Lecture 19:
Dependency Grammars

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Today’s lecture

Dependency grammars

Dependency treebanks

Dependency parsing
The basic assumption underlying all varieties of dependency grammar is the idea that syntactic structure essentially consists of words linked by binary, asymmetrical relations called dependency relations (or dependencies for short). A dependency relation holds between a syntactically subordinate word, called the dependent, and another word on which it depends, called the head.

This is illustrated in figure 1.1, which shows a dependency structure for a simple English sentence, where dependency relations are represented by arrows pointing from the head to the dependent. Moreover, each arrow has a label, indicating the dependency type. For example, the noun news is a dependent of the verb had with the dependency type subject (SBJ). By contrast, the noun effect is a dependent of type object (OBJ) with the same head verb had.

One peculiarity of the dependency structure in figure 1.1 is that we have inserted an artificial word root before the first word of the sentence. This is a mere technicality, which simplifies both formal definitions and computational implementations. In particular, we can normally assume that every real word of the sentence should have a syntactic head. Thus, instead of saying that the verb had lacks a syntactic head, we can say that it is a dependent of the artificial word root.

In chapter 2, we will define dependency structures formally as labeled directed graphs, where nodes correspond to words (including root) and labeled arcs correspond to typed dependency relations.

The information encoded in a dependency structure representation is different from the information captured in a phrase structure representation, which is the most widely used type of syntactic representation in both theoretical and computational linguistics. This can be seen by comparing the dependency structure in figure 1.1 to a typical phrase structure representation for the same sentence, shown in figure 1.2. While the dependency structure represents head-dependent relations between words, classified by functional categories such as subject (SBJ) and object (OBJ), the phrase structure represents the grouping of words into phrases, classified by structural categories such as noun phrase (NP) and verb phrase (VP).
Dependency grammar

**Word-word dependencies** are a component of many (most/all?) grammar formalisms.

**Dependency grammar** assumes that syntactic structure consists *only* of dependencies.


DG is often used for **free word order languages**.

DG is **purely descriptive** (not a generative system like CFGs etc.), but some formal equivalences are known.
Different kinds of dependencies

**Head-argument** (‘exocentric’):  
*eat sushi*  
Arguments may be obligatory, but can only occur once.  
The head alone cannot necessarily replace the construction.

**Head-modifier** (‘endocentric’):  
*fresh sushi*  
Modifiers are optional, and can occur more than once.  
The head alone can replace the entire construction.

**Head-specifier** (‘exocentric’; Tesniere’s transfer):  
*the sushi*  
Between function words (e.g. prepositions, determiners)  
and their arguments. Syntactic head ≠ semantic head

**Coordination**: (Tesniere’s junction):  
*sushi and sashimi*  
Unclear where the head is.
What is a dependency?

Dependencies are (labeled) asymmetrical binary relations between two lexical items (words).

There is a syntactic relation between a head $H$ and a dependent $D$ in a construction $C$ if:

- the head $H$ determines the syntactic category of the construction $C$.
- the head $H$ determines the semantic category of the construction $C$; $D$ gives semantic specification.
- the head $H$ is obligatory. $D$ may be optional.
- the head selects $D$ and determines whether $D$ is obligatory or not.
- The form of $D$ depends on the head $H$ (agreement)
- The linear position of $D$ depends on the head $H$. 
Dependency structures

Dependencies form a graph over the words in a sentence.

This graph is **connected** (every word is a node) and (typically) **acyclic** (no loops).

**Single-head constraint:**
Every node has at most one incoming edge. This implies that the graph is a **rooted tree**.
From CFGs to dependencies

Assume each CFG rule has one head child (bolded) The other children are dependents of the head.

\[
\begin{align*}
S & \rightarrow \text{NP VP} \quad \text{VP is head, NP is a dependent} \\
\text{VP} & \rightarrow \text{V NP NP} \\
\text{NP} & \rightarrow \text{DT NOUN} \\
\text{NOUN} & \rightarrow \text{ADJ N}
\end{align*}
\]

The headword of a constituent is the terminal that is reached by recursively following the head child.

