GRAPH-BASED DEPENDENCY PARSING

Marina Valeeva
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
What is Dependency Parsing?

- Input: a sentence, output: dependency tree

- Dependency structures contain much of predicate-argument information

<table>
<thead>
<tr>
<th>What is Dependency Parsing good for?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Translation</td>
</tr>
<tr>
<td>Synonym Generation</td>
</tr>
<tr>
<td>Relation Extraction</td>
</tr>
<tr>
<td>Lexical Resource Augmentation</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
What is a Dependency Tree?

- Consists of lexical items linked by binary asymmetric relations called dependencies
- The arcs (links) indicate certain grammatical relation between words
- Each word depends on exactly one parent
- The tree starts with a root node

```
root
John -> hit -> the ball -> with -> the bat
```
Properties of a dependency tree

- Acyclicity
- Connectivity
- Projectivity or non-projectivity
- Single head
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Projective Dependency Tree (English)
Non-Projective Dependency Tree (English)
He is mostly not even interested in the new things and in most cases, he has no money for it either.
Projectivity vs. Non-Projectivity

**Projective**
- No crossing edges
- Don’t allow complex constructions in the parse

**Non-Projective**
- With crossing edges
- Good for languages with free word order
- Good for long distance dependencies
1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Graph-based dependency parsing

- Defining candidate dependency trees for an input sentence

- **Learning**: scoring possible dependency graphs for a given sentence, usually by factoring the graphs into their component arcs

- **Parsing**: searching for the highest scoring graph for a given sentence

- Globally trained and use exact inference algorithms

- Define features over a limited history of parsing decisions
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Edge Based Factorization

- $x$ - an input sentence
- $y$ - a dependency tree for an input sentence $x$
- $(i, j) \in y$ - a dependency edge in $y$ from word $x_i$ to word $x_j$
- $w$ - a weight vector
- $f(i, j)$ - a feature representation of an edge
- $s(i, j)$ - the score of an edge

$$s(i, j) = w \cdot f(i, j)$$

- $s(x, y)$ - score of a dependency tree $y$ for sentence $x$

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

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Margin Infused Relaxed Algorithm (MIRA)

- MIRA is an online learning algorithm
- Used for learning weight vector \( w \)
- Considers a single training instance at each update to \( w \)
- Final weight vector is the average of the weight vectors after each iteration
- Loss of a tree is the number of words with incorrect parents relative to the correct tree
- **Single-best MIRA:** using only single margin constraint for the tree with the highest score
- **Factored MIRA:** the weight of the correct incoming edge to the word and the weight of all other incoming edges must be separated by a margin of 1.
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Generative Model

- Each time a word \( i \) is added, it generates a Markov sequence of (tag, word) pairs to serve as its left children and a separate sequence of (tag, word) pairs as its right children.

- Markov process begins from START state and ends at STOP state.

- Probabilities depend on: the word \( i \), its tag, the symbols which are generated are added as \( i \)'s children (from closest to farthest).

- The process recurses for each child.
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Eisner’s Algorithm(1)

- Bottom-up dependency parsing algorithm

- Adding one link at a time making it easy to multiply the model’s probability factors.

- Similar to CKY method

- Runtime: $O(n^3)$

- Instead of storing subtrees, storing spans

- Non-constituent spans will be concatenated into larger spans
Eisner’s Algorithm (2)

- Span = substring where no internal word links to any word outside of the span

- A span consists of:
  - >= 2 adjacent words
  - Tags for all these words
  - A list of all dependency links between words in the span.

- No cycles, no multiple parents, no crossing links

- Each internal word has a parent in the span
A span of the dependency parse with either one parentless endword or two parentless endwords.

In a span, only the endwords are active (meaning they still need a parent).

Internal part of the span is grammatically inert.

... dachshund over there can really **play** ...
Eisner’s Algorithm

- **Covered-concatenation**: if span a ends on the same word $i$ that starts span b, then the parser tries to combine two
1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Maximum Spanning Tree (MST)

- Finding dependency tree with highest score = finding MST in directed graphs
- Scores are independent of other dependencies
- Score of dependency tree = sum of scores of dependencies in the tree
- Runtime: $O(n^2)$
Maximum Spanning Tree (MST)

- For each input sentence $x$:
  
  - $G_x = (V_x, E_x)$ - generic directed graph
  
  - $V_x = \{x_0 = \text{root}, x_1, \ldots, x_n\}$ - vertex set
  
  - $E_x = \{(i, j) : i \neq j, (i, j) \in [0:n] \times [1:n]\}$ - set of pairs of directed edges
  
