Graph-based Dependency Parsing

Ryan McDonald
Google Research
ryanmcd@google.com
Graph-based Dependency Parsing

Ryan McDonald
Google Research
ryanmcd@google.com
Mr Tomash will remain as a director emeritus.
Definitions

\[ L = \{l_1, l_2, \ldots, l_m\} \]  
\[ X = x_0x_1 \ldots x_n \]  
\[ Y \]  

Arc label set
Input sentence
Dependency Graph/Tree
Definitions

\[ L = \{l_1, l_2, \ldots, l_m\} \] 
Arc label set

\[ X = x_0x_1 \ldots x_n \] 
Input sentence

\[ Y \] 
Dependency Graph/Tree
Definitions

\[ L = \{l_1, l_2, \ldots, l_m\} \quad \text{Arc label set} \]

\[ X = x_0 x_1 \ldots x_n \quad \text{Input sentence} \]

\[ Y \quad \text{Dependency Graph/Tree} \]

\[ (i, j, k) \in Y \quad \text{indicates} \quad x_i \xrightarrow{l_k} x_j \]
Graph-based Parsing

Factor the weight/score graphs by subgraphs

$$w(Y) = \prod_{\tau \in Y} w_{\tau}$$

$\tau$ is from a set of subgraphs of interest, e.g., arcs, adjacent arcs

Product vs. Sum:

$$Y = \arg \max_Y \prod_{\tau \in Y} w_{\tau} = \arg \max_Y \sum_{\tau \in Y} \log w_{\tau}$$
Arc-factored Graph-based Parsing

root

saw

John → Mary

Mary → John
Arc-factored Graph-based Parsing

Learn to weight arcs

$$w(Y) = \prod_{a \in Y} w_a$$
Arc-factored Graph-based Parsing

Learn to weight arcs

\[ w(Y) = \prod_{a \in Y} w_a \]

Inference/Parsing/Argmax

\[ Y = \arg \max_Y \prod_{a \in Y} w_a \]
Arc-factored Graph-based Parsing

Learn to weight arcs

\[ w(Y) = \prod_{a \in Y} w_a \]

\[ Y = \arg \max_Y \prod_{a \in Y} w_a \]
Arc-factored Projective Parsing

$W[i][j][h] =$ weight of best tree spanning words $i$ to $j$ rooted at word $h$

$w(A) \times w(B) \times w^{k}_{hh'}$

max over $k, l, h'$
Arc-factored Projective Parsing

$W[i][j][h] = \text{weight of best tree spanning words } i \text{ to } j \text{ rooted at word } h$

**Eisner '96**

$O(|L|n^5) \rightarrow O(n^3 + |L|n^2)$

![Diagram showing the calculation of $W[i][j][h]$](image)
Arc-factored Non-projective Parsing

- Non-projective Parsing (McDonald et al ’05)
- Inference: $O(|L|n^2)$ with Chu-Liu-Edmonds MST alg
- Greedy-Recursive algorithm

John saw Mary

Spanning trees
Valid dependency graphs
Arc-factored Non-projective Parsing

- Non-projective Parsing (McDonald et al ’05)
- Inference: $O(|L|n^2)$ with Chu-Liu-Edmonds MST alg
- Greedy-Recursive algorithm

We win with non-projective algorithms! ... err ...

Spanning trees

Valid dependency graphs
Arc-factored Non-projective Parsing

- Non-projective Parsing (McDonald et al '05)
- Inference: $O(|L|n^2)$ with Chu-Liu-Edmonds MST alg
- Greedy-Recursive algorithm

We win with non-projective algorithms! ... err ...
• Arc-factored models can be powerful
• But does not model linguistic reality
• Syntax is not context independent
Beyond Arc-factored Models

• Arc-factored models can be powerful
• But does not model linguistic reality
• Syntax is not context independent

Arity

• Arity of a word = # of modifiers in graph
• Model arity through preference parameters
• Arc-factored models can be powerful
• But does not model linguistic reality
• Syntax is not context independent

Arity
• Arity of a word = # of modifiers in graph
• Model arity through preference parameters

Beyond Arc-factored Models

Arity

Markovization

Vertical/Horizontal
Adjacent arcs
Projective -- Easy

\[ W[i][j][h][a] = \text{weight of best tree spanning words} \]
\[ i \text{ to } j \text{ rooted at word } h \text{ with arity } a \]

**Arity terms**

\[
\frac{w(A)}{w_{a-1}^h} \times w(B) \times w^{k}_{hh'} \times w^{a}_{h}
\]

max over \( k, l, h' \)
Non-projective -- Hard

- McDonald and Satta ‘07
- Arity (even just modified/not-modified) is NP-hard
- Markovization is NP-hard
- Can basically generalize to any non-local info
- Generalizes Nehaus and Boker ‘97

