Constituents vs. Dependencies

Traditional grammars model constituent structure: they capture the configurational patterns of sentences.

For example, verb phrases (VPs) have certain properties in English:

- a. I like ice cream. Do you ∅? (VP ellipsis)
- b. I like ice cream and hate bananas. (VP conjunction)
- c. I said I would hit Fred, and hit Fred I did. (VP fronting)

In other languages (e.g., German), there is little evidence for the existence of a VP constituent.
But from a semantic point of view, the important thing about verbs such as *like* is that they license two NPs:

- an *agent*, found in subject position or with nominative inflection;
- a *patient*, found in object position or with accusative inflection.

Which arguments are licensed, and which roles they play, depends on the verb (configuration is secondary).

To account for semantic patterns, we focus on dependency. Dependencies can be identified even in non-configurational languages.

A dependency structure consists of dependency relations, which are binary and asymmetric. A relation consists of:

- a head (H);
- a dependent (D);
- a label identifying the relation between H and D.

Formally, the dependency structure of a sentence is a graph with the words of the sentence as its nodes, linked by directed, labeled edges, with the following properties:

- **connected**: every node is related to at least one other node, and (through transitivity) to ROOT;
- **single headed**: every node (except ROOT) has exactly one incoming edge (from its head);
- **acyclic**: the graph cannot contain cycles of directed edges.

These conditions ensure that the dependency structure is a tree.

We distinguish projective and non-projective dependency trees:

A dependency tree is projective wrt. a particular linear order of its nodes if, for all edges $h \rightarrow d$ and nodes $w$, $w$ occurs between $h$ and $d$ in linear order only if $w$ is dominated by $h$. 

I heard Cecilia teach the horses to sing
A dependency tree is **non-projective** if \( w \) can occur between \( h \) and \( d \) in linear order without being dominated by \( h \).

![Dependency Trees: Projectivity](image)

A non-projective dependency grammar is not context-free. But efficient non-projective parsers exist, e.g., the MST parser.

In ANLP and FNLP, we’ve already seen various parsing algorithms for context-free languages (shift-reduce, CKY, active chart).

Why consider **dependency parsing** as a distinct topic?

- context-free parsing algorithms base their decisions on *adjacency*;
- in a dependency structure, a dependent need not be adjacent to its head (even if the structure is projective);
- we need new parsing algorithms to deal with non-adjacency (and with non-projectivity if present).

We will consider two types of dependency parsers:

1. **graph-based dependency parsing**, based on *maximum spanning trees* (MST parser, McDonald et al. 2005);

Alternative: map dependency trees to phrase structure trees and do standard CF parsing (not covered here).
### Graph-based Dependency Parsing

The score of a dependency edge \((i,j)\) is:

\[
s(i,j) = w \cdot f(i,j)
\]

where \(w\) is a weight vector and \(f(i,j)\) is a feature vector. Then the score of dependency tree \(y\) for sentence \(x\) is:

\[
s(x,y) = \sum_{(i,j) \in y} s(i,j) = \sum_{(i,j) \in y} w \cdot f(i,j)
\]

Dependency parsing is the task of finding the tree \(y\) with highest score for a given sentence \(x\).

### Maximum Spanning Tree Parsing

This task can be achieved using the following approach (McDonald et al. 2005):

- start with a totally connected graph \(G\), i.e., assume a directed edge between every pair of words;
- assume you have a scoring function that assigns a score \(s(i,j)\) to every edge \((i,j)\);
- find the maximum spanning tree (MST) of \(G\), i.e., the tree with the highest overall score that includes all nodes of \(G\);
- this is possible in \(O(n^2)\) time using the Chu-Liu-Edmonds algorithm; it finds both projective and non-projective MSTs;
- the correct parse is the MST of \(G\).

