Natural Language Processing

Syntactic parsing

Yulia Tsvetkov

yuliats@cs.washington.edu
Constituent (phrase-structure) representation

```
S
   / \
  NP   VP
     /   /\n    Pro Verb NP
       /     /\n      I  prefer Det Nom
          /     /\n         the Nom PP
              /     /\n             Nom Noun P NP
                  /     /\n                 Noun flight through Pro
                                  /     /\n                                 morning Denver
```
I prefer the morning flight through Denver
A dependency structure can be defined as a directed graph $G$, consisting of

- a set $V$ of nodes – vertices, words, punctuation, morphemes
- a set $A$ of arcs – directed edges,
- a linear precedence order $<$ on $V$ (word order).

Labeled graphs

- nodes in $V$ are labeled with word forms (and annotation).
- arcs in $A$ are labeled with dependency types
- $L = \{l_1, \ldots, l_{|L|}\}$ is the set of permissible arc labels;
- Every arc in $A$ is a triple $(i,j,k)$, representing a dependency from $w_i$ to $w_j$ with label $l_k$. 

I prefer the morning flight through Denver
Dependency vs Constituency

- Dependency structures explicitly represent
  - head-dependent relations (directed arcs),
  - functional categories (arc labels)
  - possibly some structural categories (parts of speech)

- Phrase (aka constituent) structures explicitly represent
  - phrases (nonterminal nodes),
  - structural categories (nonterminal labels)
Dependency vs Constituency trees

```
prefer
I

flight

the morning Denver

through

S

NP

Pro

Verb

I prefer

Det

Nom

Through

Nom

Noun

flight

P

NP

morning Denver
```
I prefer the morning flight through Denver

Я предпочитаю утренний перелет через Денвер
I prefer the morning flight through Denver

Я предпочитаю утренний перелет через Денвер
Я предпочитаю через Денвер утренний перелет
Утренний перелет я предпочитаю через Денвер
Перелет утренний я предпочитаю через Денвер
Через Денвер я предпочитаю утренний перелет
Я через Денвер предпочитаю утренний перелет
Dependency relations

- Head
- Governor
- Parent

- Modifier
- Dependent
- Child

- Label
- Relation
- Type

eat$_2$, fish$_4$, obj
Types of relationships

- The clausal relations NSUBJ and DOBJ identify the arguments: the subject and direct object of the predicate *cancel*.

- The NMOD, DET, and CASE relations denote modifiers of the nouns *flights* and *Houston*.
## Grammatical functions

<table>
<thead>
<tr>
<th>Clausal Argument Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSUBJ</td>
<td>Nominal subject</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Direct object</td>
</tr>
<tr>
<td>IOBJ</td>
<td>Indirect object</td>
</tr>
<tr>
<td>CCOMP</td>
<td>Clausal complement</td>
</tr>
<tr>
<td>XCOMP</td>
<td>Open clausal complement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nominal Modifier Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMOD</td>
<td>Nominal modifier</td>
</tr>
<tr>
<td>AMOD</td>
<td>Adjectival modifier</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Numeric modifier</td>
</tr>
<tr>
<td>APPOS</td>
<td>Appositional modifier</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>CASE</td>
<td>Prepositions, postpositions and other case markers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Notable Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONJ</td>
<td>Conjunct</td>
</tr>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
</tbody>
</table>

*Figure 13.2* Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)
Dependency Constraints

- Syntactic structure is complete (**connectedness**)
  - connectedness can be enforced by adding a special root node
- Syntactic structure is hierarchical (**acyclicity**)
  - there is a unique pass from the root to each vertex
- Every word has at most one syntactic head (**single-head constraint**)
  - except root that does not have incoming arcs

This makes the dependencies a tree
Projectivity

- **Projective parse**
  - arcs don’t cross each other
  - mostly true for English
- **Non-projective structures are needed to account for**
  - long-distance dependencies
  - flexible word order

JetBlue canceled our flight this morning which was already late
Dependency grammars do not normally assume that all dependency-trees are projective, because some linguistic phenomena can only be achieved using non-projective trees.

But a lot of parsers assume that the output trees are projective.

