmod
      └── case
          └── the flight through Houston
```
Features

- Configuration consist of stack, buffer, current set of relations

- Typical features
  - Features focus on top level of stack
  - Use word forms, POS, and their location in stack and buffer
Features example

- Given configuration

<table>
<thead>
<tr>
<th>Stack</th>
<th>Word buffer</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[root, canceled, flights]</td>
<td>[to Houston]</td>
<td>(canceled -&gt; United)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights -&gt; morning)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights -&gt; the)</td>
</tr>
</tbody>
</table>

- Example of useful features

\[
\begin{align*}
\langle s_1.w = \text{flights}, op = \text{shift} \rangle \\
\langle s_2.w = \text{canceled}, op = \text{shift} \rangle \\
\langle s_1.t = \text{NNS}, op = \text{shift} \rangle \\
\langle s_2.t = \text{VBD}, op = \text{shift} \rangle \\
\langle b_1.w = \text{to}, op = \text{shift} \rangle \\
\langle b_1.t = \text{TO}, op = \text{shift} \rangle \\
\langle s_1.wt = \text{flightsNNS}, op = \text{shift} \rangle \\
\langle s_1.t.s_2.t = \text{NNSVBD}, op = \text{shift} \rangle
\end{align*}
\]
Dependency Parsing

• Formalizing dependency trees

• Transition-based dependency parsing
  • Shift-reduce parsing
  • Transition system
  • Oracle
  • Learning/predicting parsing actions