Transition-based
dependency parsing

Syntactic analysis (5LN455)
2014-12-17

Sara Stymne
Department of Linguistics and Philology

Based on slides from Marco Kuhlmann
Overview

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

• Transition-based dependency parsing
  The arc-standard algorithm

• Evaluation of dependency parsers

• Projectivity
Transition-based dependency parsing
Transition-based dependency parsing

The parser starts in an initial configuration.

At each step, it asks a guide to choose between one of several transitions (actions) into new configurations.

Parsing stops if the parser reaches a terminal configuration.

The parser returns the dependency tree associated with the terminal configuration.
Transition-based dependency parsing

- Eisner’s algorithm runs in time $O(|w|^3)$. This may be too much if a lot of data is involved.
- **Idea:** Design a dumber but really fast algorithm and let the machine learning do the rest.
- Eisner’s algorithm searches over many different dependency trees at the same time.
- A transition-based dependency parser only builds *one* tree, in *one* left-to-right sweep over the input.
Transition-based dependency parsing

**Generic parsing algorithm**

Configuration $c = \text{parser.getInitialConfiguration}(\text{sentence})$

while $c$ is not a terminal configuration do

    Transition $t = \text{guide.getNextTransition}(c)$

    $c = c.\text{makeTransition}(t)$

return $c.\text{getGraph}()$
Variation

Transition-based dependency parsers differ with respect to the configurations and the transitions that they use.
The arc-standard algorithm
The arc-standard algorithm

- The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.
- It is very similar to shift–reduce parsing as it is known for context-free grammars.
- It is implemented in most practical transition-based dependency parsers, including MaltParser.
A configuration for a sentence $w = w_1 \ldots w_n$ consists of three components:

- a buffer containing words of $w$
- a stack containing words of $w$
- the dependency graph constructed so far
Configurations

• **Initial configuration:**
  • All words are in the buffer.
  • The stack is empty.
  • The dependency graph is empty.

• **Terminal configuration:**
  • The buffer is empty.
  • The stack contains a single word.
The arc-standard algorithm

Possible transitions

- **shift (sh):** push the next word in the buffer onto the stack
- **left-arc (la):** add an arc from the topmost word on the stack, $s_1$, to the second-topmost word, $s_2$, and pop $s_2$
- **right-arc (ra):** add an arc from the second-topmost word on the stack, $s_2$, to the topmost word, $s_1$, and pop $s_1$
The arc-standard algorithm

Configurations and transitions

- **Initial configuration:** $([], [0,\ldots,n], [])$
- **Terminal configuration:** $([0], [], A)$
- **shift (sh):**
  
  $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$

- **left-arc (la):**
  
  $([\sigma|i|j], B, A) \Rightarrow ([\sigma|j], B, A \cup \{j, l, i\})$  \hspace{1cm} only if $i \neq 0$

- **right-arc (ra):**
  
  $([\sigma|i|j], B, A) \Rightarrow ([\sigma|i], B, A \cup \{i, l, j\})$
Example run

