Dependency Parsing

25 March 2021
What’s the main verb of this sentence?
What’s the main verb of this sentence?
Dependency trees highlight relationships between individual words rather than constituents.

These relationships are represented as edges between words in a graph.
Dependency trees can help highlight the similarities between superficially different sentences:

Gerald gave awards to puppies.

Gerald gave puppies awards.
Constituents leave out a lot of semantic information.

From a semantic point of view, the important thing about verbs such as *like* is that they allow two NPs:

an *agent*, found in subject position (or with nominative inflection), and

a *patient*, found in object position (or with accusative inflection).

Which arguments are allowed and which roles they play depends on the verb.

To account for semantic patterns, we use *dependency*. 
Common relations

Gerald gave puppies awards for bravery.
As relationships between words are quite useful in NLP, dependency parsing has grown in popularity.
Dependency grammars
A *dependency grammar* describes structure with *dependencies* – binary, asymmetric relations between words.
Example dependency relations

Argument dependencies

- **nsubj**: nominal subject
- **csubj**: clausal subject
- **dobj**: direct object
- **iobj**: indirect object
- **pobj**: object of preposition

Modifier dependencies

- **tmod**: temporal modifier
- **appos**: appositional modifier
- **det**: determiner
- **prep**: prepositional modifier
One word is the head of the sentence.

All other words are either

  dependent on the head word or
  dependent on some other word that connects to the head word
  through a sequence of dependencies.
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 \textit{ROOT}.

- **Single-headed**: Every node (except \textit{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.
Example dependency parse

- **pred**
  - **amod**
    - **nsubj**
      - *Economic*
  - **news**
  - **had**

- **dobj**
  - **amod**
    - **little**
  - **effect**
  - **on**
    - **pobj**
      - **amod**
        - *financial*
      - *markets*

- **punct**
Universal dependencies
Universal dependencies project

Set of universal dependency relations that are

  linguistically motivated
  computationally useful
  cross-linguistically applicable

universaldependencies.org
- **acl**: clausal modifier of noun (adjectival clause)
- **advcl**: adverbial clause modifier
- **advmod**: adverbial modifier
- **amod**: adjectival modifier
- **appos**: appositional modifier
- **aux**: auxiliary
- **case**: case marking
- **cc**: coordinating conjunction
- **ccomp**: clausal complement
- **clf**: classifier
- **compound**: compound
- **conj**: conjunct
- **cop**: copula
- **csubj**: clausal subject
- **dep**: unspecified dependency
- **det**: determiner
- **discourse**: discourse element
- **dislocated**: dislocated elements
- **expl**: expletive
- **fixed**: fixed multiword expression
- **flat**: flat multiword expression
- **goeswith**: goes with
- **iobj**: indirect object
- **list**: list
- **mark**: marker
- **nmod**: nominal modifier
- **nsubj**: nominal subject
- **nummod**: numeric modifier
- **obj**: object
- **obi**: oblique nominal
- **orphan**: orphan
- **parataxis**: parataxis
- **punct**: punctuation
- **reparandum**: overridden disfluency
- **root**: root
- **vocative**: vocative
- **xcomp**: open clausal complement
The “universal” in Universal Dependencies

