Graph-Based Dependency Parsing

Sandra Kübler, Markus Dickinson

Based on slides from Ryan McDonald and Joakim Nivre
Notation

- Sentence $x = w_0, w_1, \ldots, w_n$, with $w_0 = \text{root}$
- $L = \{l_1, \ldots, l_{|L|}\}$ set of permissible arc labels
- Let $G = (V, A)$ be a dependency graph for sentence $x$ where:
  - $V = \{0, 1, \ldots, n\}$ is the vertex set
  - $A$ is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from $w_i$ to $w_j$ with label $l_k \in L$
- By the usual definition, $G$ is a tree
Data-Driven Parsing

- Goal: Learn a good predictor of dependency graphs
- Input: $x$
- Output: dependency graph/tree $G$

**Tuesday:**
- Parameterize parsing by transitions
- Learn to predict transitions given the input and a history
- Predict new graphs using deterministic parsing algorithm

**Today:**
- Parameterize parsing by dependency arcs
- Learn to predict entire graphs given the input
- Predict new graphs using spanning tree algorithms
Some Graph Theory

- A graph $G = (V, A)$ is a set of vertices $V$ and arcs $(i, j) \in A$, where $i, j \in V$
- Undirected graphs: $(i, j) \in A \iff (j, i) \in A$
- Directed graphs (digraphs): $(i, j) \in A \not\Rightarrow (j, i) \in A$
Multi-Digraphs

- A multi-digraph is a digraph where there can be multiple arcs between vertices
- $G = (V, A)$
- $(i, j, k) \in A$ represents the $k^{th}$ arc from vertex $i$ to vertex $j$
Directed Spanning Trees (a.k.a. Arborescence)

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - $G'$ is a tree (acyclic)

- A spanning tree of the following (multi-)digraphs

![Directed Spanning Trees Diagrams]
Weighted Directed Spanning Trees

- Assume we have a weight function for each arc in a multi-digraph \( G = (V, A) \)
- Define \( w_{ij}^k \geq 0 \) to be the weight of \((i, j, k) \in A\) for a multi-digraph
- Define the weight of directed spanning tree \( G' \) of graph \( G \) as

\[
w(G') = \prod_{(i,j,k) \in G'} w_{ij}^k
\]

- **Notation:** \((i, j, k) \in G = (V, A) \iff \text{the arc } (i, j, k) \in A\)
Maximum Spanning Trees (MST) of (Multi-)Digraphs

- Let $T(G)$ be the set of all spanning trees for graph $G$.

- The MST Problem: Find the spanning tree $G'$ of the graph $G$ that has highest weight.

\[
G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \prod_{(i,j,k) \in G'} w_{ij}^k
\]

- Solutions ... to come.
Arc-Factored Dependency Models

- Remember: Data-driven parsing parameterizes model and then learns parameters from data

- **Arc-factored model**
  - Assumes that the score / probability / **weight** of a dependency graph factors by its arcs

  \[ w(G) = \prod_{(i,j,k) \in G} w_{ij}^k \]

  look familiar?

  - \( w_{ij}^k \) is the weight of creating a dependency from word \( w_i \) to \( w_j \) with label \( l_k \)

  - Thus there is an assumption that each dependency decision is independent
    - Strong assumption! Will address this later.
Arc-Factored Dependency Models Example

- Weight of dependency graph is $10 \times 30 \times 30 = 9000$

- In practice arc weights are much smaller
Three Important Problems

1. **Inference** \( \equiv \) finding the MST of \( G_x \)

\[
G = \arg \max_{G \in T(G_x)} w(G) = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k
\]

2. Defining \( w_{ij}^k \) and its **feature space**

3. **Learning** \( w_{ij}^k \)
   - Can use perceptron-based learning if we solve (1)
Chu-Liu-Edmonds Algorithm

- Finds the MST originating out of a vertex of choice
- Assumes weight of tree is sum of arc weights
- No problem, we can use logarithms

\[
G = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k
\]

\[
= \arg \max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} w_{ij}^k
\]

\[
= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} \log w_{ij}^k
\]

So if we let \( w_{ij}^k = \log w_{ij}^k \), then we get

\[
G = \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w_{ij}^k
\]
Chu-Liu-Edmonds

- $x = \text{root}$ John saw Mary
- Remove all arcs into the root node
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex

- If this is a tree, then we have found MST!!
Chu-Liu-Edmonds

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle
Chu-Liu-Edmonds

- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John → Mary is 3 and saw → Mary is 30
Chu-Liu-Edmonds

- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29
Chu-Liu-Edmonds

► This is a tree and the MST for the contracted graph!!

► Go back up recursive call and reconstruct final graph
Chu-Liu-Edmonds

- This is the MST!!
Chu-Liu-Edmonds

- Naive implementation $O(n^3 + |L|n^2)$
  - Converting $G_x$ to a digraph – $O(|L|n^2)$
  - Finding best arc – $O(n^2)$
  - Contracting cycles – $O(n^2)$
  - At most $n$ recursive calls

- Better algorithms run in $O(|L|n^2)$

- Chu-Liu-Edmonds searches all dependency graphs
  - Both projective and non-projective
  - Thus, it is an exact non-projective search algorithm!!!