### Chu-Liu-Edmonds (CLE) Algorithm

Example: \(x = \text{John saw Mary}\), with graph \(G_x\). Start with the fully connected graph, with scores:

![Chu-Liu-Edmonds Algorithm Diagram](image-url)

Each node \(j\) in the graph greedily selects the incoming edge with the highest score \(s(i,j)\):

If a tree results, it is the maximum spanning tree. If not, there must be a cycle.
**CLE Algorithm: Recursion**

Identify the cycle and contract it into a single node and recalculate scores of incoming and outgoing edges:

```
root 40
   |     9
  40  
   |     30
   |  
  30  
   |   31
   |   saw
  30  
   |  
  30  
   |     30
   |    Mary
  30  
   |   John
  30  
   |  
  30  
   |   w js
  30 
```

Now call CLE recursively on this contracted graph. MST on the contracted graph is equivalent to MST on the original graph.

**CLE Algorithm: Reconstruction**

Now reconstruct the uncontracted graph: the edge from w js to Mary was from saw. The edge from ROOT to w js was a tree from ROOT to saw to John, so we include these edges too:

```
root 40
   |     10
  40  
   |     30
   |  
  30  
   |   saw
  30  
   |  
  30  
   |   30
   |    Mary
  30  
   |   John
  30  
   |  
  30  
   |   w js
  30 
```

This is a tree, hence it must be the MST of the graph.

**MST Parser: Features**

Parsing accuracy depends on the scoring function, i.e., the features \( f(i,j) \) and the weight vector \( w \).

The Margin Infused Relaxed Algorithm (MIRA) can be used to learn \( w \) (McDonald et al. 2005).

Features \( f(i,j) \) used:

<table>
<thead>
<tr>
<th>Unigram Features</th>
<th>Bigram Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-word, p-pos</td>
<td>p-word, p-pos, c-word, c-pos</td>
</tr>
<tr>
<td>p-word</td>
<td>p-word, p-pos, c-word, c-pos</td>
</tr>
<tr>
<td>p-pos</td>
<td>p-word, p-pos, c-word, c-pos</td>
</tr>
<tr>
<td>c-word, c-pos</td>
<td>p-word, p-pos, c-word, c-pos</td>
</tr>
<tr>
<td>c-word</td>
<td>p-word, p-pos, c-word, c-pos</td>
</tr>
<tr>
<td>c-pos</td>
<td>p-word, p-word, c-pos, c-pos</td>
</tr>
<tr>
<td></td>
<td>p-word, p-pos, c-pos</td>
</tr>
<tr>
<td></td>
<td>p-word, c-word</td>
</tr>
<tr>
<td></td>
<td>p-pos, c-pos</td>
</tr>
<tr>
<td></td>
<td>c-pos</td>
</tr>
</tbody>
</table>

Frank Keller | Natural Language Understanding | 20
More features:

<table>
<thead>
<tr>
<th>In Between PoS Features</th>
<th>p-pos, b-pos, c-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surrounding Word PoS Features</td>
<td>p-pos, p-pos+1, c-pos, c-pos</td>
</tr>
<tr>
<td></td>
<td>p-pos−1, p-pos, c-pos−1, c-pos</td>
</tr>
<tr>
<td></td>
<td>p-pos−1, p-pos+1, c-pos, c-pos+1</td>
</tr>
</tbody>
</table>

p: parent, c: child, b: between parent and child, −1: to left, +1: to right.

Total feature: 6,998,447 for English 13,450,672 for Czech.

The MST parser builds a dependency tree through graph surgery. The main alternative is the MALT parser (Nivre 2003):

- the MALT parser is a *shift-reduce parser*, defined through a transition system;
- for a given parse state, the transition system defines a set of actions $T$ which the parser can take;
- if more than one action is applicable, a classifier (e.g., an SVM) is used to decide which action to take;
- just like in the MST model, this requires a large set of hand-engineered features.
Summary

Comparing the MST and MALT parsers:

- the MST parser selects the globally optimal tree, given a set of edges with scores;
- it can naturally handle projective and non-projective trees;
- the MALT parser makes a sequence of local decisions about the best parse action;
- it can be extended to "pseudo-projective" dependency trees with transformation techniques;
- accuracy of MST and MALT is similar, but MALT is faster;
- both require a set of manually engineered featured and a model that learns feature weights.

Recent work on dependency parsing uses neural networks to learn both features and feature weights: next lecture.

References

