Graph Based Dependency Parsing

Wei Qiu

December 15, 2011
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
What is dependency parsing?

Input: a sentence
John saw Mary yesterday.

Output: a dependency tree

John -> saw -> Mary -> yesterday
Dependency tree

Directed tree with root

The arcs indicate certain grammatical relations between words

Properties

- **Single-headness**: Each word depends on exactly one parent.
- **Connectivity**
- **acyclic**
Projective dependency tree

- The edges above the words don't cross.
- Word and its descendents form a substring.
Projective dependency tree

- The edges above the words don’t cross.
- Word and its descendents form a substring.
Projective dependency tree

- The edges above the words don't cross.
- Word and its descendents form a substring.
Projective dependency tree

- The edges above the words don't cross.
- Word and its descendents form a substring.
Non-projective dependency tree

John saw Mary yesterday who was a young lady.
Non-projective dependency tree

John saw Mary yesterday who was a young lady.
Non-projective dependency tree

John saw Mary yesterday who was a young lady.
Non-projective dependency tree

John saw Mary yesterday who was a young lady.
Data-driven parsing framework

Training data \{sentence, tree\} pairs

Learning algorithm

Parser

Parsing model

Parsing algorithm

X: sentence

Y: dependency tree

language independent
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
What is graph based dependency parsing?

Graph-based models: Define a space of candidate dependency trees of input sentence.

- **Learning**: induce a model for scoring a candidate tree
- **Parsing**: find a tree with the highest score given the model
What is graph based dependency parsing?

Candidate trees for “John saw mary yesterday”
What is graph based dependency parsing?

Candidate trees for “John saw mary yesterday”
Arc-factored model

$X$: an input sentence
$Y$: a candidate dependency tree
$x_i \rightarrow x_j$: a dependency edge from word $i$ to word $j$
$\Phi(X)$: the set of possible dependent trees over $X$

$$
Y^* = \arg \max_{Y \in \Phi(X)} \text{score}(Y|X)
= \arg \max_{Y \in \Phi(X)} \sum_{(x_i \rightarrow x_j) \in Y} \text{score}(x_i \rightarrow x_j)
$$
Arc-factored model

\[
Y^* = \arg \max_{Y \in \Phi(X)} \text{score}(Y \mid X)
\]

\[
= \arg \max_{Y \in \Phi(X)} \sum_{(x_i \rightarrow x_j) \in Y} \text{score}(x_i \rightarrow x_j)
\]

\text{score}(x_i \rightarrow x_j) \text{ can be either probability or not.}

Mcdonald2005:

\[
\text{score}(x_i \rightarrow x_j) = \vec{w} \cdot \vec{f}(x_i \rightarrow x_j)
\]
Arc-factored model

Graph Based Dependency Parsing

John

saw

Mary

Root

yesterday
Arc-factored model

John → saw
Root → John
yesterday → saw
Mary → saw
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
Naive CYK-like parsing

Ideas

- Legal subtree spans on contiguous string.
- Subtree can be built from smaller subtrees step by step. In each step, always combine only 2 subtrees! (Exactly CYK!)
Naive CYK-like parsing: example

Example

John saw mary
Naive CYK-like parsing: example

Example

John saw mary

\[
\text{John} \quad \text{saw} \quad \text{saw} \quad \text{Mary} \\
\text{John} \quad \text{saw} \quad \text{saw} \quad \text{Mary} \\
(\text{John}) \quad (\text{saw mary})
\]
Naive CYK-like parsing: example

Example

John saw mary

\[
\text{John} \quad \text{saw} \quad \text{saw} \quad \text{Mary} \\
\text{John} \quad \text{saw} \quad \text{saw} \quad \text{Mary} \\
\text{(John)} (\text{saw mary}) \quad \text{John} \\
\text{John} \quad \text{saw} \\
\text{saw} \quad \text{Mary}
\]
Naive CYK-like parsing: example