Arc-factored: non-projective “easier”
Beyond arc-factored: non-projective “harder”
Non-projective Solutions

- In all cases we augment \( w(Y) \)

\[
  w(Y) = \prod_{(i,j,k)} \ w_{ij}^k \times \beta
\]

- Calculate \( w(Y) \) using:
  - Approximations (Jason’s talk!)
  - Exact ILP methods
  - Chart-parsing Algorithms
  - Re-ranking
  - MCMC
Annealing Approximations
(McDonald & Pereira 06)

• Start with initial guess
• Make small changes to increase $w(Y)$

$$w(Y) = \prod_{(i,j,k)} w_{ij}^k \times \beta$$
Annealing Approximations
(McDonald & Pereira 06)

• Start with initial guess
• Make small changes to increase $w(Y)$

$w(Y) = \prod_{(i,j,k)} w_{ij}^k \times \beta$

Initial guess: $\arg \max_Y \prod_{(i,j,k)} w_{ij}^k$

Arc Factored
Annealing Approximations
(McDonald & Pereira 06)

- Start with initial guess
- Make small changes to increase $w(Y)$

Initial guess: 
$$\arg\max_Y \prod_{(i,j,k)} w_{ij}^k$$

Arc Factored

Until convergence
- Find arc change to maximize
- Make the change to guess

$$w(Y) = \prod_{(i,j,k)} w_{ij}^k \times \beta$$
Annealing Approximations
(McDonald & Pereira 06)

- Start with initial guess
- Make small changes to increase \( w(Y) \)

\[
\begin{align*}
    w(Y) &= \prod_{(i,j,k)} w_{ij}^k \times \beta \\
    \text{Initial guess: } \arg \max_Y \prod_{(i,j,k)} w_{ij}^k
\end{align*}
\]

Until convergence
- Find arc change to maximize
- Make the change to guess

Good in practice, but suffers from local maxima
Integer Linear Programming (ILP)
(Riedel and Clarke 06, Kubler et al 09, Martins, Smith and Xing 09)

- An ILP is an optimization problem with:
  - A linear objective function
  - A set of linear constraints
- ILPs are NP-hard in worst-case, but well understood w/ fast algorithms in practice
- Dependency parsing can be cast as an ILP

Note: we will work in the log space

\[ Y = \arg \max_{Y \in Y(Gx)} \sum_{(i,j,k)} \log w_{ij}^k \]
Define integer variables:

\[ a_{ij}^k \in \{0, 1\} \]

\[ a_{ij}^k = 1 \text{ iff } (i, j, k) \in Y \]

\[ b_{ij} \in \{0, 1\} \]

\[ b_{ij} = 1 \text{ iff } x_i \rightarrow \ldots \rightarrow x_j \in Y \]
Arc-Factored Dependency Parsing as an ILP
(from Kubler, McDonald and Nivre 2009)

\[
\max_\mathbf{a} \sum_{i,j,k} a_{ij}^k \times \log w_{ij}^k
\]

such that:

\[
\sum_{i,k} a_{i0}^k = 0 \quad \forall j : \sum_{i,k} a_{ij}^k = 1
\]

\[
\forall i, j, k : b_{ij} - a_{ij}^k \geq 0
\]

\[
\forall i, j, k : 2b_{ik} - b_{ij} - b_{jk} \geq -1
\]

\[
\forall i : b_{ii} = 0
\]

Constrain arc assignments to produce a tree
Arc-Factored Dependency Parsing as an ILP
(from Kubler, McDonald and Nivre 2009)

\[
\max_a \sum_{i,j,k} a_{i,j}^k \times \log w_{i,j}^k
\]

Can add non-local constraints & preference parameters
Riedel & Clarke ’06, Martins et al. 09

\[
\forall i, j, k : b_{ij} - a_{i,j}^k \geq 0
\]

\[
\forall i, j, k : 2b_{ik} - b_{ij} - b_{jk} \geq -1
\]

\[
\forall i : b_{ii} = 0
\]

Constrain arc assignments to produce a tree
Dynamic Prog/Chart-based methods
Dynamic Prog/Chart-based methods

- **Question**: are there efficient non-projective chart parsing algorithms for unrestricted trees?
- Most likely not: we could just augment them to get tractable non-local non-projective models
Dynamic Prog/Chart-based methods

- **Question**: are there efficient non-projective chart parsing algorithms for unrestricted trees?
  - Most likely not: we could just augment them to get tractable non-local non-projective models
  - Gomez-Rodriguez et al. 09, Kuhlmann 09
  - For well-nested dependency trees of gap-degree 1
    - Kuhlmann & Nivre: Accounts for >> 99% of trees
    - $O(n^7)$ deductive/chart-parsing algorithms
Dynamic Prog/Chart-based methods

• **Question**: are there efficient non-projective chart parsing algorithms for unrestricted trees?