<table>
<thead>
<tr>
<th>Language</th>
<th>Tokens</th>
<th>Type</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>49K</td>
<td>IE, Germanic</td>
<td></td>
</tr>
<tr>
<td>Akkadian</td>
<td>1K</td>
<td>Afro-Asiatic, Semitic</td>
<td></td>
</tr>
<tr>
<td>Amharic</td>
<td>10K</td>
<td>Afro-Asiatic, Semitic</td>
<td></td>
</tr>
<tr>
<td>Ancient Greek</td>
<td>417K</td>
<td>IE, Greek</td>
<td></td>
</tr>
<tr>
<td>Arabic</td>
<td>1,042K</td>
<td>Afro-Asiatic, Semitic</td>
<td></td>
</tr>
<tr>
<td>Armenian</td>
<td>22K</td>
<td>IE, Armenian</td>
<td></td>
</tr>
<tr>
<td>Bambara</td>
<td>13K</td>
<td>Mande</td>
<td></td>
</tr>
<tr>
<td>Basque</td>
<td>121K</td>
<td>Basque</td>
<td></td>
</tr>
<tr>
<td>Belarusian</td>
<td>8K</td>
<td>IE, Slavic</td>
<td></td>
</tr>
<tr>
<td>Breton</td>
<td>10K</td>
<td>IE, Celtic</td>
<td></td>
</tr>
<tr>
<td>Bulgarian</td>
<td>156K</td>
<td>IE, Slavic</td>
<td></td>
</tr>
<tr>
<td>Buryat</td>
<td>10K</td>
<td>Mongolic</td>
<td></td>
</tr>
<tr>
<td>Cantonese</td>
<td>6K</td>
<td>Sino-Tibetan</td>
<td></td>
</tr>
<tr>
<td>Catalan</td>
<td>531K</td>
<td>IE, Romance</td>
<td></td>
</tr>
<tr>
<td>Chinese</td>
<td>160K</td>
<td>Sino-Tibetan</td>
<td></td>
</tr>
<tr>
<td>Classical Chinese</td>
<td>34K</td>
<td>Sino-Tibetan</td>
<td></td>
</tr>
<tr>
<td>Coptic</td>
<td>22K</td>
<td>Afro-Asiatic, Egyptian</td>
<td></td>
</tr>
<tr>
<td>Croatian</td>
<td>197K</td>
<td>IE, Slavic</td>
<td></td>
</tr>
<tr>
<td>Czech</td>
<td>2,222K</td>
<td>IE, Slavic</td>
<td></td>
</tr>
<tr>
<td>Danish</td>
<td>100K</td>
<td>IE, Germanic</td>
<td></td>
</tr>
<tr>
<td>Dutch</td>
<td>307K</td>
<td>IE, Germanic</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>586K</td>
<td>IE, Germanic</td>
<td></td>
</tr>
<tr>
<td>Erzya</td>
<td>15K</td>
<td>Uralic, Mordvin</td>
<td></td>
</tr>
<tr>
<td>Estonian</td>
<td>44K</td>
<td>Uralic, Finnic</td>
<td></td>
</tr>
</tbody>
</table>

Approx. 90 languages
Projectivity
Projective dependency trees have no crossing dependency arcs.
John saw a dog yesterday which was a Yorkshire Terrier.
Non-projective dependencies are more frequent in languages with flexible word order, like German, Dutch, or Czech.

... dat ik Cecilia de paarden hoord leren zingen
... that I Cecilia the horses heard teach sing
How frequent are non-projective structures?

<table>
<thead>
<tr>
<th>Language</th>
<th>%NPD</th>
<th>%NPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>5.4</td>
<td>36.4</td>
</tr>
<tr>
<td>German</td>
<td>2.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>Slovene</td>
<td>1.9</td>
<td>22.2</td>
</tr>
<tr>
<td>Portuguese</td>
<td>1.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Danish</td>
<td>1.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>

NPD = non-projective dependencies
NPS = non-projective sentences

Statistics from CoNLL shared task
Formally, a dependency tree is *projective* with respect to a particular linear order of its nodes (words) 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 $h$. 
Projective trees can be described with context-free grammars.
Non-projective grammars are not context-free, which makes parsing harder, but it lets us handle these constructions without doing weird things.

Traces: Making trees weird™
Dependency parsing
Why consider dependency parsing as a distinct topic from constituent parsing?

Context-free parsing algorithms base their decisions on \textit{adjacency}, but 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).
Data-driven dependency parsing

Goal: Learn a good predictor of dependency graphs

Input: Sentence
Output: Dependency graph/tree $G = (V, A)$

This can be framed as a “structured prediction” task

Very large output space
With interdependent labels
There are two dominant approaches:

*graph-based dependency* parsing, based on maximum spanning trees (MST parser)

*transition-based dependency parsing*, an extension of shift-reduce parsing (MALT parser)
Another possibility we won’t go into:

Map dependency trees to phrase structure trees
Do standard CFG parsing (for projective trees) or LCFRS variants (for non-projective trees).
Each of these approach arises from different views of syntactic structure:

as a *set of constraints* (MST),
as the *actions of an automaton* (transition-based),
or as the *derivations of a grammar* (CFG parsing).

