Non-projective Dependency Parsing using Spanning Tree Algorithms

Qian Zhong
Outline

• Basic Concepts
  • Edge-based factorization
  • Parsing Algorithm & Learning Algorithm
  • Experiments and Results
Recap: Dependency Parsing Basics

• **Dependency relations**: syntactic structure essentially consists of words linked by binary, asymmetrical relations

*Dependency Structure of English Sentence, Figure Adapted from Dependency Parsing (Kübler et.al, 2009, p2)*
Recap: Projective Trees

• If we say a tree is **projective**, we mean that if we put the words in their linear order, preceded by the root, **the edges can be drawn above the words without crossings**, or, equivalently, a word and its descendants form a contiguous substring of the sentence.
Recap: Projective vs. non-projective Dependency Trees

- Projective Dependency Trees

![Figure A](image)

- Non-Projective Dependency Trees

![Figure B](image)

Figures adapted from McDonald et al., 2005
Motivation: Dependency Parsing

• More efficient to Learn and parse while still encoding much of the predicate-argument information needed in applications

• Applications

  Relation Extraction(Culotta and Sorensen, 2004)
  Machine Translation(Ding and Palmer, 2005)
  Synonym Generation(Shinya et al., 2002)
  Lexical Resource Augmentation(Snow et al., 2004)
Motivation: Non-projective Trees

Why?

25% of more of the sentences in some languages cannot be given a linguistically adequate analysis without invoking non-projective structures

(Nivre, 2009; Nivre, 2006; Kuhlman and Nivre, 2006; Havelka, 2007)

In languages with more flexible word order than English, such as German, Dutch and Czech, non-projective dependencies are more frequent.
Outline

- Basic Concepts
- **Edge-based Factorization**
- Parsing Algorithm & Learning Algorithm
- Experiments and Results
Dependency Parsing and Spanning Trees

• Edge based Factorization

Sentence: $x = x_1 \ldots x_n$

**Dependency Tree** $y$

• the set of tree edges
• $(i, j) \in y$ if there is a dependency in $y$ from word $x_i$ to word $x_j$

**Score of the dependency tree**
the sum of score of all the edges in the tree
• Edge based Factorization

**score of an edge**: the dot product between a high dimensional feature representation of the edge and a weight vector

\[ s(i, j) = w \cdot f(i, j) \]

**score of a dependency tree y for sentence x**: 

\[ s(x, y) = \sum_{(i,j) \in y} s(i, j) = \sum_{(i,j) \in y} w \cdot f(i, j) \]

**Dependency parsing**: finding the dependency tree y with the **highest score** for given sentence x
Outline

• Basic Concepts
• Edge Based Factorization
• Parsing Algorithm & Learning Algorithm
• Experiments and Results
Maximum Spanning Trees: \textbf{Projective}

\section*{A Generic Directed Graph $G=(V, E)$}

Vertex Set: $V=\{v_1, \ldots, v_n\}$

Set $E \subseteq [1:n] \times [1:n]$ of pairs $(i, j)$ of directed edges $v_i \rightarrow v_j$

Score of each edge $s(i, j)$

$G$ is directed, $s(i, j)$ does not necessarily equal $s(j, i)$

\section*{A Maximum Spanning Tree (MST) of $G$ is a tree $y$}

$y \subseteq E$

that maximizes the value $\sum_{(i,j) \in y} s(i, j)$ for every vertex in $V$

\section*{For each sentence $x$, we define the directed graph}

$G_x = (V_x, E_x)$

$V_x = \{x_0 = \text{root}, x_1, \ldots, x_n\}$

$E_x = \{(i,j): i \neq j, (i, j) \in [0:n] \times [1:n]\}$
**Algorithm**

- **Chu-Liu-Edmonds** $(G, s)$
  - Graph $G = (V, E)$
  - Edge weight function $s : E \rightarrow \mathbb{R}$
  - Let $M = \{(x^*, x) : x \in V, x^* = \arg \max_{x'} s(x', x)\}$
  - Let $G_M = (V, M)$
  - If $G_M$ has no cycles, then it is an MST: return $G_M$
  - Otherwise, find a cycle $C$ in $G_M$
  - Let $G_C = \text{contract}(G, C, s)$
  - Let $y = \text{Chu-Liu-Edmonds}(G_C, s)$
  - Find a vertex $x \in C$ s.t. $(x', x) \in y, (x'', x) \in C$
  - return $y \cup C - \{(x'', x)\}$

