Empirical Methods in Natural Language Processing
Lecture 18: Dependency Parsing

(transition-based slides from Harry Eldridge)

3 April 2019
Linguists have long observed that the meanings of words within a sentence depend on one another, mostly in asymmetric, binary relations.

- Though some constructions don’t cleanly fit this pattern: e.g., coordination, relative clauses.
Dependency Parse

Equivalently, but showing word order (**head → modifier**):

Because it is a tree, every word has exactly one parent.
Some treebanks prefer **content heads**: 

Little kids were always watching birds with fish

Others prefer **functional heads**: 

Little kids were always watching birds with fish
Edge Labels

It is often useful to distinguish different kinds of head → modifier relations, by labeling edges:

Important relations for English include subject, direct object, determiner, adjective modifier, adverbial modifier, etc. (Different treebanks use somewhat different label sets.)

- How would you identify the subject in a constituency parse?
Dependency Paths

For information extraction tasks involving real-world relationships between entities, chains of dependencies can provide good features:

British officials in Tehran have been meeting with their Iranian counterparts

(example from Brendan O’Connor)
Projectivity

- A sentence’s dependency parse is said to be projective if every subtree (node and all its descendants) occupies a contiguous span of the sentence.

- The dependency parse can be drawn on top of the sentence without any crossing edges.

A hearing on the issue is scheduled today
Nonprojectivity

• Other sentences are **nonprojective**:  

```
A hearing is scheduled on the issue today
```

• Nonprojectivity is rare in English, but quite common in many languages.
Dependency Parsing

Some of the algorithms you have seen for PCFGs can be adapted to dependency parsing.

- **CKY** can be adapted, though efficiency is a concern: obvious approach is $O(Gn^5)$; Eisner algorithm brings it down to $O(Gn^3)$
  
Transition-based Parsing

- Adapts shift-reduce methods: stack and buffer

- Remember: latent structure is just edges between words. Train a classifier to predict next action (shift, reduce, attach-left, or attach-right), and proceed left-to-right through the sentence. $O(n)$ time complexity!

- Only finds projective trees (without special extensions)

- Pioneering system: Nivre’s MALT_PARSER

- See http://spark-public.s3.amazonaws.com/nlp/slides/Parsing-Dependency.pdf (Jurafsky & Manning Coursera slides) for details and examples
Graph-based Parsing

- **Global algorithm**: From the fully connected directed graph of all possible edges, choose the best ones that form a tree.

- **Edge-factored** models: Classifier assigns a nonnegative score to each possible edge; **maximum spanning tree** algorithm finds the spanning tree with highest total score in $O(n^2)$ time.
  - Edge-factored assumption can be relaxed (higher-order models score larger units; more expensive).
  - Unlabeled parse $\rightarrow$ edge-labeling classifier (pipeline).

- **Pioneering work**: McDonald’s **MSTParser**

- **Can be formulated as constraint-satisfaction with integer linear programming** (Martins’s **TURBOParser**)

Graph-based vs. Transition-based vs. Conversion-based

- TB: Features in scoring function can look at any part of the stack; no optimality guarantees for search; linear-time; (classically) projective only

- GB: Features in scoring function limited by factorization; optimal search within that model; quadratic-time; no projectivity constraint

- CB: In terms of accuracy, sometimes best to first constituency-parse, then convert to dependencies (e.g., Stanford Parser). Slower than direct methods.
Dependency Parsing Evaluation

For training and evaluation, we can automatically convert constituency treebanks (like the Penn Treebank) to dependencies—see below—or we can use dependency treebanks like Universal Dependencies, available in many languages (http://universaldependencies.org).

Standard metrics for comparing against a gold standard are:

- **UAS** (unlabeled attachment score): % of words attached correctly (correct head)

- **LAS** (labeled attachment score): % of words attached to the correct head with the correct relation label
Choosing a Parser: Criteria

• Target representation: constituency or dependency?

• Efficiency? In practice, both runtime and memory use.

• Incrementality: parse the whole sentence at once, or obtain partial left-to-right analyses/expectations?

• Retrainable system?
Advanced Topic: Relationship between constituency and dependency parses

Constituency parses/grammars can be extended with a notion of lexical head, which can

- improve constituency parsing, or

- help convert a constituency parse to a dependency parse
Vanilla PCFGs: no lexical dependencies

Replacing one word with another with the same POS will never result in a different parsing decision, even though it should!