Example

John saw mary

[(John) (saw mary)]

\[\text{John} \rightarrow \text{saw} \rightarrow \text{John} \rightarrow \text{saw} \rightarrow \text{saw} \rightarrow \text{Mary} \rightarrow \text{saw} \rightarrow \text{Mary} \rightarrow \text{John} \]

\[\text{John} \rightarrow \text{saw} \rightarrow \text{John} \rightarrow \text{saw} \rightarrow \text{Mary} \rightarrow \text{saw} \rightarrow \text{Mary} \rightarrow \text{John} \]

\[\text{John} \rightarrow \text{saw} \rightarrow \text{Mary} \rightarrow \text{John} \rightarrow \text{saw} \rightarrow \text{Mary} \rightarrow \text{John} \rightarrow \text{saw} \rightarrow \text{Mary} \]

\[\ldots\]
Graph Based Dependency Parsing

Parsing algorithms

Projective dependency parsing

Naive CYK-like parsing: example

Example

John saw mary

(John saw) (mary)
Naive CYK-like parsing: example

Example

John saw mary

\[ \text{John} \quad \text{saw} \quad \text{saw} \quad \text{Mary} \]

\[ \text{John} \quad \text{saw} \quad \text{John} \quad \text{saw} \quad \text{saw} \quad \text{Mary} \quad \text{saw} \quad \text{Mary} \]

\( (\text{John saw}) \quad (\text{mary}) \)

\[ \text{John} \quad \text{Mary} \]

\[ \text{John} \quad \text{saw} \]

\[ \ldots \]
Naive CYK-like parsing: example

Example

John saw mary
Naive CYK-like parsing

\[
C[s][t][i] = \max_{s \leq q < t, s \leq j \leq t} \left\{ C[s][q][i] + C[q + 1][t][j] + \lambda(w_i, w_j) \text{ if } j > i \\
C[s][q][j] + C[q + 1][t][i] + \lambda(w_j, w_i) \text{ if } j < i \right. 
\]
Naive CYK-like parsing: why is it slow?

The time complexity is $O(n^5)$!

Heads are in the middle, we need extra indices.
Graph Based Dependency Parsing

- Parsing algorithms
- Projective dependency parsing

Extension: Eisner's algorithm

Intuition

- For each word, building left dependents is independent of building right dependents.
- Get rid of the inner indices of heads.
Eisner’s algorithm

CKY

Eisner

\[ \begin{align*}
\text{a} & : (i, j) \\
\text{b} & : (i, j) \\
\text{c} & : (i, j) \\
\text{d} & : (i, j)
\end{align*} \]
Eisner’s algorithm

Figure: $E[s][t][0][1]$  

Figure: $E[s][t][0][0]$  

Figure: $E[s][t][1][1]$  

Figure: $E[s][t][1][0]$
Eisner’s algorithm

\[ E[s][t][0][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda_{(w_t, w_s)}) \]
Eisner’s algorithm

\[ E[s][t][0][1] = \max_{s \leq q < t} \left( E[s][q][1][0] + E[q+1][t][0][0] + \lambda_{(w_t,w_s)} \right) \]
Eisner’s algorithm

John saw Mary

\[ E[s][t][0][1] = \max_{s \leq q < t}(E[s][q][1][0] + E[q + 1][t][0][0] + \lambda_{(w_t, w_s)}) \]

\[ E[s][t][1][1] = \max_{s \leq q < t}(E[s][q][1][0] + E[q + 1][t][0][0] + \lambda_{(w_s, w_t)}) \]
Eisner’s algorithm

\[ E[s][t][0][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda_{(wt,ws)}) \]

\[ E[s][t][1][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda_{(ws,wt)}) \]

\[ E[s][t][1][0] = \max_{s < q \leq t} (E[s][q][1][1] + E[q][t][1][0]) \]

\[ E[s][t][0][0] = \max_{s \leq q < t} (E[s][q][0][0] + E[q][t][0][1]) \]
Eisner’s algorithm

\[ E[s][t][0][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda(w_t,w_s)) \]

\[ E[s][t][1][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda(w_s,w_t)) \]

\[ E[s][t][1][0] = \max_{s < q \leq t} (E[s][q][1][1] + E[q][t][1][0]) \]
Eisner’s algorithm

\[
E[s][t][0][1] = \max_{s \leq q < t}(E[s][q][1][0] + E[q + 1][t][0][0] + \lambda(w_t, w_s))
\]

\[
E[s][t][1][1] = \max_{s \leq q < t}(E[s][q][1][0] + E[q + 1][t][0][0] + \lambda(w_s, w_t))
\]

\[
E[s][t][1][0] = \max_{s < q \leq t}(E[s][q][1][1] + E[q][t][1][0])
\]

\[
E[s][t][0][0] = \max_{s \leq q < t}(E[s][q][0][0] + E[q][t][0][1])
\]
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
Maximum spanning tree

