Dependency grammar and dependency parsing

Syntactic analysis (5LN455)

2016-12-05

Sara Stymne
Department of Linguistics and Philology

Based on slides from Marco Kuhlmann
Activities - dependency parsing

- 4 lectures (December)
- 1 literature seminar (January)
- 1 or 2 assignment (DL: January 12)
- Project (master 7.5; DL: January 12)
- Supervision on demand, by email or book a meeting
- Also: masters: literature review, DL Dec 18
Overview

- Arc-factored dependency parsing
  - Collins’ algorithm
  - Eisner’s algorithm
- Evaluation of dependency parsers
- Transition-based dependency parsing
  - The arc-standard algorithm
- Projectivity
- Reordering and oracles
Dependency grammar
Dependency grammar

• The term ‘dependency grammar’
does not refer to a specific grammar formalism.

• Rather, it refers to a specific way
to describe the syntactic structure of a sentence.
The notion of dependency

- The basic observation behind **constituency** is that groups of words may act as one unit.
  
  *Example:* noun phrase, prepositional phrase

- The basic observation behind **dependency** is that words have grammatical functions with respect to other words in the sentence.
  
  *Example:* subject, modifier
Phrase structure trees
Dependency grammar

Dependency trees

- In an arc $h \rightarrow d$, the word $h$ is called the head, and the word $d$ is called the dependent.
- The arcs form a rooted tree.
- Each arc has a label, $l$, and an arc can be described as $(h, d, l)$.
• In an arc $h \rightarrow d$, the word $h$ is called the head, and the word $d$ is called the dependent.

• The arcs form a rooted tree.

• Each arc has a label, $l$, and an arc can be described as $(h, d, l)$.
In an arc $h \rightarrow d$, the word $h$ is called the head, and the word $d$ is called the dependent.

The arcs form a rooted tree.

Each arc has a label, $l$, and an arc can be described as $(h, d, l)$
Heads in phrase structure grammar

• In phrase structure grammar, ideas from dependency grammar can be found in the notion of heads.

• Roughly speaking, the head of a phrase is the most important word of the phrase: the word that determines the phrase function.

  *Examples:* noun in a noun phrase, preposition in a prepositional phrase
Heads in phrase structure grammar

Dependency grammar

Heads in phrase structure grammar
The history of dependency grammar

• The notion of dependency can be found in some of the earliest formal grammars.

• Modern dependency grammar is attributed to Lucien Tesnière (1893–1954).

• Recent years have seen a revived interest in dependency-based description of natural language syntax.
Linguistic resources

• Descriptive dependency grammars exist for some natural languages.

• Dependency treebanks exist for a wide range of natural languages.

• These treebanks can be used to train accurate and efficient dependency parsers.

• We will not use grammars in the parsing algorithms we discuss in the course.
Projectivity

• An important characteristic of dependency trees is projectivity.

• A dependency tree is projective if:
  
  • For every arc in the tree, there is a directed path from the head of the arc to all words occurring between the head and the dependent (that is, the arc \((i, l, j)\) implies that \(i \rightarrow^* k\) for every \(k\) such that \(\min(i, j) < k < \max(i, j)\))
Projective and non-projective trees

Figure 1. Dependency tree for an English sentence with dummy root node.

Figure 2. Non-projective dependency tree for an English sentence.
Projectivity and dependency parsing

• Many dependency parsing algorithms can only handle projective trees

• Non-projective trees do occur in natural language
  • How often depends on the language (and treebank)
Projectivity in the course

• The algorithms we will discuss in detail during the lectures will only concern projective parsing

• Non-projective parsing:
  • Seminar 2: Pseudo-projective parsing
  • Lecture 10: Transition-based parsing + swap
  • Other variants mentioned briefly
  • You can read more about it in the course book!
Arc-factored dependency parsing
Just like phrase structure parsing, dependency parsing has to deal with ambiguity.
Ambiguity

Just like phrase structure parsing, dependency parsing has to deal with ambiguity.