The arc-standard algorithm

Stack

Buffer

I booked a flight from LA

I booked a flight from LA
The arc-standard algorithm

Example run

Stack

Buffer

I booked a flight from LA

sh
The arc-standard algorithm

Example run

Stack

Buffer

I

booked

a

flight

from LA

I booked a flight from LA
The arc-standard algorithm

Example run

Stack

Buffer

Stack: I
Buffer: booked a flight from LA

I booked a flight from LA
Example run

Stack

Buffer

I booked a flight from LA
The arc-standard algorithm

Example run

Stack

Buffer

I booked a flight from LA

I booked a flight from LA

la-subj
The arc-standard algorithm

Example run

Stack

Buffer

subj

I booked a flight from LA
Example run

The arc-standard algorithm

Stack

Buffer

booked

a

flight

from LA

subj

I booked a flight from LA

sh
The arc-standard algorithm

Example run

Stack
- booked
- a

Buffer
- flight
- from LA

subj
I booked a flight from LA
The arc-standard algorithm

Example run

Stack

Buffer

I booked a flight from LA

subj

sh
The arc-standard algorithm

Example run

Stack

Buffer

I booked a flight from LA

subj

from LA
The arc-standard algorithm

Example run

Stack

- booked
- a
- flight

Buffer

- from LA

subj

I booked a flight from LA

la-det
The arc-standard algorithm

Example run

Stack

Buffer

I booked a flight from LA
Example run

The arc-standard algorithm

Stack

Buffer

subj

l

booked

flight

from LA

sh
Example run

Stack

- booked
- flight
- from LA

Buffer

subj

I booked

det

a flight from LA

The arc-standard algorithm
The arc-standard algorithm

Example run

Stack
- booked
- flight
- from LA

Buffer

subj: I booked a flight from LA

det: a flight from LA
The arc-standard algorithm

Example run

I booked a flight from LA.
Example run

The arc-standard algorithm
Example run

The arc-standard algorithm

Stack

booked

Buffer

I booked a flight from LA

subj
dobj
det
pmod
The arc-standard algorithm

**Example run**

The arc-standard algorithm

Stack

Buffer

I booked a flight from LA

done!
Complexity and optimality

• Time complexity is linear, $O(n)$, since we only have to treat each word once
• This can be achieved since the algorithm is greedy, and only builds one tree, in contrast to Eisner’s algorithm, where all trees are explored
• There is no guarantee that we will even find the best tree given the model, with the arc-standard model
• There is a risk of error propagation
• An advantage is that we can use very informative features, for the ML algorithm
Training a guide
Guides

• We need a guide that tells us what the next transition should be.

• The task of the guide can be understood as **classification**: Predict the next transition (class), given the current configuration.
Training a guide

• We let the parser run on gold-standard trees.

• Every time there is a choice to make, we simply look into the tree and do ‘the right thing’™.

• We collect all (configuration, transition) pairs and train a classifier on them.

• When parsing unseen sentences, we use the trained classifier as a guide.
Training a guide

- The number of (configuration, transition) pairs is far too large.
- We define a set of features of configurations that we consider to be relevant for the task of predicting the next transition.

Example: word forms of the topmost two words on the stack and the next two words in the buffer

- We can then describe every configuration in terms of a feature vector.
Training a guide

Transition-based dependency parsing

configurations in which we want to do la

configurations in which we want to do ra

score for feature 1

score for feature 2
Training a guide

Transition-based dependency parsing

Classification function learned by the classifier
Training a guide

• In practical systems, we have thousands of features and hundreds of transitions.

• There are several machine-learning paradigms that can be used to train a guide for such a task.

  Examples: perceptron, decision trees, support-vector machines, memory-based learning
### Example features

<table>
<thead>
<tr>
<th>Address</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack[0]</td>
<td>FORM</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Stack[1]</td>
<td></td>
</tr>
<tr>
<td>Ldep(Stack[0])</td>
<td></td>
</tr>
<tr>
<td>Rdep(Stack[0])</td>
<td></td>
</tr>
<tr>
<td>Buffer[0]</td>
<td>X</td>
</tr>
<tr>
<td>Buffer[1]</td>
<td></td>
</tr>
<tr>
<td>Ldep(Buffer[0])</td>
<td></td>
</tr>
<tr>
<td>Ldep(Buffer[0])</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- Combinations of addresses and attributes (e.g. those marked in the table)
- Other features, such as distances, number of children, ...
Alternative transition models
Alternatives

• The arc-standard model as I presented it is just one example of a transition model
  • In the book you can see another version of the arc-standard model, where arcs are added between the topmost word on the stack and the topmost word on the buffer
  • There are many other alternatives
Alternative transition models

Arc-eager model

- Contains four transitions:
  - Shift
  - Reduce
  - Left-arc
  - Right-arc

- Advantage: not strictly bottom-up, can create arcs earlier than in the arc-standard model
Non-projective transition model

- Allows non-projective parsing by adding a swap transition
- Contains four transitions:
  - Shift
  - Swap
  - Left-arc
  - Right-arc
- Runtime is $O(n^2)$ in the worst case (but usually less in practice)
Alternative transition models

Transition models in Maltparser

• Nivre family
  • Arcs created between stack and buffer
    • arc-eager model
    • arc-standard (variant from course book)

• Stack family
  • Arcs between two topmost words on stack
    • arc-standard model (from slides)
    • models with swap transition

• Other families available as well
Evaluation of dependency parsing
Evaluation of dependency parsers

- labelled attachment score (LAS): percentage of correct arcs, relative to the gold standard
- labelled exact match (LEM): percentage of correct dependency trees, relative to the gold standard
- unlabelled attachment score/exact match (UAS/UEM): the same, but ignoring arc labels
Word- vs sentence-level AS

• Example: 2 sentence corpus
  sentence 1: 9/10 arcs correct
  sentence 2: 15/45 arcs correct

• Word-level (micro-average):
  \[
  \frac{(9+15)}{(10+45)} = \frac{24}{55} = 0.436
  \]

• Sentence-level (macro-average):
  \[
  \frac{(9/10+15/45)}{2} = \frac{(0.9+0.33)}{2} = 0.617
  \]

• Word-level evaluation is normally used
Projectivity
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

In order to be a well-formed dependency tree, the directed graph must also satisfy the following conditions:

1. Single-Head: Every node has at most one incoming arc (that is, the arc \((i,j)\) implies that there is an arc from \(i\) to \(j\)).
2. Connected: 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 \((i,j)\) contains all possible arcs \((k,l)\) such that \(i < k < j\)).
3. Root: The dummy root node 0 does not have any incoming arc (that is, there is no arc \((0,j)\) for every \(j\)).
4. Projective: In a complete graph there is a path between any two nodes.