(here, \(V\) is the head word of \(S\), and \(N\) is the head word of \(NP\)).

If in rule \(XP \rightarrow X Y\), \(X\) is head child and \(Y\) dependent, the headword of \(Y\) depends on the headword of \(X\).

The maximal projection of a terminal \(w\) is the highest nonterminal in the tree that \(w\) is headword of.

Here, \(Y\) is a maximal projection.
Context-free grammars

CFGs capture only **nested** dependencies

The dependency graph is a **tree**

The dependencies **do not cross**
Beyond CFGs: Nonprojective dependencies

Dependencies: tree with crossing branches

Arise in the following constructions

- (Non-local) **scrambling** (free word order languages)
  
  *Die Pizza hat Klaus versprochen zu bringen*

- **Extraposition** *(The guy is coming who is wearing a hat)*

- **Topicalization** *(Cheeseburgers, I thought he likes)*
Dependency Treebanks

Dependency treebanks exist for many languages:

- Czech
- Arabic
- Turkish
- Danish
- Portuguese
- Estonian
- ....

Phrase-structure treebanks (e.g. the Penn Treebank) can also be translated into dependency trees (although there might be noise in the translation)
The Prague Dependency Treebank

Three levels of annotation:

**morphological**: [<2M tokens]
Lemma (dictionary form) + detailed analysis
(15 categories with many possible values = 4,257 tags)

**surface-syntactic (“analytical”)**: [1.5M tokens]
Labeled dependency tree encoding grammatical functions
(subject, object, conjunct, etc.)

**semantic (“tectogrammatical”)**: [0.8M tokens]
Labeled dependency tree for predicate-argument structure,
information structure, coreference (not all words included)
(39 labels: agent, patient, origin, effect, manner, etc....)
Examples: analytical level
Turkish is an agglutinative language with free word order.

- Rich morphological annotations
- Dependencies (next slide) are at the morpheme level

- iyileştiriyo\n\n  - (literally) while it is being caused to become good
  - while it is being improved
- iyi+Adj ^DB+Verb+Become\n  ^DB+Verb+Caus
  ^DB+Verb+Pass+Pos+Pres^DB+Adverb+While

Very small -- about 5000 sentences
METU-Sabancı Turkish Treebank

This school-at+that-is student-s-1 most intelligence+with+of there standing little girl+is

The most intelligent of the students in this school is the little girl standing there.

[this and prev. example from Kemal Oflazer’s talk at Rochester, April 2007]
Dependency or phrase structure annotation?

No clear consensus which is better. May depend on the language.

It may also depend on the annotation guidelines:
- Early phrase-structure treebanks (Penn Treebank) are not explicit enough (not all nodes have function tags).
- Dependency treebanks (e.g. Sabanci) often omit long-range dependencies.
- They also can’t express scope relations.
Parsing algorithms for DG

‘Transition-based’ parsers:
learn a sequence of actions to parse sentences

Models:
State = stack of partially processed items
+ queue/buffer of remaining tokens
+ set of dependency arcs that have been found already
Transitions (actions) = add dependency arcs; stack/queue operations

‘Graph-based’ parsers:
learn a model over dependency graphs

Models:
a function (typically sum) of local attachment scores
Transition-based parsing (Nivre et al.)
Transition-based parsing

Transition-based shift-reduce parsing processes the sentence \( S = w_0 w_1 \ldots w_n \) from left to right. Unlike CKY, it constructs a single tree.

N.B: this only works for projective dependency trees

Notation:

- \( w_0 \) is a special ROOT token.
- \( V_S = \{w_0, w_1, \ldots, w_n\} \) is the vocabulary of the sentence
- \( R \) is a set of dependency relations

The parser uses three data structures:

- \( \sigma \): a stack of partially processed words \( w_i \in V_S \)
- \( \beta \): a buffer of remaining input words \( w_i \in V_S \)
- \( A \): a set of dependency arcs \( (w_i, r, w_j) \in V_S \times R \times V_S \)
Parser configurations \((\sigma, \beta, A)\)

The **stack** \(\sigma\) is a list of **partially processed words**

- We push and pop words onto/off of \(\sigma\).
- \(\sigma|w\) : \(w\) is on top of the stack.
- Words on the stack are not (yet) attached to any other words.
- Once we attach \(w\), \(w\) can’t be put back onto the stack again.