- What about the projective case?
One more Example

▶ $x = \text{root If shit happens, you deserve it. (Catholicism)}$

<table>
<thead>
<tr>
<th>head</th>
<th>dep.</th>
<th>weight</th>
<th>head</th>
<th>dep.</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>shit</td>
<td>9</td>
<td>happens</td>
<td>it</td>
<td>17</td>
</tr>
<tr>
<td>root</td>
<td>happens</td>
<td>21</td>
<td>you</td>
<td>shit</td>
<td>13</td>
</tr>
<tr>
<td>root</td>
<td>you</td>
<td>11</td>
<td>you</td>
<td>happens</td>
<td>11</td>
</tr>
<tr>
<td>root</td>
<td>deserve</td>
<td>30</td>
<td>you</td>
<td>deserve</td>
<td>35</td>
</tr>
<tr>
<td>root</td>
<td>it</td>
<td>12</td>
<td>deserve</td>
<td>If</td>
<td>15</td>
</tr>
<tr>
<td>If</td>
<td>happens</td>
<td>6</td>
<td>deserve</td>
<td>shit</td>
<td>32</td>
</tr>
<tr>
<td>If</td>
<td>deserve</td>
<td>7</td>
<td>deserve</td>
<td>happens</td>
<td>29</td>
</tr>
<tr>
<td>shit</td>
<td>happens</td>
<td>13</td>
<td>deserve</td>
<td>you</td>
<td>33</td>
</tr>
<tr>
<td>shit</td>
<td>deserve</td>
<td>12</td>
<td>deserve</td>
<td>it</td>
<td>22</td>
</tr>
<tr>
<td>happens</td>
<td>If</td>
<td>20</td>
<td>it</td>
<td>shit</td>
<td>4</td>
</tr>
<tr>
<td>happens</td>
<td>shit</td>
<td>22</td>
<td>it</td>
<td>happens</td>
<td>15</td>
</tr>
<tr>
<td>happens</td>
<td>you</td>
<td>27</td>
<td>it</td>
<td>deserve</td>
<td>3</td>
</tr>
<tr>
<td>happens</td>
<td>deserves</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graph-Based Dependency Parsing
Arc Features: \( f(i, j, k) \)

Features from McDonald et al.
- Identities of the words \( w_i \) and \( w_j \) and the label \( l_k \)

head=\textit{saw} & dependent=\textit{with}
Arc Features: $f(i, j, k)$

- Features from McDonald et al.
  - Part-of-speech tags of the words $w_i$ and $w_j$ and the label $l_k$

  head-pos=Verb & dependent-pos=Preposition
Arc Features: $f(i, j, k)$

- Features from McDonald et al.
  - Part-of-speech of words surrounding and between $w_i$ and $w_j$
    
    inbetween-pos=Noun
    inbetween-pos=Adverb
    dependent-pos-right=Pronoun
    head-pos-left=Noun
    ...

John saw Mary McGuire yesterday with his telescope

N  V  N  N  R  P  PR  N
Arc Features: $f(i, j, k)$

- Features from McDonald et al.
  - Number of words between $w_i$ and $w_j$, and their orientation

  $\text{arc-distance} = 3$
  $\text{arc-direction} = \text{right}$
Arc Features: $f(i, j, k)$

Label features

arc-label=PP
Arc Features: $f(i, j, k)$

- Combos of the above
  
  head-pos=Verb & dependent-pos=Preposition & arc-label=PP
  head-pos=Verb & dependent=with & arc-distance=3
  
  ... 

- No limit: any feature over arc $(i, j, k)$ or input $x$
Arc-factored Projective Parsing

- Projective dependency structures are nested
- Can use CFG like parsing algorithms – chart parsing
- Each **chart item** (triangle) represents the weight of the best tree rooted at word $h$ spanning all the words from $i$ to $j$
  - Analog in CFG parsing: items represent best tree rooted at non-terminal $NT$ spanning words $i$ to $j$
- **Goal**: Find chart item rooted at 0 spanning 0 to $n$

Base case
Length 1, $h = i = j$, has weight 1

![Diagram of a triangle with a base case](Image)
Arc-factored Projective Parsing

- All projective graphs can be written as the combination of two smaller adjacent graphs

- Inductive hypothesis – algorithm has calculated score of smaller items correctly (just like CKY)
Arc-factored Projective Parsing

- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY

Weight of new item: $\max_{l, j, k} \ w(A) \times w(B) \times w_{hh'}^k$

- Algorithm runs in $O(|L|n^5)$
- Use back-pointers to extract best parse (like CKY)
Arc-factored Projective Parsing

- $O(|L|n^5)$ is not that good
- Eisner showed how this can be reduced to $O(|L|n^3)$
  - Key: split items so that sub-roots are always on periphery
Inference in Arc-Factored Models

- Non-projective case
  - $O(|L|n^2)$ with the Chu-Liu-Edmonds MST algorithm
- Projective case
  - $O(|L|n^3)$ with the Eisner algorithm