Reasons
- conversion from constituency to dependency
- the most widely used families of parsing algorithms impose projectivity
The idea is to use the inorder traversal of the tree: \(<\text{left-child}, \text{root}, \text{right-child}>\)

- This is well defined for binary trees. We need to extend it to \(n\)-ary trees.
- If we have a projective tree, the inorder traversal will give us the original linear order.
Non-Projective Statistics

Arabic: 11.2 %
Bulgarian: 5.4 %
Chinese: 0.0 %
Czech: 23.2 %
Danish: 15.6 %
Dutch: 36.4 %
German: 27.8 %
Japanese: 5.3 %
Polish: 18.9 %
Slovene: 22.2 %
Spanish: 1.7 %
Swedish: 9.8 %
Turkish: 11.6 %
English: 0.0% (SD: 0.1%)

Percentage of non-projective trees for some treebanks of the CoNLL-X Shared Task and English.
Dependency Treebanks

▪ the major English dependency treebanks converted from the WSJ sections of the PTB (Marcus et al., 1993)
▪ OntoNotes project (Hovy et al. 2006, Weischedel et al. 2011) adds conversational telephone speech, weblogs, usenet newsgroups, broadcast, and talk shows in English, Chinese and Arabic
▪ annotated dependency treebanks created for morphologically rich languages such as Czech, Hindi and Finnish, eg Prague Dependency Treebank (Bejcek et al., 2013)
▪ http://universaldependencies.org/
  ▪ 150 treebanks, 90 languages
Xia and Palmer (2001)

- mark the head child of each node in a phrase structure, using the appropriate head rules
- make the head of each non-head child depend on the head of the head-child
The parsing problem for a dependency parser is to find the optimal dependency tree $y$ given an input sentence $x$.

This amounts to assigning a syntactic head $i$ and a label $l$ to every node $j$ corresponding to a word $x_j$ in such a way that the resulting graph is a tree rooted at the node 0.
This is equivalent to finding a spanning tree in the complete graph containing all possible arcs.
Parsing algorithms

- **Transition based**
  - greedy choice of local transitions guided by a good classifier
  - deterministic
  - MaltParser (Nivre et al. 2008)

- **Graph based**
  - Minimum Spanning Tree for a sentence
  - McDonald et al.’s (2005) MSTParser
  - Martins et al.’s (2009) Turbo Parser
Transition Based Parsing

- greedy discriminative dependency parser
- motivated by a stack-based approach called **shift-reduce parsing** originally developed for analyzing programming languages (Aho & Ullman, 1972).
- Nivre 2003

```
I prefer the morning flight through Denver
```
Figure 13.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.
**Configuration**

**Buffer**: unprocessed words

**Stack**: partially processed words

**Oracle**: a classifier
Operations

**Buffer**: unprocessed words

**Stack**: partially processed words

**Oracle**: a classifier

At each step choose:
- Shift
Operations

Buffer: unprocessed words

Stack: partially processed words

Oracle: a classifier

At each step choose:
- Shift
- Reduce left
Operations

Buffer: unprocessed words

Stack: partially processed words

Oracle: a classifier

At each step choose:
- Shift
- LeftArc or Reduce left
- RightArc or Reduce right
Shift-Reduce Parsing

Configuration:
- Stack, Buffer, Oracle, Set of dependency relations

Operations by a classifier at each step:
- **Shift**
  - remove w1 from the buffer, add it to the top of the stack as s1
- **LeftArc or Reduce left**
  - assert a head-dependent relation between s1 and s2
  - remove s2 from the stack
- **RightArc or Reduce right**
  - assert a head-dependent relation between s2 and s1
  - remove s1 from the stack
## Shift-Reduce Parsing

**Book me the morning flight**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
</table>
Shift-Reduce Parsing

```
Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

![Diagram of a parse tree with a sentence "Book me the morning flight" and a table showing the stack and word list at each step.]
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

```
[Root, Book, me, the, morning, flight]
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

![Diagram of word structure]

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

#### Diagram

```
  root  \\
  |    \\
  |    \\
  obj  \\
  |    \\
  |    \\
  dobj \\
  |    \\
  |    \\
  det  \\
  |    \\
  |    \\
  nmod \\
```