  - $\sum_{(i, j) \in y} s(i, j)$ - MST of $G$ is a tree $y \subseteq E$ that maximizes the value such that every vertex in $V$ appears in $y$. 
Finding the MST with Chu-Liu-Edmonds Algorithm

**Chu-Liu-Edmonds**\((G, s)\)

Graph \(G = (V, E)\)

Edge weight function \(s : E \rightarrow \mathbb{R}\)

1. Let \(M = \{(x^*, x) : x \in V, x^* = \arg \max_{x'} s(x', x)\}\)
2. Let \(G_M = (V, M)\)
3. If \(G_M\) has no cycles, then it is an MST: return \(G_M\)
4. Otherwise, find a cycle \(C\) in \(G_M\)
5. Let \(G_C = \text{contract}(G, C, s)\)
6. Let \(y = \text{Chu-Liu-Edmonds}(G_C, s)\)
7. Find a vertex \(x \in C\) s.t. \((x', x) \in y, (x'', x) \in C\)
8. return \(y \cup C - \{(x'', x)\}\)

**contract**\((G = (V, E), C, s)\)

1. Let \(G_C\) be the subgraph of \(G\) excluding nodes in \(C\)
2. Add a node \(c\) to \(G_C\) representing cycle \(C\)
3. For \(x \in V - C : \exists_{x' \in C}(x', x) \in E\)
   Add edge \((c, x)\) to \(G_C\) with \(s(c, x) = \max_{x' \in C} s(x', x)\)
4. 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 \(s(C) = \sum_{v \in C} s(a(v), v)\)
5. return \(G_C\)

**Greedy:** edges with the highest weight are selected.

**Contract:** if cycle occur, tries to break the cycle with the least value lost

**Recursive:** repeat until get the MST
Chu-Liu-Edmonds Algorithm (1)

- John saw Mary
- Goal: find the highest scoring tree for the input sentence
- Directed graph representation $G_x$
For each word in the graph, find the incoming edge with the highest weight

Check if the result is the tree
- if yes, then MST
- If not, there we have a cycle
Chu-Liu-Edmonds Algorithm (3)

- Identify a cycle, then contract into a single node and recalculate the weights
Chu-Liu-Edmonds Algorithm (4)

- Adding the outgoing edge with the highest score (Mary)
- Adding the incoming edge with the highest score (root)
- Keep track of the real endpoints of the edges which go into and out of $w_{js}$

![Diagram showing the edges and labels]
The resulting spanning tree = the best non-projective dependency tree
Outline

1. Introduction
   - What is Dependency Parsing?
   - What is a Dependency Tree?
   - Projectivity vs. Non-Projectivity

2. Graph-Based Dependency Parsing

3. Models
   - Edge Based Factorization
   - MIRA
   - Generative Model

4. Parsing Algorithms
   - Projective Dependency Parsing
     - Eisner’s Algorithm
   - Non-Projective Dependency Parsing
     - Maximum Spanning Tree
     - Chu-Liu-Edmonds Algorithm

5. Experiments and Evaluation Results
Experiments

- **Czech**: on Czech Prague Dependency Treebank (PDT)
  - Using predefined training, dev and test sets
  - 23% of sentences on average have at least one non-projective dependency
  - 2 data sets
    - Czech A: entire PDT
    - Czech B: subset of Czech A containing only sentences with at least one non-projective dependency

- **English**: on Penn Treebank

- **Accuracy**: number of words that correctly identified parent in the tree

- **Complete**: number of completely correct trees
## Evaluation results (Czech)

- Dependency parsing results for Czech.

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

- Dependency parsing results for English using spanning tree algorithms

<table>
<thead>
<tr>
<th>Model</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>Single-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>
## Eisner Algorithm vs. Chu-Liu-Edmonds Algorithm

<table>
<thead>
<tr>
<th>Eisner Algorithm</th>
<th>Chu-Liu-Edmonds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projective</td>
<td>Non-Projective</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>Top-Down Recursive</td>
</tr>
<tr>
<td>Runtime (O(n^3))</td>
<td>Runtime (O(n^2))</td>
</tr>
<tr>
<td>Works better for</td>
<td>Works better for multiple</td>
</tr>
<tr>
<td>English (efficiency)</td>
<td>languages</td>
</tr>
<tr>
<td></td>
<td>Works better with languages of</td>
</tr>
<tr>
<td></td>
<td>free word order</td>
</tr>
</tbody>
</table>