Graph-Based Dependency Parsing

CMSC 473/673 Spring 2017
Bryan Wilkinson
Shortcomings of Transition Based Parsers

- Greedy
- Non-projectivity

Today’s lecture brought to you by
  - Marine Carput’s lecture from UMD
  - Ryan McDonald’s papers and tutorials
  - (President of ACL) Joakim Nivre’s tutorials
  - Sara Stymne’s syntax course
  - Students like you
## Non-Projectivity Statistics

<table>
<thead>
<tr>
<th>Language</th>
<th>% Non-Projective Arcs</th>
<th>% Non-Projective Sents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>0.4</td>
<td>10.1</td>
</tr>
<tr>
<td>Basque</td>
<td>2.9</td>
<td>26.2</td>
</tr>
<tr>
<td>Catalan</td>
<td>0.1</td>
<td>2.9</td>
</tr>
<tr>
<td>Chinese</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>English</td>
<td>0.3</td>
<td>6.7</td>
</tr>
<tr>
<td>Greek</td>
<td>1.1</td>
<td>20.3</td>
</tr>
<tr>
<td>Hungarian</td>
<td>2.9</td>
<td>26.4</td>
</tr>
<tr>
<td>Italian</td>
<td>0.5</td>
<td>7.4</td>
</tr>
<tr>
<td>Turkish</td>
<td>5.5</td>
<td>33.3</td>
</tr>
</tbody>
</table>

Graph Parser Intuition

- A set of dependencies for a sentence can be viewed as a directed acyclic graph (DAG).
- We can think of all possible dependencies for a sentence as a directed graph (with cycles).
  - Our task is to find a spanning tree of that graph, which will be the dependency set.
  - We are looking for the global optimum.
  - This is a recursive solution.
- Alternatively, we can view this graph as something that needs to be built from a graph that originally has no edges.
  - This is the generative paradigm.
Case 1: Start with a complete (multi-) graph

ROOT

UMBC

is a good school

ROOT

10

NSUBJ

4

ATTR

4

DET

20

AMOD

30

NSUBJPAS

1

ROOT

2

DET

10

NN

10

NN

15

DET

20

AMOD

7

NN

16

good

a

DET

4

DET

10
Determining Arc Weights

- Train a structured prediction model over all training instances of (sentence, dependency set) pairs
- Each dependency in the set is represented by a feature vector $f$
  
<table>
<thead>
<tr>
<th></th>
<th>head=is</th>
<th>dep=UMBC</th>
<th>dep=school</th>
<th>rel=NSUBJ</th>
<th>rel=ATTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>is $\rightarrow$ UMBC</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>is $\rightarrow$ school</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Goal is to find a weight matrix $w$ such that the equation below is highest for the correct dependency parse

$$\sum_{(i,j) \in \text{DEP}} w \cdot f(i, j)$$
Determining Arc Weights

- To learn $w$, for each sentence and correct parse,
  - Calculate $\sum_{(i,j) \in \text{DEP}} w \cdot f(i, j)$ for the correct parse and $k$ other parses
  - The $k$ other parses could be generated by using a less accurate parser or programmatically say different DAGs from the given connected cyclic graph
- Update $w$ so that the distance between $\sum_{(i,j) \in \text{DEP}} w \cdot f(i, j)$ for the correct parse, and $\sum_{(i,j) \in \text{DEP}} w \cdot f(i, j)$ for an incorrect parse is maximized.
Case 1: Start with a complete (multi-) graph

ROOT

UMBC

is a good school

ROOT 10

NSUBJ 4

ATTR 4

DET 20

AMOD 30

NN 16

AMOD 7

good

ROOT 2

NN 10

DET 20

NN 15

a

DET 4

DET 10

NSUBJPAS 1

ROOT 3

ROOT 4

NSUBJ 4
UMBC is a good school.
UMBC is a good school.
UMBC is a good school.
UMBC is a good school.
Chu-Liu-Edmonds Algorithm

- Chu-Liu-Edmonds is a greedy recursive algorithm to find a maximum spanning tree of a DAG
- Find the subgraph in which each node has at most one incoming edge, which is the highest scoring of all incoming edges
  - If this is a tree, you’re done
- If it is not a tree, it must have a cycle
  - Select any cycle in the subgraph
  - Contract subgraph into new pseudonode
    - Recalculate edge weights
  - Run CLU on new graph, with cycle replaced with contracted pseudonode
  - Reconstruct original edges
CLU(G):

\[ M = \{(\text{argmax}(x',x) \text{ in } E), x) \text{ for } x \text{ in } V\} \]

\[ G_M = \text{Graph}(V,M) \]

If \( G_M \) is a tree, return \( G_M \)

Else:

\[ C = \text{cycle in } G \]

\[ G_C = \text{contract}(G,C) \]

\[ y = \text{CLU}(G_C) \]

Find \((x',x)\) in \( y[E] \) and \((x'',x)\) in \( C \)

Return \( y \cup \{C - (x'',x)\} \)
Pick Highest Incoming Edge

UMBC is a good school.
Pick Highest Incoming Edge

UMBC is a good school.
Is it a tree?