Book me the morning flight

#### Table

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Shift-Reduce Parsing

![Diagram of shift-reduce parsing](image)

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
# Shift-Reduce Parsing

## Diagram

```
  ~
 /     
root   obj
   
~~
/     
root iobj
   
/     
root dobj det nmod
   
Book me the morning flight
```

## Table

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

**Diagram:**
```
  ^  
 root  ~  ~
    /   \
  iobj
    \
     
  dobj
    \
     
 det
  \
     
  nmod
```

**Sentence:**
Book me the morning flight

**Table:**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
</tbody>
</table>
## Shift-Reduce Parsing

![Diagram of Shift-Reduce Parsing]

### Word List and Actions

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

**Diagram:**
- **Root:** Book
- **Direct Object:** me
- **Indirect Object:** the
- **Determiner:** morning
- **Modifier:** flight

**Word List:**
- [book, me, the, morning, flight]
- [me, the, morning, flight]
- [the, morning, flight]
- [the, morning, flight]
- [morning, flight]
- [flight]
- [flight]

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td>(book → me)</td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book → flight)</td>
</tr>
</tbody>
</table>
## Shift-Reduce Parsing

**Diagram:**

```
root      iobj
   
  dobj  det  nmod
```

**Example:**

*Book me the morning flight*

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book → flight)</td>
</tr>
</tbody>
</table>
# Shift-Reduce Parsing

![Diagram](image)

## Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book → flight)</td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

Configuration:
- Stack, Buffer, Oracle, Set of dependency relations

Operations by a classifier at each step:
- **Shift**
  - remove \( w_1 \) from the buffer, add it to the top of the stack as \( s_1 \)
- **LeftArc or Reduce left**
  - assert a head-dependent relation between \( s_1 \) and \( s_2 \)
  - remove \( s_2 \) from the stack
- **RightArc or Reduce right**
  - assert a head-dependent relation between \( s_2 \) and \( s_1 \)
  - remove \( s_1 \) from the stack

Complexity?

Oracle decisions can correspond to unlabeled or labeled arcs
Training an Oracle

- Oracle is a supervised classifier that learns a function from the configuration to the next operation
- How to extract the training set?
Training an Oracle

- How to extract the training set?
  - if LeftArc → LeftArc
  - if RightArc
    - if s1 dependents have been processed → RightArc
  - else → Shift
Training an Oracle

- How to extract the training set?
  - if LeftArc → LeftArc
  - if RightArc
    - if s1 dependents have been processed → RightArc
  - else → Shift

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Predicted Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, the, flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[the, flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>2</td>
<td>[root, book, the]</td>
<td>[flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>3</td>
<td>[root, book, the, flight]</td>
<td>[through, houston]</td>
<td>LEFTARC</td>
</tr>
<tr>
<td>4</td>
<td>[root, book, flight]</td>
<td>[through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>5</td>
<td>[root, book, flight, through]</td>
<td>[houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>6</td>
<td>[root, book, flight, through, houston]</td>
<td>[]</td>
<td>LEFTARC</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, flight, houston]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>9</td>
<td>[root, book]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
</tbody>
</table>
Training an Oracle

- Oracle is a supervised classifier that learns a function from the configuration to the next operation.

- How to extract the training set?
  - if LeftArc → LeftArc
  - if RightArc
    - if s1 dependents have been processed → RightArc
    - else → Shift

- What features to use?
- POS, word-forms, lemmas on the stack/buffer
- morphological features for some languages
- previous relations
- conjunction features (e.g. Zhang&Clark’08; Huang&Sagae’10; Zhang&Nivre’11)