The **buffer** \(\beta\) is the **remaining input words**

- We read words from \(\beta\) (left-to-right) and push them onto \(\sigma\).
- \(w|\beta\) : \(w\) is on top of the buffer.

The **set of arcs** \(A\) defines the **current tree**.

- We can add new arcs to \(A\) by attaching the word on top of the stack to the word on top of the buffer, or vice versa.
Parser configurations \((\sigma, \beta, A)\)

We start in the initial configuration \(([w_0], [w_1, \ldots, w_n], \{\})\) 
(Root token, Input Sentence, Empty tree)
We can attach the first word \((w_1)\) to the root token \(w_0\),
or we can push \(w_1\) onto the stack.
\((w_0\) is the only token that can’t get attached to any other word) 

We want to end in the terminal configuration \(([], [], A)\) 
(Empty stack, Empty buffer, Complete tree)
Success!
We have read all of the input words (empty buffer) and have attached all input words to some other word (empty stack)
Parser actions

$(\sigma, \beta, A)$: Parser configuration with stack $\sigma$, buffer $\beta$, set of arcs $A$

$(w, r, w')$: Dependency with head $w$, relation $r$ and dependent $w'$

**SHIFT:** Push the next input word from the buffer $\beta$ onto the stack $\sigma$

$$(\sigma, \text{w}_i|\beta, A) \Rightarrow (\sigma|\text{w}_i, \beta, A)$$

**LEFT-ARC$_r$:** ... $\text{w}_i...\text{w}_j...$

Attach dependent $\text{w}_i$ (top of stack $\sigma$) to head $\text{w}_j$ (top of buffer $\beta$) with relation $r$ from $\text{w}_j$ to $\text{w}_i$. Pop $\text{w}_i$ off the stack.

$$(\sigma|\text{w}_i, \text{w}_j|\beta, A) \Rightarrow (\sigma, \text{w}_j|\beta, A \cup \{(\text{w}_j, r, \text{w}_i)\})$$

**RIGHT-ARC$_r$:** ...$\text{w}_i...\text{w}_j...$

Attach dependent $\text{w}_j$ (top of buffer $\beta$) to head $\text{w}_i$ (top of stack $\sigma$) with relation $r$ from $\text{w}_i$ to $\text{w}_j$. Move $\text{w}_i$ back to the buffer

$$(\sigma|\text{w}_i, \text{w}_j|\beta, A) \Rightarrow (\sigma, \text{w}_i|\beta, A \cup \{(\text{w}_i, r, \text{w}_j)\})$$
The basic assumption underlying all varieties of dependency grammar is the idea that syntactic structure essentially consists of words linked by binary, asymmetrical relations called dependency relations (or dependencies for short). A dependency relation holds between a syntactically subordinate word, called the dependent, and another word on which it depends, called the head. This is illustrated in figure 1.1, which shows a dependency structure for a simple English sentence, where dependency relations are represented by arrows pointing from the head to the dependent.

Moreover, each arrow has a label, indicating the dependency type. For example, the noun news is a dependent of the verb had with the dependency type subject (SBJ). By contrast, the noun effect is a dependent of type object (OBJ) with the same head verb had. Note also that the noun news is itself a syntactic head in relation to the word Economic, which stands in the attribute (ATT) relation to its head noun.

One peculiarity of the dependency structure in figure 1.1 is that we have inserted an artificial word root before the first word of the sentence. This is a mere technicality, which simplifies both formal definitions and computational implementations. In particular, we can normally assume that every real word of the sentence should have a syntactic head. Thus, instead of saying that the verb had lacks a syntactic head, we can say that it is a dependent of the artificial word root. In chapter 2, we will define dependency structures formally as labeled directed graphs, where nodes correspond to words (including root) and labeled arcs correspond to typed dependency relations.