- **contract** $(G = (V, E), C, s)$
  - Let $G_C$ be the subgraph of $G$ excluding nodes in $C$
  - Add a node $c$ to $G_C$ representing cycle $C$
  - For $x \in V - C : \exists_{x' \in C} (x', x) \in E$
    - Add edge $(c, x)$ to $G_C$ with
      - $s(c, x) = \max_{x' \in C} s(x', x)$
  - For $x \in V - C : \exists_{x' \in C} (x, x') \in E$
    - Add edge $(x, c)$ to $G_C$ with
      - $s(x, c) = \max_{x' \in C} [s(x, x') - s(a(x'), x') + s(C)]$
        - where $a(v)$ is the predecessor of $v$ in $C$
        - and $s(C) = \sum_{v \in C} s(a(v), v)$
  - return $G_C$

---

**Figure 3**: Chiu-Liu-Edmonds algorithm for finding maximum Spanning Trees in Directed Graph
Maximum Spanning Trees: Non-projective
Maximum Spanning Trees: projective

A dynamic programming table

\[ C[s][t][i] \]

string start \( W_s \)

Root

string ends \( W_t \)

represents the value of the highest scoring projective tree that spans the string \( w_s \ldots w_t \) and which is rooted at word \( w_i \), where \( s \leq i \leq t \)

E.g. \( s=0, i=0 \)

Then \( C[0][n][0] \) would represent the value of highest scoring dependency tree for an input sentence \( S = w_0w_1 \ldots w_n \), which is precisely the value we are interested in for the parsing problem.
Maximum Spanning Trees: projective

\[ C[i][i][i] = 0.0, \text{ for all } 0 \leq i \leq n \]

**Figure 4.3:** Illustration showing that every projective subgraph can be broken into a combination of smaller adjacent subgraphs.
Maximum Spanning Trees: projective

The final tree for a sentence $S$ is then $G = (V, A[0][n][0])$.

**Figure 4.4:** CKY algorithm for projective dependency parsing.

$$
C[s][t][i] = \max_{s \leq q < t, s \leq j \leq t} \begin{cases} 
C[s][q][i] + C[q + 1][t][j] + \lambda(w_i, w_j) & \text{if } j > i \\
C[s][q][j] + C[q + 1][t][i] + \lambda(w_i, w_j) & \text{if } j < i
\end{cases}
$$

$$
A[s][t][i] = \begin{cases} 
A[s][q][i] \cup A[q + 1][t][j] \cup (w_i, w_j) & \text{if } j > i \\
A[s][q][j] \cup A[q + 1][t][i] \cup (w_i, w_j) & \text{if } j < i
\end{cases}
$$
Maximum Spanning Trees: projective

**Figure 4.5:** Illustration showing Eisner’s projective dependency parsing algorithm relative to CKY.

**Figure 4.6:** Illustration showing each type of subgraph in the dynamic program table used in Eisner’s algorithm.
Maximum Spanning Trees: projective

**Pseudo-code for Eisner’s algorithm**

```
Eisner(S, Γ, λ)
    Sentence S = w_0 w_1 ... w_n
    Arc weight parameters λ_{(w_i, w_j)} ∈ λ
    1. Instantiate E[n][n][2][2] ∈ ℝ
    2. Initialization: E[s][s][d][c] = 0.0 for all s, d, c
    3. for m : 1..n
        4. for s : 1..n
            5. t = s + m
            6. if t > n then break

    % Create subgraphs with c = 1 by adding arcs (step a-b in figure 4.5)
    7. E[s][t][0][1] = max_{s ≤ q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + λ_{(w_t, w_s)})
    8. E[s][t][1][1] = max_{s ≤ q < t} (E[s][q][1][0] + E[q + 1][t][0][0] + λ_{(w_s, w_t)})

    % Add corresponding left/right subgraphs (step b-c in figure 4.5)
    9. E[s][t][0][0] = max_{s ≤ q < t} (E[s][q][0][0] + E[q][t][0][1])
    10. E[s][t][1][0] = max_{s < q ≤ t} (E[s][q][1][1] + E[q][t][1][0])
```