<table>
<thead>
<tr>
<th>Source</th>
<th>Feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>One word</td>
<td>(s_1.w) (s_1.t) (s_1.wt)</td>
</tr>
<tr>
<td></td>
<td>(s_2.w) (s_2.t) (s_2.wt)</td>
</tr>
<tr>
<td></td>
<td>(b_1.w) (b_1.t) (b_0.wt)</td>
</tr>
<tr>
<td>Two word</td>
<td>(s_1.w \circ s_2.w) (s_1.t \circ s_2.t) (s_1.t \circ b_1.w)</td>
</tr>
<tr>
<td></td>
<td>(s_1.t \circ s_2.wt) (s_1.w \circ s_2.w \circ s_2.t) (s_1.w \circ s_1.t \circ s_2.t)</td>
</tr>
<tr>
<td></td>
<td>(s_1.w \circ s_1.t \circ s_2.t) (s_1.w \circ s_1.t)</td>
</tr>
</tbody>
</table>
Learning

- Before 2014: SVMs,
- After 2014: Neural Nets
Chen & Manning 2014

Stack

ROOT has_VBZ good JJ

nsubj

He_PRP

Buffer

control NN ...

binary, sparse
dim = 10^6 ~ 10^7

0 0 0 1 0 0 1 0 ... 0 0 1 0

Indicator
features

s_2.w = has \land s_2.t = VBZ

s_1.w = good \land s_1.t = JJ \land b_1.w = control

lc(s_2).t = PRP \land s_2.t = VBZ \land s_1.t = JJ

lc(s_2).w = He \land lc(s_2).l = nsubj \land s_2.w = has
Softmax probabilities

Output layer $y$
$y = \text{softmax}(Uh + b_2)$

cross-entropy error will be back-propagated to the embeddings.

Hidden layer $h$
$h = \text{ReLU}(Wx + b_1)$

Input layer $x$
lookup + concat

Stack
- ROOT
- has_VBZ
- good_JJ

Buffer
- control_NN
- ...

He_PRP
nssubj
**Features**

- `s1, s2, s3, b1, b2, b3`
- leftmost/rightmost children of `s1` and `s2`
- leftmost/rightmost grandchildren of `s1` and `s2`
- POS tags for the above
- arc labels for children/grandchildren
Evaluation of Dependency Parsers

$$\frac{\text{#correct dependencies}}{\text{#of dependencies}}$$

- LAS - labeled attachment score
- UAS - unlabeled attachment score
<table>
<thead>
<tr>
<th>Parser</th>
<th>UAS</th>
<th>LAS</th>
<th>sent. / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaltParser</td>
<td>89.8</td>
<td>87.2</td>
<td>469</td>
</tr>
<tr>
<td>MSTParser</td>
<td>91.4</td>
<td>88.1</td>
<td>10</td>
</tr>
<tr>
<td>TurboParser</td>
<td>92.3*</td>
<td>89.6*</td>
<td>8</td>
</tr>
<tr>
<td>C &amp; M 2014</td>
<td>92.0</td>
<td>89.7</td>
<td>654</td>
</tr>
<tr>
<td>Method</td>
<td>UAS</td>
<td>LAS (PTB WSJ SD 3.3)</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>Chen &amp; Manning 2014</td>
<td>92.0</td>
<td>89.7</td>
<td></td>
</tr>
<tr>
<td>Weiss et al. 2015</td>
<td>93.99</td>
<td>92.05</td>
<td></td>
</tr>
<tr>
<td>Andor et al. 2016</td>
<td>94.61</td>
<td>92.79</td>
<td></td>
</tr>
</tbody>
</table>
Transition-Based Dependency Parsing with Stack Long Short-Term Memory

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith

Marianas Labs, NLP Group, Pompeu Fabra University, Carnegie Mellon University

chris@marianaslabs.com, miguel.ballesteros@upf.edu, wangling@cs.upf.edu, austin.matthews@cmu.edu, noahsmith@cmu.edu

(i) 

(ii) 

(iii) 

REDUCE-LEFT(amod)

SHIFT
Arc-Eager

- LEFTARC: Assert a head-dependent relation between s1 and b1; pop the stack.
- RIGHTARC: Assert a head-dependent relation between s1 and b1; shift b1 to be s1.
- SHIFT: Remove b1 and push it to be s1.
- REDUCE: Pop the stack.