Another difference compared to phrase structure parsing is that there are no part-of-speech nodes. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

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 \((i,j)\) contains all possible arcs \((k,l)\) such that \(i < k < j\)).

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.

Projectivity is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing. Broadly speaking, dependency-based grammar theories and annotation schemes normally do not assume that all dependency trees are projective, because some linguistic phenomena involving discontinuous structures can only be adequately represented using non-projective trees. By contrast, many dependency-based syntactic parsers assume that dependency relations are projective. This information can therefore be exploited in the feature representations used for projective dependency parsing, as the nodes are given by the input and only the arcs have to be inferred. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.

Projectivity is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing. Broadly speaking, dependency-based grammar theories and annotation schemes normally do not assume that all dependency trees are projective, because some linguistic phenomena involving discontinuous structures can only be adequately represented using non-projective trees. By contrast, many dependency-based syntactic parsers assume that dependency relations are projective. This information can therefore be exploited in the feature representations used for projective dependency parsing, as the nodes are given by the input and only the arcs have to be inferred. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.

Projectivity is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing. Broadly speaking, dependency-based grammar theories and annotation schemes normally do not assume that all dependency trees are projective, because some linguistic phenomena involving discontinuous structures can only be adequately represented using non-projective trees. By contrast, many dependency-based syntactic parsers assume that dependency relations are projective. This information can therefore be exploited in the feature representations used for projective dependency parsing, as the nodes are given by the input and only the arcs have to be inferred. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.

Projectivity is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing. Broadly speaking, dependency-based grammar theories and annotation schemes normally do not assume that all dependency trees are projective, because some linguistic phenomena involving discontinuous structures can only be adequately represented using non-projective trees. By contrast, many dependency-based syntactic parsers assume that dependency relations are projective. This information can therefore be exploited in the feature representations used for projective dependency parsing, as the nodes are given by the input and only the arcs have to be inferred. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.

Projectivity is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing. Broadly speaking, dependency-based grammar theories and annotation schemes normally do not assume that all dependency trees are projective, because some linguistic phenomena involving discontinuous structures can only be adequately represented using non-projective trees. By contrast, many dependency-based syntactic parsers assume that dependency relations are projective. This information can therefore be exploited in the feature representations used for projective dependency parsing, as the nodes are given by the input and only the arcs have to be inferred. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.

Projectivity is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing. Broadly speaking, dependency-based grammar theories and annotation schemes normally do not assume that all dependency trees are projective, because some linguistic phenomena involving discontinuous structures can only be adequately represented using non-projective trees. By contrast, many dependency-based syntactic parsers assume that dependency relations are projective. This information can therefore be exploited in the feature representations used for projective dependency parsing, as the nodes are given by the input and only the arcs have to be inferred. However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence nodes). However, most dependency parsers instead assume that part-of-speech tags are part of the input, so that the input sentence is a set of nodes, one for each position of a word \(x\) in the sentence, annotated with their tags in the syntactic representations (because there are no pre-terminal nodes, only terminal nodes). This makes the parsing problem more constrained than in the case of phrase structure parsing, as the nodes are given by the input and only the arcs have to be inferred.

In graph-theoretic terms, this is equivalent to finding a spanning tree, as the nodes are given by the input and only the arcs have to be inferred.
Projectivity and dependency parsing

- Many dependency parsing algorithms can only handle projective trees
  - Including all algorithms we have discussed in detail
- Non-projective trees do occur in natural language
  - How often depends on language (and treebank)
Non-projective dependency parsing

- Variants of transition-based parsing
  - Using a swap-transition
  - Using more than one stack
  - Pseudo-projective parsing (seminar 2)
- Graph-based parsing
  - Minimum spanning tree algorithms
- ...
In transition-based dependency parsing, one does not score graphs but computations, sequences of (configuration, transition) pairs.

In its simplest form, transition-based dependency parsing uses classification.

One specific instance of transition-based dependency parsing is the arc-standard algorithm.
The end of the course

- Seminar 2, Pseudo-projective parsing
  - Easier and shorter article than last seminar, some more general questions
- Assignment 3: Disambiguation in arc-factored and transition-based parsing
- Assignment 4: Try and evaluate MaltParser
- Course evaluation in the student portal