Dependency Parsing

CMSC 470
Marine Carpuat
Dependency Grammars

- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies
Example Dependency Parse

Dependencies (usually) form a tree:
- Connected
- Acyclic
- Single-head

```
root  India  won  the  world  cup  by  beating  Lanka
       nsubj  det  nn  pcomp  dobj
          prep  dobj
     nsubj
```

Compare with constituent parse…

What's the relation?
Universal Dependencies project

• Set of dependency relations that are
  • Linguistically motivated
  • Computationally useful
  • Cross-linguistically applicable
  [Nivre et al. 2016]

• 100+ dependency treebanks for more than 60 languages

universaldependencies.org
Universal Dependencies

Design principles

• UD needs to be satisfactory on linguistic analysis grounds for individual languages.

• UD needs to be good for linguistic typology, i.e., providing a suitable basis for bringing out cross-linguistic parallelism across languages and language families.

• UD must be suitable for rapid, consistent annotation by a human annotator.

• UD must be suitable for computer parsing with high accuracy.

• UD must be easily comprehended and used by a non-linguist, whether a language learner or an engineer with prosaic needs for language processing. We refer to this as seeking a *habitable* design, and it leads us to favor traditional grammar notions and terminology.

• UD must support well downstream language understanding tasks (relation extraction, reading comprehension, machine translation, ...).
Syntax in NLP

• Syntactic analysis can be useful in many NLP applications
  • Grammar checkers
  • Dialogue systems
  • Question answering
  • Information extraction
  • Machine translation
  • …

• Sequence models can go a long way but syntactic analysis is particularly useful
  • In low resource settings
  • In tasks where precise output structure matters
Syntactic analysis can help NLP tasks by

Providing scaffolding for semantic analysis (and representing or resolving ambiguity)

Helping generalization (e.g., by capturing long-distance dependencies)

After much economic progress over the years, the country has...

The country, which has made much economic progress over the years, still has...
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, 1972]

- **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.
Defining Transitions

• **Transitions**
  • Are functions that 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]

**SHIFT**
- Remove word at head of input buffer
- Push it on the stack

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

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

---

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.
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

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

```
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
```

Properties of this algorithm:
- Linear in sentence length
- A greedy algorithm
- Output quality depends on oracle
Exercise: find a sequence of transitions to generate this parse

**SHIFT**
- Remove word at head of input buffer
- Push it on the stack

**LEFT-ARC**
- create head-dependent relation between word at top of stack and 2\textsuperscript{nd} word (under top)
- remove 2\textsuperscript{nd} word from stack

**RIGHT-ARC**
- Create head-dependent relation between word on 2\textsuperscript{nd} word on stack and word on top
- Remove word at top of stack
### Transition-Based Parsing Illustrated

- **Diagram**: A tree diagram illustrating a parse of the sentence *Book me the morning flight*.

- **Table**: A step-by-step breakdown of the parse process with actions and relation added at each step.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book → flight)</td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 14.7** Trace of a transition-based parse.
Where do 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 of a parse tree with labels: root, dobj, det, nmod, case. The sentence is "Book the flight through Houston." ]

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Predicted Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, the, flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[the, flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>2</td>
<td>[root, book, the]</td>
<td>[flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>3</td>
<td>[root, book, the, flight]</td>
<td>[through, houston]</td>
<td>LEFTARC</td>
</tr>
<tr>
<td>4</td>
<td>[root, book, flight]</td>
<td>[through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>5</td>
<td>[root, book, flight, through]</td>
<td>[houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>6</td>
<td>[root, book, flight, through, houston]</td>
<td>[]</td>
<td>LEFTARC</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, flight, houston ]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>9</td>
<td>[root, book]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>Done</td>
</tr>
</tbody>
</table>

*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 $R_c$
  • A reference parse $(V,R_p)$

• Do

  $\text{LEFTARC}(r)$: if $(S_1 r S_2) \in R_p$

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

  $\text{SHIFT}$: otherwise

Additional condition on RightArc makes sure a word is not removed from stack before it's been attached to all its dependent
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

![Diagram of a tree with nodes labeled root, dobj, nmod, det, case, Book, 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 → United)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights → morning)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights → the)</td>
</tr>
</tbody>
</table>

- Example of useful features

\[
\langle s_1.w = \text{flights}, op = \text{shift} \rangle \\
\langle s_2.w = \text{canceled}, op = \text{shift} \rangle \\
\langle s_1.t = \text{NNS}, op = \text{shift} \rangle \\
\langle s_2.t = \text{VBD}, op = \text{shift} \rangle \\
\langle b_1.w = \text{to}, op = \text{shift} \rangle \\
\langle b_1.t = \text{TO}, op = \text{shift} \rangle \\
\langle s_1.wt = \text{flightsNNS}, op = \text{shift} \rangle \\
\langle s_1.s_2t = \text{NNSVBD}, op = \text{shift} \rangle \\
\]
### Features example

<table>
<thead>
<tr>
<th>Source</th>
<th>Feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One word</strong></td>
<td></td>
</tr>
<tr>
<td>$s_1.w$</td>
<td>$s_1.t$</td>
</tr>
<tr>
<td>$s_2.w$</td>
<td>$s_2.t$</td>
</tr>
<tr>
<td>$b_1.w$</td>
<td>$b_1.w$</td>
</tr>
<tr>
<td><strong>Two word</strong></td>
<td></td>
</tr>
<tr>
<td>$s_1.w \circ s_2.w$</td>
<td>$s_1.t \circ s_2.t$</td>
</tr>
<tr>
<td>$s_1.t \circ s_2.wt$</td>
<td>$s_1.w \circ s_2.w \circ s_2.t$</td>
</tr>
<tr>
<td>$s_1.w \circ s_1.t \circ s_2.t$</td>
<td>$s_1.w \circ s_1.t$</td>
</tr>
</tbody>
</table>

**Figure 14.9** Standard feature templates for training transition-based dependency parsers. In the template specifications $s_n$ refers to a location on the stack, $b_n$ refers to a location in the word buffer, $w$ refers to the wordform of the input, and $t$ refers to the part of speech of the input.