ROOT

is

a

UMBC

good

school

a

DET 20

AMOD 30

NN 16

ATTR 4

NN 10

ROOT 10

ROOT 10

NN 10
UMBC is a good school.
Contract and Recalculate??

UMBC is a good school
Contract Function

- This method takes in a graph and a cycle to contract, and recalculates the edge weights.
- First we replace the cycle with a pseudonode, \( p \).
- For each node \( x \) in the original graph that is not in the cycle and that has at least one incoming node originating from within the cycle:
  - Add a new edge from \( p \) to \( x \), whose value is the maximum of all edges originating in \( p \) and going to \( x \).
- For each node \( x \) in the original graph that is not in the cycle and has at least one outgoing edge to a node in the cycle:
  - Add an new edge from \( x \) to \( p \), whose value is
    - For the node \( y \) in the cycle that maximizes
      \[
      \text{value}_x \text{of}(x,y) + \text{value}_x \text{of}(\text{parent}_x \text{of}(y),y) + \text{sum}(\text{value}_x \text{of}(\text{parent}_x \text{of}(z),z) \text{ for } z \text{ in } C)
      \]
Contract Pseudonode

Contract(G,C):
    G[V].append(p)
    sub_score = sum([(z,y) for (z,y) in C[E] for y in C[V]])
    For x in G[V]
        If x not in C[V] and len([(q,x) for q in C[V]]) > 0
            (p,x) = max[(q,x) for q in C[V]]
        If x not in C[V] and len([(x,q) for q in C[V]]) > 0
            (x,p) = max[(x,q) - (parent_of(q),q)) + sub_score for q in C[V]]
UMBC is a good school.
UMBC is a good school.
Recalculate Outgoing Edges

UMBC is a good school.
Recalculate Outgoing Edges

ROOT

is

a

UMBC

school

good

DET 4

AMOD 30

DET 20

NN 16

NN 10

NN 15

DET 20

ATTR 4

ROOT 2

ROOT 10

NSUBJ 4

ROOT 3
Recalculate Incoming Edges

ROOT

is

school

a

good

UMBC

is a good school.
Recalculate Incoming Edges

\[ \text{sub\_score} = 25 = (\text{school, UMBC}) + (\text{UMBC, school}) \]

\[ \text{max}[\text{ROOT}, \text{UMBC}] - (\text{school, UMBC}) + \text{sub\_score} \]

\[ (\text{ROOT, school}) - (\text{UMBC, school}) + \text{sub\_score} ] \]
Recalculate Incoming Edges

\[
\text{sub\_score} = 25 = (\text{school}, \text{UMBC}) + (\text{UMBC}, \text{school})
\]

\[
\text{Max}[3 - 10 + 25, 2 - 15 + 25] = \text{Max}[18, 12]
\]
Recalculate Incoming Edges

$\text{sub\_score} = 25 = (\text{school}, \text{UMBC}) + (\text{UMBC}, \text{school})$

$\text{Max} [3 - 10 + 25, 2 - 15 + 25]$

$\text{Max} [18, 12]$
Recalculate Incoming Edges

ROOT

UMBC

is a good school

ROOT 10

NSUBJ 4

ATTR 6

DET 20

AMOD 30

ROOT 18

NN 10

NN 15

DET 4

NN 16

good

a
UMBC is a good school.
UMBC is a good school.
UMBC is a good school.
Is it a Tree? NO :(

UMBC is a good school
Find a Cycle....
UMBC is a good school.
Generative Graph Parsers

- Rather than start from a graph that contains all possible dependency graphs, build one starting with single nodes
  - Just like a constituency parser
Generative Graph Parsers

- Rather than start from a graph that contains all possible dependency graphs, build one starting with single nodes
  - Just like a constituency parser

```
NNP  VB  A  JJ  NN
UMBC  is  a  good  school
```

```
ROOT
UMBC
is
a
school
good
```
Generative Graph Parsers