*Figure Adapted from Dependency Parsing (Kübler et al., 2009, p53)*
Outline

• Basic Concepts
• Edge-based factorization
• Parsing Algorithm & Learning Algorithm
• Experiments and Results
Online Large Margin Learning: MIRA

Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al, 2003)

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

1. $w_0 = 0; \ v = 0; \ i = 0$
2. for $n : 1..N$
3. \quad for $t : 1..T$
4. \quad \min \left\| w^{(i+1)} - w^{(i)} \right\|
\quad \text{s.t.} \ s(x_t, y_t) - s(x_t, y') \geq L(y_t, y'), \forall y' \in dt(x_t)$
5. \quad $v = v + w^{(i+1)}$
6. \quad $i = i + 1$
7. $w = v / (N \times T)$

MIRA learning Algorithm
Online Large Margin Learning: single-best MIRA

The resulting online update

\[
\begin{align*}
\min & \|w^{(i+1)} - w^{(i)}\| \\
\text{s.t.} & \quad s(x_t, y_t) - s(x_t, y') \geq L(y_t, y') \\
\text{where} & \quad y' = \arg\max_{y'} s(x_t, y')
\end{align*}
\]

Related

- k highest-scoring trees with small k (McDonald et al., 2005)
- averaged perceptron algorithm (Collins, 2002) using the single highest scoring tree to update the weight vector

MIRA updates \(w\) to **maximise the margin between the corrected tree and the highest scoring tree leading to increasing accuracy**
Online Large Margin Learning: factored MIRA

Factoring the output by edges to obtain the following statements

\[
\min \| \mathbf{w}^{(i+1)} - \mathbf{w}^{(i)} \|
\]

s.t. \( s(l, j) - s(k, j) \geq 1 \)

\( \forall (l, j) \in \mathbf{y}_t, (k, j) \notin \mathbf{y}_t \)

- the weight of the correct incoming edge to the word \( x_j \) and the weight of all other incoming edges must be separated by a margin of 1

- the correct spanning tree and all incorrect spanning trees are separated by a score at least as large as the number of incorrect incoming edges.
Outline

• Basic Concepts
• Edge-based factorization
• Learning Algorithm & parsing algorithm

• Experiments and Results
Experiments

• Czech Prague Dependency Treebank (PDT) (Hajič, 1998; Hajič et al., 2001)
• they used predefined training, developing and testing split of the data set
• automatically generated POS tags that are provided with the data
• features only relied on the reduced POS tag set from Collins et al. (1999)
• **23%** of the sentences in the training, development and test sets have at least one non-projective dependency
• less than **2%** of total edges are actually non-projective
• therefore, handling non-projective edges correctly have a relatively small effect on overall accuracy
• Czech A, consists of the entire PDT
• Czech B, includes only the **23%** of sentences with at least one non-projective dependency


3. **McD2005**: The projective parser of McDonald et al. (2005) that uses the Eisner algorithm for both training and testing. This system uses $k$-best MIRA with $k=5$.

4. **Single-best MIRA**: In this system we use the Chu-Liu-Edmonds algorithm to find the best dependency tree for Single-best MIRA training and testing.

5. **Factored MIRA**: Uses the quadratic set of constraints based on edge factorization as described in Section 3.2. We use the Chu-Liu-Edmonds algorithm to find the best tree for the test data.
Results

Table 1: Dependency parsing results for Czech. Czech-B is the subset of Czech-A containing only sentences with at least one non-projective dependency.

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

Table 2: Dependency parsing results for English using spanning tree algorithms.

<table>
<thead>
<tr>
<th></th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td>McD2005</td>
<td><strong>90.9</strong></td>
</tr>
<tr>
<td>Single-best MIRA</td>
<td>90.2</td>
</tr>
<tr>
<td>Factored MIRA</td>
<td>90.2</td>
</tr>
</tbody>
</table>
Summary

• formalize weighted dependency parsing as searching for maximum spanning trees (MSTs) in directed graphs

• Parsing Algorithm
  Non-projective: Chiu-Liu-Edmonds
  Projective: Eisner’s Algorithm

• Learning Algorithm: single/factored MIRA

• evaluated on the Prague Dependency Treebank and increasing in efficiency and accuracy
References


Thank you!

Any Questions?