- kids saw birds with fish vs. kids saw birds with binoculars
- She stood by the door covered in tears vs. She stood by the door covered in ivy
- stray cats and dogs vs. Siamese cats and dogs
A way to fix PCFGs: lexicalization

Create new categories, this time by adding the lexical head of the phrase (note: N level under NPs not shown for brevity):

S-saw

NP-kids

kids

VP-saw

VP-saw

V-saw

saw

NP-birds

birds

PP-binoculars

P-with

with

NP-binoculars

binoculars

• Now consider:

VP-saw → VP-saw PP-fish vs. VP-saw → VP-saw PP-binoculars
How to get lexical heads?
Head Rules

The standard solution is to use head rules: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. $S \rightarrow NP \ VP$, $VP \rightarrow VP \ PP$, $PP \rightarrow P \ NP$ (content head), etc.

- Heuristics to scale this to large grammars: e.g., within an NP, last immediate N child is the head.
Head Rules

Then, propagate heads up the tree:

```
S
  NP-kids
    kids
  VP
    VP
      V-saw
        saw
      NP-birds
        birds
    PP
      P-with
        with
      NP-binoculars
        binoculars
```
Then, propagate heads up the tree:

- **S**
  - NP-kids
    - kids
  - VP
    - VP-saw
      - V-saw
        - saw
      - NP-birds
        - birds
    - PP
      - P-with
        - with
      - NP-binoculars
        - binoculars
Head Rules

Then, propagate heads up the tree:

```
S
  NP-kids
    kids
  VP
    VP-saw
      V-saw
        saw
      NP-birds
        birds
  PP-binoculars
    P-with
      with
    NP-binoculars
      binoculars
```
Head Rules

Then, propagate heads up the tree:

```
S
  NP-kids
    kids

VP-saw
  V-saw
    saw
  NP-birds
    birds

PP-binoculars
  P-with
    with
  NP-binoculars
    binoculars
```
Head Rules

Then, propagate heads up the tree:

\[
S \rightarrow \text{NP-kids} \leftarrow \text{VP-saw} \\
\text{NP-kids \rightarrow kids} \\
\text{VP-saw \rightarrow VP-saw \rightarrow V-saw \rightarrow saw} \\
\text{VP-saw \rightarrow NP-birds} \\
\text{PP-binoculars \rightarrow P-with \rightarrow NP-binoculars} \\
\text{NP-binoculars} \\
\text{VP} \rightarrow \text{saw} \\
\text{V} \rightarrow \text{saw} \\
\text{NP} \rightarrow \text{birds} \\
\text{PP} \rightarrow \text{binoculars} \\
\text{P-with} \rightarrow \text{with} \\
\text{NP} \rightarrow \text{binoculars}
\]
Lexicalized Constituency Parse (reading 1)

S-saw
  NP-kids
    kids
  VP-saw
    VP-saw
      V-saw
        saw
      NP-birds
        birds
    PP-binoculars
      P-with
        with
      NP-binoculars
Lexicalized Constituency Parse (reading 2)

S-saw
  NP-kids  VP-saw
    kids
  V-saw
    saw
    NP-birds
      birds
      PP-fish
        P-with
        NP-fish
          with
          fish
Constituency Tree $\rightarrow$ Dependency Tree

The lexical heads can then be used to collapse down to an unlabeled dependency tree.
Lexicalized Constituency Parse

S-saw
  NP-kids
    kids
  VP-saw
    V-saw
      NP-birds
        NP-birds
          birds
        PP-fish
          P-with
            NP-fish
              with
              fish

... remove the phrasal categories. ...

```
( (kids (saw (birds (saw (birds (fish))))))
```

... remove the (duplicated) terminals. ...
and collapse chains of duplicates.

\[
\text{saw} \quad \text{kids saw} \quad \text{saw birds} \quad \text{birds fish with fish}
\]
... and collapse chains of duplicates. ...
and collapse chains of duplicates.
and collapse chains of duplicates.
. . . and collapse chains of duplicates. . .

- saw
  - kids
    - saw
      - saw
        - birds
          - fish
            - with
... and collapse chains of duplicates. ...

- saw
  - kids
  - birds
    - fish
      - with
Practicalities of Lexicalized CFG
Constituency Parsing

• Leads to huge grammar blowup and very sparse data (bad!)
  – There are fancy techniques to address these issues. . . and they can work pretty well.
  – But: Do we really need phrase structures in the first place? Not always!

• Hence: Sometimes we want to parse directly to dependencies, as with transition-based or graph-based algorithms.
Summary

• While constituency parses give hierarchically nested phrases, dependency parses represent syntax with trees whose edges connect words in the sentence. (No abstract phrase categories like NP.) Edges often labeled with relations like subject.

• Head rules govern how a lexicalized constituency grammar can be extracted from a treebank, and how a constituency parse can be converted to a dependency parse.

• For English, it is often fastest and most convenient to parse directly to dependencies. Two main paradigms, graph-based and transition-based, with different kinds of models and search algorithms.