Fast and Accurate Arc Filtering for Dependency Parsing

Shane Bergsma
University of Alberta

Colin Cherry
National Research Council Canada

COLING 2010
Overview

• **Goal:**
  – speed-up graph-based dependency parsing by removing implausible head-mod pairs (arcs)

• **Method:**
  – Classifier says which nodes are leaves/roots/etc.

• **Results:**
  – remove 78% of arcs, keep 99.5% of true ones
  – speeds up 2 state-of-the-art dependency parsers
Dependency Parsing (DP)

Bob ate the pizza with his fork

NN  VBD  DT  NN  IN  POS  NN

Bob ate the pizza with his fork

NN  VBD  DT  NN  IN  POS  NN
Motivation

• DP could be used to parse the web
  – Requirements: Top Accuracy and Fast
  – This is not a contradiction [Bohnet, 25 minutes ago]

• Graph-based vs. Transition-based
  – Transition-based are fast, but make orthogonal errors to graph-based
  – To combine graph-based and transition-based parsers in practice, both must be fast
Graph-Based DP

• Tree score decomposes into scores over arcs ([h,m] pairs):

\[
\text{parse}(s) = \arg\max_{t \in s} \sum_{[h,m] \in t} \bar{w} \cdot \bar{f}(h, m, s)
\]

• argmax computed using MST \(O(n^3)\) or [Eisner, 1996]’s algorithm \(O(n^2)\)

• But what if \(|\bar{f}| >> n\)? (\(\bar{f}\) for all \(O(n^2)\) arcs)
Features

- $\bar{f} = \{\text{head=ate, mod=with, ate-with, ate-IN, VBD-with, VBD-IN, ate-the-with, ate-pizza-with, ate-DT-with, ate-NN-with, VBD-the-with, VBD-the-IN, VBD-pizza-with, VBD-pizza-IN, ate-pizza-IN, ate-the-IN, VBD-DT-with, VBD-DT-IN, VBD-NN-with, VBD-NN-IN, ate-DT-IN...}\}$

- Also conjoined with direction, distance!
Features

• Do we really need to generate all the features?
• DT isn’t usually a head
• POS usually has its head on the right
Filter Framework

- 3 stages that prune arcs before parsing
- Each stage is a supervised SVM classifier
  - Extract training decisions from Treebank data
- SVM biased in training to have very high precision (so good arcs don’t get filtered)
  - Optimize $j$ parameter (per-class cost factor) and $C$ parameter (regularization) in LIBLINEAR

[Fan et al., 2008]
## Rules

<table>
<thead>
<tr>
<th>not a $h$</th>
<th>&quot;&quot;, . ;</th>
<th>CC PRP$ PRP EX -RRB- -LRB-</th>
</tr>
</thead>
<tbody>
<tr>
<td>no $* \leftarrow m$</td>
<td>EX LS POS PRP$</td>
<td></td>
</tr>
<tr>
<td>no $m \rightarrow *$</td>
<td>. RP</td>
<td></td>
</tr>
<tr>
<td>not a root</td>
<td>, DT</td>
<td></td>
</tr>
<tr>
<td>no $h \leftarrow m$</td>
<td>DT$\rightarrow${DT,JJ,NN,NNP,NNS,.}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CD$\rightarrow$CD NN$\rightarrow${DT,NNP}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NNP$\rightarrow${DT,NN,NNS}</td>
<td></td>
</tr>
<tr>
<td>no $m \rightarrow h$</td>
<td>{DT,IN,JJ,NN,NNP}$\rightarrow$DT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NNP$\rightarrow$IN IN$\rightarrow$JJ</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Learned rules for filtering dependency arcs using PoS tags. The rules filter 25% of possible arcs while recovering 99.9% of true links.
Linear

• For each word in a sentence \([O(n)]\) decide:
  1. It is *not* a head (i.e., it’s a leaf)
  2. Its head is on the left/right
  3. Its head is within 5 words on the left/right
  4. Its head is immediately to the left/right
  5. It is the root

