Overview

• The parsing problem
• Methods
  – Transition-based parsing
• Evaluation
• Projectivity
Parse Trees

• Part-of-speech Tagging:
  – Word classes

• Parsing:
  – From words to phrases to sentences
  – Relations between words

• Two views
  – Dependency
  – Constituency
Dependency Parsing

- Dependency structure shows which words depend on (modify or are arguments of) which other words.

*The boy put the tortoise on the rug*
Constituency (Phrase Structure) Parsing

- Phrase structure organizes words into nested constituents
- Linguists can, and do, argue about details
- Lots of ambiguity

```
new art critics write reviews with computers
```
Dependency Structure

- Syntactic structure consists of:
  - Lexical items
  - Binary asymmetric relations → dependencies

Dependencies are typed with name of grammatical relation
Dependency Structure

- Syntactic structure consists of:
  - Lexical items
  - Binary asymmetric relations → dependencies

Diagram:

- **Bills** (nsubjpass)
- **submitted** (head)
  - Arrow from head to modifier (but can be reversed)
  - Modifier (dependent, inferior, subordinate)
  - Head (governor, superior, regent)
Dependency Structure

- Syntactic structure consists of:
  - Lexical items
  - Binary asymmetric relations \(\rightarrow\) dependencies

Dependencies form a tree

\[
\begin{align*}
\text{Bills} & \quad \text{were} \quad \text{submitted} \quad \text{by} \\
\text{ports} & \quad \text{on} \quad \text{cc} \quad \text{conj} \\
\text{and} & \quad \text{immigration} \\
\text{Senator} & \quad \text{Republican} \\
\text{Brownback} & \quad \text{nn} \quad \text{appos} \\
\text{Kansas} & \quad \text{pobj} \\
\end{align*}
\]
Dependency Structure

- Syntactic structure consists of:
  - Lexical items
  - Binary asymmetric relations \(\rightarrow\) dependencies

Dependencies form a tree
Let’s Parse

John saw Mary

He said that the boy who was wearing the blue shirt with the white pockets has left the building

Start with main verb, and draw dependencies. Don’t worry about labels. Just try to get the modifiers right.
Methods for Dependency Parsing

- Dynamic programming
  - Eisner (1996): $O(n^3)$

- Graph algorithms
  - McDonald et al. (2005): score edges independently using classifier and use maximum spanning tree

- Constraint satisfaction
  - Start with all edges, eliminate based on hard constraints

- “Deterministic parsing”
  - Left-to-right, each choice is done with a classifier

```
jumped
  
  nsubj
    
    boy
      
      det
        
        the
          
          det

over
  
  prep
    
    the
      
      det
        
        the
          
          det

amod
  
  the little
    
    det
      
      fence

```
Making Decisions

What are the sources of information for dependency parsing?
1. Billexical affinities
   – [issues ➔ the] is plausible
2. Dependency distance
   – mostly with nearby words
3. Intervening material
   – Dependencies rarely span intervening verbs or punctuation
4. Valency of heads
   – How many dependents on which side are usual for a head?

ROOT Discussion of the outstanding issues was completed.
MaltParse (Nivre et al. 2008)

• Greedy transition-based parser
• Each decision: how to attach each word as we encounter it
  – If you are familiar: like shift-reduce parser
• Select each action with a classifier
• The parser has:
  – a stack $\sigma$, written with the top to the right
    • which starts with the ROOT symbol
  – a buffer $\beta$, written with the top to the left
    • which starts with the input sentence
  – a set of dependency arcs $A$
    • which starts off empty
  – a set of actions
Arc-standard Dependency Parsing

Start: \( \sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset \)

- Shift \( \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A \)
- Left-Arc \( r \) \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\} \)
- Right-Arc \( r \) \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\} \)

Finish: \( \beta = \emptyset \)

ROOT Joe likes Marry
Arc-standard Dependency Parsing

Start: $\sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset$

- Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
- Left-Arc $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\}$
- Right-Arc $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\}$

Finish: $\beta = \emptyset$

(ROOT) Joe likes Marry

<table>
<thead>
<tr>
<th>Shift</th>
<th>Left-Arc $r$</th>
<th>Right-Arc $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ROOT]</td>
<td>[ROOT]</td>
<td>[ROOT]</td>
</tr>
<tr>
<td>[ROOT, Joe]</td>
<td>[likes, marry]</td>
<td>{((likes, Joe))} = $A_1$</td>
</tr>
<tr>
<td>[ROOT, likes]</td>
<td>[like]</td>
<td>$A_1$</td>
</tr>
<tr>
<td>[]</td>
<td>[ROOT]</td>
<td>$A_1 \cup {((\text{likes, Marry}))} = A_2$</td>
</tr>
<tr>
<td>[]</td>
<td>[ROOT]</td>
<td>$A_2 \cup {(\text{ROOT, likes})} = A_3$</td>
</tr>
<tr>
<td>[ROOT]</td>
<td>[]</td>
<td>$A_3$</td>
</tr>
</tbody>
</table>
Arc-standard Dependency Parsing

Start:  \( \sigma = \text{[ROOT]}, \beta = w_1, \ldots, w_n, A = \emptyset \)

- Shift  \( \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A \)
- Left-Arc\(_r\)  \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\} \)
- Right-Arc\(_r\)  \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\} \)

Finish:  \( \beta = \emptyset \)

ROOT Happy children like to play with their friends.
Arc-eager Dependency Parsing

Start: \[ \sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset \]

- **Left-Arc** \( r \)
  \[ \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\} \]
  - Precondition: \( r'(w_k, w_i) \notin A, w_i \neq \text{ROOT} \)

- **Right-Arc** \( r \)
  \[ \sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i, w_j)\} \]

- **Reduce**
  \[ \sigma|w_i, \beta, A \rightarrow \sigma, \beta, A \]
  - Precondition: \( r'(w_k, w_i) \in A \)

- **Shift**
  \[ \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A \]

Finish: \[ \beta = \emptyset \]

This is the common “arc-eager” variant: a head can immediately take a right dependent, before *its* dependents are found.
Arc-eager

Happy children like to play with their friends.
Happy children like to play with their friends.
You terminate as soon as the buffer is empty. Dependencies = $A_9$
MaltParser (Nivre et al. 2008)

• Selecting the next action:
  – Discriminative classifier (SVM, MaxEnt, etc.)
  – Untyped choices: 4
  – Typed choices: |R| * 2 + 2
• Features: POS tags, word in stack, word in buffer, etc.
• Greedy → no search
  – But can easily do beam search
• Close to state of the art
• Linear time parser → very fast!