The information encoded in a dependency structure representation is different from the information captured in a phrase structure representation, which is the most widely used type of syntactic representation in both theoretical and computational linguistics. This can be seen by comparing the dependency structure in figure 1.1 to a typical phrase structure representation for the same sentence, shown in figure 1.2. While the dependency structure represents head-dependent relations between words, classified by functional categories such as subject (SBJ) and object (OBJ), the phrase structure represents the grouping of words into phrases, classified by structural categories such as noun phrase (NP) and verb phrase (VP).
Economic news had little effect on financial markets.

### Transition Configuration

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
<th>Arc Set</th>
<th>Transition Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH ⇒ ([root], [Economic, ... , .], ∅)</td>
<td>([root])</td>
<td>news, ... , .]</td>
<td>A1 = {(news, ATT, Economic)}</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [news, ... , .], ∅)</td>
<td>([root])</td>
<td>had, ... , .]</td>
<td>A1</td>
</tr>
<tr>
<td>LA_{SBJ} ⇒ ([root], [had, ... , .], ∅)</td>
<td>([root])</td>
<td>A2 = A1 ∪{(had, SBJ, news)}</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root, had], [little, ... , .], A2)</td>
<td>([root, had])</td>
<td>A3 = A2 ∪{(effect, ATT, little)}</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root, had, little], [effect, ... , .], A2)</td>
<td>([root, had, little])</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root, had], [effect, ... , .], A3)</td>
<td>([root, had])</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root, had, effect], [on, ... , .], A3)</td>
<td>([root, had, effect])</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root, . . on], [financial, markets, .], A3)</td>
<td>([root, . . on])</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root, . . financial], [markets, .], A3)</td>
<td>([root, . . financial])</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root, . . on], [markets, .], A4)</td>
<td>([root, . . on])</td>
<td>A4 = A3 ∪{(markets, ATT, financial)}</td>
<td></td>
</tr>
<tr>
<td>RA_{PC} ⇒ ([root, had, effect], [on, .], A5)</td>
<td>([root, had, effect])</td>
<td>A5 = A4 ∪{(on, PC, markets)}</td>
<td></td>
</tr>
<tr>
<td>RA_{ATT} ⇒ ([root, had], [effect, .], A6)</td>
<td>([root, had])</td>
<td>A6 = A5 ∪{(effect, ATT, on)}</td>
<td></td>
</tr>
<tr>
<td>RA_{OBJ} ⇒ ([root], [had, .], A7)</td>
<td>([root])</td>
<td>A7 = A6 ∪{(had, OBJ, effect)}</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root, had], [\ .], A7)</td>
<td>([root, had])</td>
<td>A7</td>
<td></td>
</tr>
<tr>
<td>RA_{PU} ⇒ ([root], [had], A8)</td>
<td>([root])</td>
<td>A8 = A7 ∪{(had, PU, .)}</td>
<td></td>
</tr>
<tr>
<td>RA_{PRED} ⇒ ([\ .], [root], A9)</td>
<td>([\ .])</td>
<td>A9 = A8 ∪{(root, PRED, had)}</td>
<td></td>
</tr>
<tr>
<td>SH ⇒ ([root], [\ .], A9)</td>
<td>([root])</td>
<td>A9</td>
<td></td>
</tr>
</tbody>
</table>
Transition-based parsing in practice

Which action should the parser take under the current configuration?

We also need a parsing model that assigns a score to each possible action given a current configuration.

Possible actions:
- **SHIFT**, and for any relation $r$: **LEFT-ARC$_r$**, or **RIGHT-ARC$_r$**

Possible features of the current configuration:
- The top $\{1,2,3\}$ words on the buffer and on the stack, their POS tags, etc.

We can learn this model from a dependency treebank.