*It’s often possible to translate between these views, albeit with some effort.*
Graph-based dependency parsing
We want to find the highest-scoring dependency tree in the space of all possible trees for a sentence.
Let $x = x_1 \ldots x_n$ be the input sentence.

Let $y$ be a dependency tree for $x$.

That is, $y$ is a set of dependency edges $(i, j)$ where there’s an edge from $x_i$ to $x_j$ in the dependency tree.
Since each word has exactly one parent, this is like a tagging problem.

The possible tags are the other words in the sentence (or the dummy node root).

We can “edge factorize” the score of a tree so that it’s simply the product of its edge scores.

Then we can select the best incoming edge for each word.

One extra constraint: The result needs to be a tree!
Formally, we can say that the score of a dependency edge \((i, j)\) is a function \(s(i, j)\).

Then the score of the dependency tree \(y\) for sentence \(x\) is:

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

Dependency parsing is the task of finding the tree \(y\) with highest score for a given sentence \(x\).
The best dependency parse is the *maximum spanning tree* (MST) of a fully connected graph of the words:

- Construct a graph $G$ with 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 of $G$.

---

*Ok, but how?*
Chu–Liu–Edmonds (CLE) algorithm

\[ x = \text{John saw Mary} \]

Graph \( G_x = \) 

Start with the fully connected graph, with scores
Chu–Liu–Edmonds (CLE) algorithm

Each node \( j \) in the graph greedily selects the incoming edge with \( d \), the highest score.

If a tree results, it’s the maximum spanning tree!

If not, there must be a cycle.
Chu–Liu–Edmonds (CLE) algorithm

Each node \( j \) in the graph greedily selects the incoming edge with \( d \), the highest score.

If a tree results, it’s the maximum spanning tree!

If not, there must be a cycle.
Chu–Liu–Edmonds (CLE) algorithm

*Intuition:* We can break the cycle if we replace a single incoming edge to one of the nodes in the cycle.

Which one? Decide recursively.
CLE algorithm: Recursion

Identify the cycle and contract it into a single node. Recalculate scores of incoming and outgoing edges.

Intuition: Edges into the cycle are the weight of the cycle with only the dependency of the target word changed.

Now call CLE recursively on this contracted graph.

MST on the contracted graph is equivalent to MST on the original graph.
CLE algorithm: Recursion

Again, greedily collect incoming edges to all nodes:

This is a tree, so it must be the MST of the graph!
CLE Algorithm: Reconstruction

Now we reconstruct the uncontracted graph:

The edge from \( w_{js} \) to \( Mary \) was from \( saw \).

The edge from \( \text{ROOT} \) to \( w_{js} \) was a tree from \( \text{ROOT} \) to \( saw \) to \( John \), so we include these edges too:
Where do we get edge scores \( s(i, j) \) from?

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

For the decade after 2005: linear model trained with clever variants of support vector machines (SVMs), etc.

More recently, neural networks, of course.
Scoring edges with a neural network

There are a few different formulations of this.

An effective one from Zhang and Lapata (2016):

\[ s(i, j) = P_{\text{head}}(w_j | w_i, x) = \frac{\exp(g(a_j, a_i))}{\sum_{k=0}^{|x|} \exp(g(a_k, a_i))} \]

We get \( a_i \) by concatenating the hidden states of a forward and backward RNN at position \( i \).

The function \( g(a_j, a_i) \) computes an association score telling us how much word \( w_i \) prefers word \( w_j \) as its head. A simple option from among many:

\[ g(a_j, a_i) = v_a^1 \cdot \tanh(U_a \cdot a_j + W_a \cdot a_i) \]

Association scores are a useful way to select from a dynamic group of candidates, and underlie the idea of attention used in machine translation.
Transition-based dependency parsing
An MST parser builds a dependency tree through graph surgery.