• Most likely not: we could just augment them to get tractable non-local non-projective models

• Gomez-Rodriguez et al. 09, Kuhlmann 09

• For well-nested dependency trees of gap-degree 1
  - Kuhlmann & Nivre: Accounts for >> 99% of trees
  - $O(n^7)$ deductive/chart-parsing algorithms

Chart-parsing == easy to extend beyond arc-factored assumptions
What is next?

- Getting back to grammars?
- Non-projective unsupervised parsing?
- Efficiency?
Getting Back to Grammars

- Almost all research has been grammar-less
- All possible structures permissible
- Just learn to discriminate good from bad
- Unlike SOTA phrase-based methods
- All explicitly use (derived) grammar
Getting Back to Grammars
Getting Back to Grammars

- Projective == CF Dependency Grammars
- Gaifman (65), Eisner & Blatz (07), Johnson (07)
Getting Back to Grammars

- Projective == CF Dependency Grammars
- Gaifman (65), Eisner & Blatz (07), Johnson (07)
- Mildly context sensitive dependency grammars
- Restricted chart parsing for well-nested/gap-degree 1
- Bodirsky et al. (05): capture LTAG derivations
Getting Back to Grammars

• Projective == CF Dependency Grammars
  • Gaifman (65), Eisner & Blatz (07), Johnson (07)
• Mildly context sensitive dependency grammars
  • Restricted chart parsing for well-nested/gap-degree 1
  • Bodirsky et al. (05): capture LTAG derivations
• ILP == Constraint Dependency Grammars (Maruyama 1990)
  • Both just put constraints on output
  • CDG constraints can be added to ILP (hard/soft)
  • Annealing algs == repair algs in CDGs
Getting Back to Grammars

• Projective == CF Dependency Grammars
• Gaifman (65), Eisner & Blatz (07), Johnson (07)
• Mildly context sensitive dependency grammars
• Restricted chart parsing for well-nested/gap-degree 1
• Bodirsky et al. (05): capture LTAG derivations

ILP == Constraint Dependency Grammars (Maruyama 1990)

• Both just put constraints on output
• CDG constraints can be added to ILP (hard/soft)
• Annealing algs == repair algs in CDGs

Questions
1. Can we flush out the connections further?
2. Can we use grammars to improve accuracy and parsing speeds?
Non-projective Unsupervised Parsing

- McDonald and Satta 07
  - Dependency model w/o valence (arity) is tractable
  - Not true w/ valence
- Klein & Manning 04, Smith 06, Headden et al. 09
  - All projective
  - Valence++ required for good performance
Non-projective Unsupervised Parsing

- McDonald and Satta 07
  - Dependency model w/o valence (arity) is tractable
  - Not true w/ valence
- Klein & Manning 04, Smith 06, Headden et al. 09
  - All projective
  - Valence++ required for good performance

Non-projective Unsupervised Systems?
### Efficiency / Resources

<table>
<thead>
<tr>
<th>Swedish</th>
<th>O(nL)</th>
<th>O(n^3 + nL)</th>
<th>O(n^3L^2)</th>
<th>O(n^3L^2)</th>
<th>O(n^2kl^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malt Joint</td>
<td>MST pipeline</td>
<td>MST joint</td>
<td>MST Joint Feat Hash</td>
<td>MST Joint Feat Hash Coarse to Fine</td>
<td></td>
</tr>
<tr>
<td>LAS</td>
<td>84.6</td>
<td>82.0</td>
<td>83.9</td>
<td>84.3</td>
<td>84.1</td>
</tr>
<tr>
<td>Parse time</td>
<td>-</td>
<td>1.00</td>
<td>~125.00</td>
<td>~30.00</td>
<td>4.50</td>
</tr>
<tr>
<td>Model size</td>
<td>-</td>
<td>88 Mb</td>
<td>200 Mb</td>
<td>11 Mb</td>
<td>15 Mb</td>
</tr>
<tr>
<td># features</td>
<td>-</td>
<td>16 M</td>
<td>30 M</td>
<td>30 M</td>
<td>30 M</td>
</tr>
</tbody>
</table>

Pretty good, but still not there! -- A*?, More pruning?
Summary
Summary

• Where we’ve been
  • Arc-factored: Eisner / MST
  • Beyond arc-factored: NP-hard
    • Approximations
    • ILP
  • Chart-parsing on defined subset
Summary

• Where we’ve been
  • Arc-factored: Eisner / MST
  • Beyond arc-factored: NP-hard
    • Approximations
    • ILP
    • Chart-parsing on defined subset

• What’s next
  • The return of grammars?
  • Non-projective unsupervised parsing
  • Making models practical on web-scale