Graph Based Dependency Parsing

- Parsing algorithms
- Non-projective dependency parsing (MST)

Root

John

saw

Mary

yesterday
Maximum spanning tree

John saw Mary yesterday.
Chu-Liu-Edmonds algorithm

**Ideas**

- **Greedy**: always try to select the edges with highest weight.
- **Contract**: if circles occur, always try to break the circle with least value lost.
- **Recursive**: repeat this procedure until get the MST.
Chu-Liu-Edmonds algorithm: example
Chu-Liu-Edmonds algorithm: example

For each node, select incoming arc with highest weight.
If there is no circle, done!
Chu-Liu-Edmonds algorithm: example
Chu-Liu-Edmonds algorithm: example

No trick for the outgoing arc from the new node. Select the arc with the highest weight.
Chu-Liu-Edmonds algorithm: example

John saw Mary

root

9

10

30

w_{js} 20

Mary
Chu-Liu-Edmonds algorithm: example

Graph Based Dependency Parsing
- Parsing algorithms
- Non-projective dependency parsing (MST)
Chu-Liu-Edmonds algorithm: example

```
John saw Mary
```

Diagram:
- Root node
- Edges: John → saw → Mary
- Weights: 9, 10, 20, 30, w_{js}
Chu-Liu-Edmonds algorithm: example

Graph Based Dependency Parsing
- Parsing algorithms
- Non-projective dependency parsing (MST)
Chu-Liu-Edmonds algorithm: example

```
root

9
John

0
saw

w_{js} 20

40

Mary
```
Chu-Liu-Edmonds algorithm: example

- John saw Mary
- John's weight is 40
- Mary's weight is 30
- The edge weight $w_{js}$ is 0
- The weight of the path is 31
Chu-Liu-Edmonds algorithm: example
Chu-Liu-Edmonds algorithm: example

John saw Mary

root

saw

10

30

30

w_{js}

Mary

John
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
Local learning

Given training data \((X, Y)\)

<table>
<thead>
<tr>
<th>Word_pair</th>
<th>Link_label</th>
<th>Instance_weight</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>John-saw</td>
<td>L</td>
<td>1</td>
<td>(W_1\text{-John, } W_2\text{-saw,})  (W_1\ W_2\text{-John_saw,})  (Pos_1\text{-noun,})  (Pos_2\text{-verb,}\ldots)</td>
</tr>
<tr>
<td>saw-Mary</td>
<td>R</td>
<td>1</td>
<td>(W_1\text{-saw, } W_2\text{-Mary,})  (W_1\ W_2\text{-saw_Mary,})  (Pos_1\text{-verb,})  (Pos_2\text{-noun,}\ldots)</td>
</tr>
<tr>
<td>John-Mary</td>
<td>N</td>
<td>1</td>
<td>(W_1\text{-John, } W_2\text{-Mary,})  (W_1\ W_2\text{-saw_Mary,})  (Pos_1\text{-noun,})  (Pos_2\text{-noun,}\ldots)</td>
</tr>
</tbody>
</table>

...
Local learning

- linear classifier $\rightarrow$ link classifier
- For each word pair in a sentence: No arc, left arc, right arc?
- Each arc is scored separately without knowledge of other arcs
Online large-margin training

Intuition

- Not feed all of the training data once.
- Update $\vec{w}$ step by step instead.
- Average on the sequence of $\vec{w}^*$
Outline

1. Introduction to dependency parsing
2. Graph based dependency parsing
3. Parsing algorithms
   - Projective dependency parsing
   - Non-projective dependency parsing (MST)
4. Learning framework
5. Evaluation
Evaluation methods