I booked a flight from LA.
Disambiguation

• We need to disambiguate between alternative analyses.

• We develop mechanisms for scoring dependency trees, and disambiguate by choosing a dependency tree with the highest score.
Scoring models and parsing algorithms

Distinguish two aspects:

- **Scoring model:**
  How do we want to score dependency trees?

- **Parsing algorithm:**
  How do we compute a highest-scoring dependency tree under the given scoring model?
The arc-factored model

- Split the dependency tree $t$ into parts $p_1, ..., p_n$, score each of the parts individually, and combine the score into a simple sum.

\[ \text{score}(t) = \text{score}(p_1) + \ldots + \text{score}(p_n) \]

- The simplest scoring model is the **arc-factored model**, where the scored parts are the arcs of the tree.
To score an arc, we define **features** that are likely to be relevant in the context of parsing.

We represent an arc by its **feature vector**.
Arc-factored dependency parsing

Examples of features
Arc-factored dependency parsing

Examples of features

• ‘The head is a verb.’
Examples of features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
Examples of features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
- ‘The head is a verb
  and the dependent is a noun.’
Examples of features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
- ‘The head is a verb
  and the dependent is a noun.’
- ‘The head is a verb
  and the predecessor of the head is a pronoun.’
Examples of features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
- ‘The head is a verb
  and the dependent is a noun.’
- ‘The head is a verb
  and the predecessor of the head is a pronoun.’
- ‘The arc goes from left to right.’
Examples of features

• ‘The head is a verb.’
• ‘The dependent is a noun.’
• ‘The head is a verb and the dependent is a noun.’
• ‘The head is a verb and the predecessor of the head is a pronoun.’
• ‘The arc goes from left to right.’
• ‘The arc has length 2.’
Arc-factored dependency parsing

**Feature vectors**

Feature: 'The head is a verb.'

Feature: 'The dependent is a noun.'
Arc-factored dependency parsing

Feature vectors

- Feature: ‘The head is a verb.’
- Feature: ‘The dependent is a noun.’

Example:

booked → flight

flight → from LA

flight → a

booked → 1
Implementation of feature vectors

• We assign each feature a unique number.
• For each arc, we collect the numbers of those features that apply to that arc.
• The feature vector of the arc is the list of those numbers.

Example: [1, 2, 42, 313, 1977, 2008, 2010]
Feature weights

- Arc-factored dependency parsers require a training phase.
- During training, our goal is to assign, to each feature $f_i$, a feature weight $w_i$.
- Intuitively, the weight $w_i$ quantifies the effect of the feature $f_i$ on the likelihood of the arc.

How likely is it that we will see an arc with this feature in a useful dependency tree?
Feature weights

We define the score of an arc $h \rightarrow d$ as the weighted sum of all features of that arc:

$$\text{score}(h \rightarrow d) = f_1w_1 + \ldots + f_nw_n$$
Training using structured prediction

- Take a sentence $w$ and a gold-standard dependency tree $g$ for $w$.
- Compute the highest-scoring dependency tree under the current weights; call it $p$.
- Increase the weights of all features that are in $g$ but not in $p$.
- Decrease the weights of all features that are in $p$ but not in $g$. 
Training involves repeatedly parsing (treebank) sentences and refining the weights.

Hence, training presupposes an efficient parsing algorithm.
The arc-factored model is a first-order model, because scored subgraphs consist of a single arc.

An nth-order model scores subgraphs consisting of (at most) n arcs.