An alternative is \textit{transition-based parsing}:

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 mechanism to compute scores over a set of candidates.
Transition-based dependency parsing

The *arc-standard transition* system:

Configuration $c = (s, b, A)$ with stack $s$, buffer $b$, set of dependency arcs $A$;

initial configuration for sentence $w_1, \ldots, w_n$ is

- $s = [\text{ROOT}]$
- $b = [w_1, \ldots, w_n]$
- $A = \emptyset$

$c$ is terminal if buffer is empty, stack contains only $\text{ROOT}$, and parse tree is given by $A_c$;

if $s_i$ is the $i$th top element on stack, and $b_i$ the $i$th element on buffer, then we have the transitions on the right.

**LEFT-ARC($l$):**
Adds arc $s_1 \rightarrow s_2$ with label $l$ and removes $s_2$ from stack
Precondition: $|s| \geq 2$

**RIGHT-ARC($l$):**
Adds arc $s_2 \rightarrow s_1$ with label $l$ and removes $s_1$ from stack
Precondition: $|s| \geq 2$

**SHIFT:**
Moves $b_1$ from buffer to stack
Precondition: $|b| \geq 1$
Transition-based dependency parsing

ROOT  He  has  good  control .
PRP  VBZ  JJ  NN .

<table>
<thead>
<tr>
<th>Transition</th>
<th>Stack</th>
<th>Buffer</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>[ROOT]</td>
<td>[He has good control .]</td>
<td>Ø</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT He]</td>
<td>[has good control .]</td>
<td>AU nsbj(has,He)</td>
</tr>
<tr>
<td>LEFT-ARC(nsubj)</td>
<td>[ROOT has]</td>
<td>[good control .]</td>
<td>AU amod(control,good)</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT has good]</td>
<td>[control .]</td>
<td>AU dobj(has,control)</td>
</tr>
<tr>
<td>LEFT-ARC(amod)</td>
<td>[ROOT has good control]</td>
<td>[.]</td>
<td></td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT has control]</td>
<td>[.]</td>
<td></td>
</tr>
<tr>
<td>RIGHT-ARC(dobj)</td>
<td>[ROOT has]</td>
<td>[.]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>[ROOT]</td>
<td>AU root(ROOT,has)</td>
</tr>
</tbody>
</table>
Comparison

The MST parser selects the globally optimal tree, given a set of edges with scores.

It can naturally handle projective and non-projective trees.

A transition-based parser makes a sequence of local decisions about the best parse action.

It can be extended to projective dependency trees by changing the transition set.

Both require dynamic classifiers for edge scores

These can be implemented using neural networks, conditioned on bidirectional RNN encodings of the sentence.
Dependency parsing nowadays

Two main approaches:

- Graph-based: better results
- Transition-based: far faster

Google uses transition-based dependency parsing for their machine-translation systems.

Current state-of-the-art results: 93.9%

- Merge of transition-based and graph-based.
What to use

Popular/effective Python tools:

StanfordNLP / Stanza for universal dependencies or Stanford dependencies
Kitaev parser for PTB
spaCy for non-UD dependencies – especially fast/easy

Performance is in the 90s on most intuitive measures of accuracy for clean English data.

Usable performance on many more languages/settings.
Demos

Stanford CoreNLP:

https://corenlp.run

spaCy:

https://explosion.ai/demos/displacy
Syntactic analysis recap

Part-of-speech (POS) tagging

Figuring out the syntactic type of each word
Necessary ingredient in parsing

Parsing

Constituency parsing

Breaking down sentences into meaningful parts
Important background for standard compositional semantics and natural logic

Dependency parsing

Working out relationships between words in sentences
Helpful for feature engineering, relation extraction, and semantic parsing for question-answering.
Acknowledgments

The lecture incorporates material from:

Sam Bowman, New York University
Marine Carpuat, University of Maryland
Nancy Ide, Vassar College
Daniel Jurafsky and James Martin, *Speech and Language Processing*