- Rather than start from a graph that contains all possible dependency graphs, build one starting with single nodes
  - Just like a constituency parser

```
NNP   VB   A    JJ   NN
UMBC  is   a   good school

ROOT

good
  AMOD

is
  NSUBJ

UMBC

a
```
Generative Graph Parsers

- Rather than start from a graph that contains all possible dependency graphs, build one starting with single nodes
  - Just like a constituency parser
Generative Graph Parsers

- Rather than start from a graph that contains all possible dependency graphs, build one starting with single nodes
  - Just like a constituency parser

```
UMBC

is

a
good

school
```

```
ROOT

is

ATTR

school
```

```
DET

AMOD

NSUBJ

ROOT
```

```
NP

NP

VP
```

```
NNP

VB

A

JJ

NN

UMBC

is

a
good

school
```
Generative Graph Parsers

- Rather than start from a graph that contains all possible dependency graphs, build one starting with single nodes
  - Just like a constituency parser

```
NNP  VB  A  JJ  NN
|     |     |    |    |
UMBC is a good school
```

```
S

VP

NP

NP

ROOT

good

is

school

is

a
good

school

ROOT

AMOD

ATTR

DET

NSUBJ

UMBC

a
```
Let's look at the dependency graph as it is constructed side by side with the arc representation.
• Let's look at the dependency graph as it is constructed side by side with the arc representation.

UMBC is a good school
Let's look at the dependency graph as it is constructed side by side with the arc representation.
Another look

- Let's look at the dependency graph as it is constructed side by side with the arc representation.

```
ROOT
UMBC
is
a
good
school
```

```
ROOT
good
```

```
is
ROOT
UMBC
NSUBJ
AMOD
ATTR
DET
school
```
Dynamic Programming

- **Modified CKY**
  - Collins $O(n^5)$
  - Eisner $O(n^3)$
- **Older than MST parsing, and slower BUT**
  - Easier to extend into really accurate parsers
- **The general idea is to connect spans instead of subtrees**
  - A span is a substring of the sentence that only has dependencies among itself
  - Only the end words of the span can used to combine two spans
Eisner’s Algorithm (1996)

- For completeness, this is how you modify CKY to use for dependency parses
- This can get pretty confusing
  - The very best explanation I have ever seen is [http://curtis.ml.cmu.edu/w/courses/index.php/Eisner_algorithm](http://curtis.ml.cmu.edu/w/courses/index.php/Eisner_algorithm)
- The pseudocode from that site is

```plaintext
for i := 1 to n
    s := the item for w_{i,i+1} produced by SEND
    Discover(i, i + 1, s)
    Discover(i, i + 1, OPT-LINK-L(s))
    Discover(i, i + 1, OPT-LINK-R(s))
for width := 2 to n
    for i := 1 to (n − 1) − width
        k := i + width
        for j := i + 1 to k − 1
            foreach simple item s in \(C_{i,j}^l\)
                foreach item s2 in \(C_{i,k}^r\) such that \(\text{COMBINE}(s_1, s_2)\) is defined
                    s := \(\text{COMBINE}(s_1, s_2)\)
                    Discover(i, j, s)
                    if OPT-LINK-L(s) and OPT-LINK-R(s) are defined
                        Discover(i, j, OPT-LINK-L(s))
                        Discover(i, j, OPT-LINK-R(s))
            foreach s in \(C_{j,k}^{l+1}\)
                if ACCEPT(s) is defined
                    return accept
            return reject
```
In this generative model, each potential connection has access to all existing connections.

When deciding where to connect *is*, you can look at things like, which words will become my grandchildren (*a, good*) and does this make sense?

- Makes very powerful and more accurate parsers
  - Makes very complex and detailed algorithms

> UMBC is a good school
Alternatives to Graph-Based Parsing for Non-Projective Sentences

- Graph Based Parsing can get complex and have long run times
- Wouldn’t it be nice if we could do something simpler?
- Post-processing
  - Run a shift-reduce parser and then look for any dependencies in the tree which should actually be non-projective
- Limit non-productiveness
  - If you assume only one crossing edge, you can tailor the algorithm to be a little smoother and less complex
- Personal Observation
  - Innovation seems to happen in graph-based parsers, then when certain ones prove useful, e.g. grandparents and siblings, scientist figure out how to make them work in transition based parsers.
MidTerm Review

- 30 minutes here to review
- I’ll be in my lab ITE 361, from 8PM-10PM, Wednesday March 8th, to answer any questions