Diagram:

```
       root
      /   \
   dobj  nmod
     /     /   \
    det   case
```

Sentence: Book the flight through Houston
### Arc-Eager

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, the, flight, through, houston]</td>
<td></td>
<td>(root → book)</td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[the, flight, through, houston]</td>
<td>SHIFT</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>2</td>
<td>[root, book, the]</td>
<td>[flight, through, houston]</td>
<td>LEFTARC</td>
<td>(book → flight)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[flight, through, houston]</td>
<td>RIGHTARC</td>
<td>(through ← houston)</td>
</tr>
<tr>
<td>4</td>
<td>[root, book, flight]</td>
<td>[through, houston]</td>
<td>SHIFT</td>
<td>(flight → houston)</td>
</tr>
<tr>
<td>5</td>
<td>[root, book, flight, through]</td>
<td>[houston]</td>
<td>LEFTARC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, flight]</td>
<td>[houston]</td>
<td>RIGHTARC</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>[root, book, flight, houston]</td>
<td>[]</td>
<td>REDUCE</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>REDUCE</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>[root, book]</td>
<td>[]</td>
<td>REDUCE</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>REDUCE</td>
<td>Done</td>
</tr>
</tbody>
</table>
function DEPENDENCYBEAMPARSE(words, width) returns dependency tree

state ← {[root], [words], [], 0.0} ; initial configuration
agenda ← ⟨state⟩; initial agenda

while agenda contains non-final states

newagenda ← ⟨⟩

for each state ∈ agenda do

for all \{t | t ∈ VALIDOPERATORS(state)\} do

child ← APPLY(t, state)

newagenda ← ADDTOBEAM(child, newagenda, width)

agenda ← newagenda

return BESTOF(agenda)

function ADDTOBEAM(state, agenda, width) returns updated agenda

if LENGTH(agenda) < width then

agenda ← INSERT(state, agenda)

else if SCORE(state) > SCORE(WORSTOF(agenda))

agenda ← REMOVE(WORSTOF(agenda))

agenda ← INSERT(state, agenda)

return agenda
Parsing algorithms

- **Transition based**
  - greedy choice of local transitions guided by a good classifier
  - deterministic
  - MaltParser (Nivre et al. 2008), Stack LSTM (Dyer et al. 2015)

- **Graph based**
  - Minimum Spanning Tree for a sentence
  - non-projective
  - globally optimized
  - McDonald et al.’s (2005) MSTParser
  - Martins et al.’s (2009) Turbo Parser
Graph-Based Parsing Algorithms

- Start with a fully-connected directed graph
- Find a Minimum Spanning Tree
  - Chu and Liu (1965) and Edmonds (1967) algorithm

edge-factored approaches
Chu-Liu Edmonds algorithm

function MAXSPANNINGTREE(G=(V,E), root, score) returns spanning tree

F ← []
T' ← []
score' ← []
for each v ∈ V do
    bestInEdge ← argmax_{e=(u,v) ∈ E} score[e]
    F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
        score'[e] ← score[e] − score[bestInEdge]
if T=(V,F) is a spanning tree then return it
else
    C ← a cycle in F
    G' ← CONTRACT(G, C)
    T' ← MAXSPANNINGTREE(G', root, score')
    T ← EXPAND(T', C)
return T

function CONTRACT(G, C) returns contracted graph

function EXPAND(T, C) returns expanded graph
Chu-Liu Edmonds algorithm

- Select best incoming edge for each node
Chu-Liu Edmonds algorithm

- Subtract its score from all incoming edges
Chu-Liu Edmonds algorithm

- Contract nodes if there are cycles
Chu-Liu Edmonds algorithm

- Recursively compute MST
Chu-Liu Edmonds algorithm

- Expand contracted nodes
Scores

\[ score(S, e) = w \cdot f \]

- Wordforms, lemmas, and parts of speech of the headword and its dependent.
- Corresponding features derived from the contexts before, after and between the words.
- Word embeddings.
- The dependency relation itself.
- The direction of the relation (to the right or left).
- The distance from the head to the dependent.
Summary

- **Transition-based**
  - + Fast
  - + Rich features of context
  - - Greedy decoding

- **Graph-based**
  - + Exact or close to exact decoding
  - - Weaker features

Well-engineered versions of the approaches achieve comparable accuracy (on English), but make different errors

→ combining the strategies results in a substantial boost in performance