A dependency parse

Dependency grammar

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

Dependency grammar assumes that syntactic structure consists only of dependencies. Many variants. Modern DG began with Tesniere (1959).

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.

S → NP VP
VP → V NP NP
NP → DT NOUN
NOUN → ADJ N

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 → 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

- iyileştirmiyor
  - (literally) while it is being caused to become good
  - while it is being improved

iyi+Adj ^DB+Verb+Become^DB+Verb+Poss

Very small -- about 5000 sentences

METU-Sabanci Turkish Treebank

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 shift-reduce parsing processes the sentence $S = w_0w_1...w_n$ 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.
- $V_S = \{w_0, w_1, ..., w_n\}$ 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 \((\sigma, \beta, \Lambda)\)

We start in the initial configuration \([w_0], [w_1, \ldots, w_n], \emptyset\)  
(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 \(([], [], \Lambda)\)  
(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

\((\alpha, \beta, \Lambda):\) Parser configuration with stack \(\alpha\), buffer \(\beta\), set of arcs \(\Lambda\)

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

**SHIFT:** Push the next input word from the buffer \(\beta\) onto the stack \(\alpha\)

\((\alpha, w_i|\beta, \Lambda) \Rightarrow (\alpha|w_i, \beta, \Lambda)\)

**LEFT-ARC:** \(\ldots w_i \ldots w_j \ldots\)

Attach dependent \(w_i\) (top of stack \(\alpha\)) to head \(w_j\) (top of buffer \(\beta\))
with relation \(r\) from \(w_i\) to \(w_j\). Pop \(w_j\) off the stack.

\((\alpha|w_i, w_j|\beta, \Lambda) \Rightarrow (\alpha, w_i|\beta, \Lambda \cup \{(w_j, r, w_i)\})\)

**RIGHT-ARC:** \(\ldots w_i \ldots w_j \ldots\)

Attach dependent \(w_j\) (top of buffer \(\beta\)) to head \(w_i\) (top of stack \(\alpha\))
with relation \(r\) from \(w_i\) to \(w_j\). Move \(w_i\) back to the buffer

\((\alpha|w_i, w_j|\beta, \Lambda) \Rightarrow (\alpha, w_i|\beta, \Lambda \cup \{(w_i, r, w_j)\})\)

---

An example sentence & parse

Economic news had little effect on financial markets.

---

Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\text{root}], \text{Economic}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([\text{root}, \text{Economic}], \text{news}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>LA (\Rightarrow) ([\text{root}], \text{news}, \ldots], \text{had}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([\text{root}, \text{had}, \text{effect}], \text{on}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([\text{root}, \ldots], \text{financial}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>LA (\Rightarrow) ([\text{root}, \ldots], \text{markets}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>RA (\Rightarrow) ([\text{root}, \text{had}, \text{effect}], \text{on}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>RA (\Rightarrow) ([\text{root}, \text{had}, \text{effect}], \text{on}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>RA (\Rightarrow) ([\text{root}, \text{had}, \text{effect}], \text{on}, \ldots], \emptyset)</td>
</tr>
<tr>
<td>RA (\Rightarrow) ([\text{root}, \text{had}, \text{effect}], \text{on}, \ldots], \emptyset)</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.