Simply use(labeled) dependency accuracy

Root: John saw Mary yesterday.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Root</th>
<th>PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>saw</td>
<td>SBJ</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>saw</td>
<td>OBJ</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>saw</td>
<td>TMP</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>saw</td>
<td>PU</td>
</tr>
</tbody>
</table>

Table: Gold
Evaluation methods

Simply use(labeled) dependency accuracy

Root John saw Mary yesterday.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>Root</td>
<td>PRED</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>saw</td>
<td>SBJ</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>saw</td>
<td>OBJ</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>saw</td>
<td>TMP</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>saw</td>
<td>PU</td>
</tr>
</tbody>
</table>

Table: Parsed result
Evaluation methods

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Root</th>
<th>PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>saw</td>
<td>SBJ</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>saw</td>
<td>OBJ</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>saw</td>
<td>TMP</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>saw</td>
<td>PU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Root</th>
<th>PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>saw</td>
<td>SBJ</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>saw</td>
<td>OBJ</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>saw</td>
<td>TMP</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>saw</td>
<td>TMP</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>saw</td>
<td>PU</td>
</tr>
</tbody>
</table>

Table: Gold

\[
\text{accuracy} = \frac{4}{5} = 0.8
\]

- No need to use F-measure.
- Other metrics: Complete right tree . . .
### Evaluation results

<table>
<thead>
<tr>
<th></th>
<th>Czech-A</th>
<th></th>
<th>Czech-B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Complete</td>
<td>Accuracy</td>
<td>Complete</td>
</tr>
<tr>
<td>COLL1999</td>
<td>82.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>N&amp;N2005</td>
<td>80.0</td>
<td>31.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>McD2005</td>
<td>83.3</td>
<td>31.3</td>
<td>74.8</td>
<td>0.0</td>
</tr>
<tr>
<td>Single-best MIRA</td>
<td>84.1</td>
<td>32.2</td>
<td>81.0</td>
<td>14.9</td>
</tr>
<tr>
<td>Factored MIRA</td>
<td>84.4</td>
<td>32.3</td>
<td>81.5</td>
<td>14.3</td>
</tr>
</tbody>
</table>

**Table:** Dependency parsing results for Czech. Czech-B is the subset of Czech-A containing only sentences with at least one non-projective dependency.
### Evaluation results

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>McD2005</td>
<td>90.9</td>
<td>37.5</td>
</tr>
<tr>
<td>Singe-best MIRA</td>
<td>90.2</td>
<td>33.2</td>
</tr>
<tr>
<td>Factored MIRA</td>
<td>90.2</td>
<td>32.3</td>
</tr>
</tbody>
</table>

**Table:** Dependency parsing results for English using spanning tree algorithms.
Thank you!

Any questions?
Eisner’s algorithm

\[ S = w_0 w_1 \cdots w_n \]

Arc weight parameters \( \lambda_{w_i, w_j} \in \Lambda \)

Instantiate \( E[n][n][2][2] \in R \)

Initialization: \( E[s][s][d][c] = 0 \)

for \( m: 1 \ldots n \) do
  for \( s: 1 \ldots n \) do
    \( t = s + m \)
    if \( t > n \) then
      break
    end if
    \( E[s][t][0][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda(w_t, w_s)) \)
    \( E[s][t][1][1] = \max_{s \leq q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + \lambda(w_s, w_t)) \)
    \( E[s][t][0][0] = \max_{s \leq q < t} (E[s][q][0][0] + E[q][t][0][1]) \)
    \( E[s][t][1][0] = \max_{s < q \leq t} (E[s][q][1][1] + E[q][t][1][0]) \)
  end for
end for
MIRA learning algorithm

Training data: $\mathcal{T} = (x_t, y_t)_{t=1}^T$

$\vec{w}_0 = 0; \vec{(v)} = 0; i = 0$

for $n : 1 \cdots N$ do
  for $t : 1 \cdots T$ do
    min $||\vec{w}^{i+1} - \vec{w}^i||$
    s.t. $s(x_t, y_t) - s(x_t, y') \geq L(y_t, y'), \forall y' \in \Phi(x_t)$
    $\vec{v} = \vec{v} + \vec{w}^{i+1}$
    $i = i + 1$
  end for
end for

$\vec{w} = \frac{\vec{v}}{N \cdot T}$