Second-order: siblings, grand-parents

Third-order: tri-siblings, grand-siblings

Higher-order models capture more linguistic structure and give higher parsing accuracy, but are less efficient.
Parsing algorithms

- Projective parsing
  - Inspired by the CKY algorithm
    - Collins’ algorithm
    - Eisner’s algorithm
- Non-projective parsing:
  - Minimum spanning tree (MST) algorithms
Arc-factored dependency parsing

Graph-based parsing

• Arc-factored parsing is an instance of graph-based dependency parsing
• Because it scores the dependency graph (tree)
• Graph-based models are often contrasted with transition-based models (Dec 12+14)
• There are also grammar-based methods, which we will not discuss
The term ‘arc-factored dependency parsing’ refers to dependency parsers that score a dependency tree by scoring its arcs. Arcs are scored by defining features and assigning weights to these features. The resulting parsers can be trained using structured prediction. More powerful scoring models exist.
Overview

- Arc-factored dependency parsing
  
  Collins’ algorithm
  
  Eisner’s algorithm

- Evaluation of dependency parsers

- Transition-based dependency parsing
  
  The arc-standard algorithm

- Projectivity

- Reordering and oracles
Collins’ algorithm
Collins’ algorithm

- Collin’s algorithm is a simple algorithm for computing the highest-scoring dependency tree under an arc-factored scoring model.

- It can be understood as an extension of the CKY algorithm to dependency parsing.

- Like the CKY algorithm, it can be characterized as a bottom-up algorithm based on dynamic programming.
Collins’ algorithm

Signatures, CKY

$[\text{min}, \text{max}, C]$
Collins’ algorithm

Signatures, Collins’

[min, max, root]
Collins’ algorithm

Initialization

I booked a flight from LA
Collins’ algorithm

Initialization

I booked a flight from LA

[0, 1, I] [1, 2, booked] [2, 3, a] [3, 4, flight] [4, 5, from LA]
Collins’ algorithm

## Adding a left-to-right arc

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>booked</th>
<th>a</th>
<th>flight</th>
<th>from LA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Collins’ algorithm

Adding a left-to-right arc

I booked a flight from LA
Adding a left-to-right arc

Collins’ algorithm

I booked a flight from LA

[3, 4, flight]  [4, 5, from LA]
Collins' algorithm

Adding a left-to-right arc

I booked a flight from LA
Collins’ algorithm

Adding a left-to-right arc
Adding a left-to-right arc
Collins’ algorithm

Adding a left-to-right arc
Collins’ algorithm

Adding a left-to-right arc
Adding a left-to-right arc

\[
\text{score}(t) = \text{score}(t_1) + \text{score}(t_2) + \text{score}(l \to r)
\]
Adding a left-to-right arc

for each \([\text{min}, \text{max}]\) with \(\text{max} - \text{min} > 1\) do

for each \(l\) from \(\text{min}\) to \(\text{max} - 2\) do

\[\text{double best} = \text{score}[\text{min}][\text{max}][l]\]

for each \(r\) from \(l + 1\) to \(\text{max} - 1\) do

\[\text{for each mid from } l + 1 \text{ to } r \text{ do}\]

\[t_1 = \text{score}[\text{min}][\text{mid}][l]\]

\[t_2 = \text{score}[\text{mid}][\text{max}][r]\]

\[\text{double current} = t_1 + t_2 + \text{score}(l \rightarrow r)\]

if \(\text{current} > \text{best}\) then

\[\text{best} = \text{current}\]

\[\text{score}[\text{min}][\text{max}][l] = \text{best}\]
Collins’ algorithm

Adding a right-to-left arc

I booked a flight from LA
Collins’ algorithm

Adding a right-to-left arc

```
I  booked  a  flight  from LA
0  1  2  3  4  5

[0, 1, I]  [1, 2, booked]
```

pmod
Collins’ algorithm

Adding a right-to-left arc

I booked a flight from LA

Collins’ algorithm

Adding a right-to-left arc
Collins’ algorithm

Adding a right-to-left arc

I booked a flight from LA

[0, 2, booked]
Collins’ algorithm

Adding a right-to-left arc
Collins’ algorithm

Adding a right-to-left arc
Collins’ algorithm

Adding a right-to-left arc

\[ l \quad \quad \quad \quad r \]

\[ t_1 \quad \quad \quad \quad t_2 \]

\[ min \quad mid \quad max \]
Collins’ algorithm

Adding a right-to-left arc
Collins’ algorithm

Adding a right-to-left arc

\[
\text{score}(t) = \text{score}(t_1) + \text{score}(t_2) + \text{score}(r \rightarrow l)
\]
Collins’ algorithm

Adding a right-to-left arc

\[
\text{for each } [\text{min}, \text{max}] \text{ with } \text{max} - \text{min} > 1 \text{ do}
\]

\[
\text{for each } r \text{ from } \text{min} + 1 \text{ to } \text{max} - 1 \text{ do}
\]

\[
\text{double } \text{best} = \text{score}[	ext{min}][\text{max}][r]
\]

\[
\text{for each } l \text{ from } \text{min} \text{ to } r - 1 \text{ do}
\]

\[
\text{for each } \text{mid} \text{ from } l + 1 \text{ to } r \text{ do}
\]

\[
\text{t}_1 = \text{score}[	ext{min}][\text{mid}][l]
\]

\[
\text{t}_2 = \text{score}[	ext{mid}][\text{max}][r]
\]

\[
\text{double } \text{current} = \text{t}_1 + \text{t}_2 + \text{score}(r \rightarrow l)
\]

\[
\text{if } \text{current} > \text{best} \text{ then}
\]

\[
\text{best} = \text{current}
\]

\[
\text{score}[	ext{min}][\text{max}][r] = \text{best}
\]
Collins’ algorithm

Finishing up

I booked a flight from LA
Finishing up

Collins’ algorithm

I booked a flight from LA

[2, 3, a] [3, 5, flight]
Finishing up

Collins’ algorithm

I booked a flight from LA

[2, 5, flight]
Finishing up

Collins’ algorithm

I booked a flight from LA

[0, 2, booked] [2, 5, flight]
Collins’ algorithm

Finishing up

I booked a flight from LA

Collins' algorithm
Collins’ algorithm

Complexity analysis

- Runtime?
- Space?

```java
for each [min, max] with max - min > 1 do
    for each r from min + 1 to max - 1 do
        double best = score[min][max][r]
        for each l from min to r - 1 do
            for each mid from l + 1 to r do
                t₁ = score[min][mid][l]
                t₂ = score[mid][max][r]
                double current = t₁ + t₂ + score(r → l)
                if current > best then
                    best = current
        score[min][max][r] = best
```
 Complexity analysis

- Runtime?
- Space?

for each \([\text{min}, \text{max}]\) with \(\text{max} - \text{min} > 1\) do

  for each \(r\) from \(\text{min} + 1\) to \(\text{max} - 1\) do

    double best = \(\text{score}[\text{min}][\text{max}][r]\)

    for each \(l\) from \(\text{min}\) to \(r - 1\) do

      for each mid from \(l + 1\) to \(r\) do

        \(t_1 = \text{score}[\text{min}][\text{mid}][l]\)

        \(t_2 = \text{score}[\text{mid}][\text{max}][r]\)

        double current = \(t_1 + t_2 + \text{score}(r \rightarrow l)\)

        if current > best then

          best = current

          \(\text{score}[\text{min}][\text{max}][r] = \text{best}\)
Collins’ algorithm

Complexity analysis

- Runtime?
- Space?

```java
for each [min, max] with max - min > 1 do
    for each r from min + 1 to max - 1 do
        double best = score[min][max][r]
        for each l from min to r - 1 do
            for each mid from l + 1 to r do
                t1 = score[min][mid][l]
                t2 = score[mid][max][r]
                double current = t1 + t2 + score(r \rightarrow l)
                if current > best then
                    best = current
                score[min][max][r] = best
```
Collins’ algorithm

Complexity analysis

- **Space requirement:**
  \[ O(|w|^3) \]

- **Runtime requirement:**
  \[ O(|w|^5) \]
Collins’ algorithm is a CKY-style algorithm for computing the highest-scoring dependency tree under an arc-factored scoring model.

- It runs in time $O(|w|^5)$.
- This may not be practical for long sentences.

- We have not discussed labels yet - we will do that next lecture.