• Filter arcs accordingly

• E.g., if \(^{i}\text{th}\) word is the root, no crossing arcs
Linear Features

• Have same feature vector for each of 8 linear classifiers (for speed)
  – Build feature vector once, multiply with 8 different weights

\[ f_i = \{\text{tag/word}_i, \text{tag/word}_{i+1}, \text{tag/word}_{i-1}, \ldots, \text{tag/word}_i^{\text{tag}_{i+2}}, \text{tag/word}_i^{\text{tag}_{i+3}}, \ldots\} \]
Search for Best Linear Filters

- Could optimize each linear classifier separately on dev data
- Better: jointly optimize linear filter hyperparameters ($C, j$ for LibLinear)
- Ongoing work: jointly optimize linear filter parameters (weights) as a single latent SVM optimization
Quadratic

• Decide yes/no to filtering each arc
  – Theoretically, $O(n^2)$ just like the parser’s arc-scoring function (so why bother?)

• Can be a *light* preprocessing step, use less features
Quadratic Features

• With a feature for every *between-tag*, feature extraction actually $O(n^3)$ [Galley & Manning, 2009]

• Us: only between-tags within $\pm 5$ of head or mod

• Plus: a few real-valued tags+distance features
Related Work

• Vine parsing [Eisner & Smith, 2005]
  – Hard cap on arc length
  – Can depend on tags being linked (Tag-Vine)
  – Root + L/R head-marking [Søgaard and Kuhn, 2009]

• CFG cell classification for constituency parsers [Roark & Hollingshead, 2008]

• Coarse-to-fine Parsing
  – Used in DP [Carreras et al., 2008]
Experiments

- WSJ portion of Treebank, standard train/dev/test sets
- Stanford tagger
## Test Set Performance

<table>
<thead>
<tr>
<th>Filter</th>
<th>Coverage</th>
<th>Reduction</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag-Vine</td>
<td>99.6%</td>
<td>44%</td>
<td>2.9s</td>
</tr>
<tr>
<td>Rules</td>
<td>99.9%</td>
<td>26%</td>
<td>1.3s</td>
</tr>
<tr>
<td>Linear</td>
<td>99.7%</td>
<td>54%</td>
<td>7.3s</td>
</tr>
<tr>
<td>Quadratic</td>
<td>99.5%</td>
<td>78%</td>
<td>16.1s</td>
</tr>
</tbody>
</table>
Analysis

• Which linear filters are most important?
  – See paper for details
Incorporate filters into parsers

- McDonald et al.'s MST parser
  - Trained using 5-best MIRA
  - First and second-order (MST-1, MST-2)

- In-house “DepPerceptron”
  - Trained via averaged perceptron [Collins, 2002]
  - Features mix of [McDonald et al. 2005] and [Koo et al. 2008]
### Big Parsing Speed-Ups

<table>
<thead>
<tr>
<th>Filter</th>
<th>Cost</th>
<th>DepPercep-1</th>
<th>DepPercep-2</th>
<th>MST-1</th>
<th>MST-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc.</td>
<td>Time</td>
<td>Acc.</td>
<td>Time</td>
</tr>
<tr>
<td>None</td>
<td>+0</td>
<td>91.8</td>
<td>348</td>
<td>92.5</td>
<td>832</td>
</tr>
<tr>
<td>Vine</td>
<td>+3</td>
<td>91.7</td>
<td>192</td>
<td>92.3</td>
<td>407</td>
</tr>
<tr>
<td>Rules</td>
<td>+1</td>
<td>91.7</td>
<td>264</td>
<td>92.4</td>
<td>609</td>
</tr>
<tr>
<td>Linear</td>
<td>+7</td>
<td>91.7</td>
<td>168</td>
<td>92.4</td>
<td>334</td>
</tr>
<tr>
<td>Quad.</td>
<td>+16</td>
<td>91.7</td>
<td>79</td>
<td>92.3</td>
<td>125</td>
</tr>
</tbody>
</table>

- MST-2 goes from 12 to 23 sentences/s
Conclusion

• Linear and quadratic-time filtering leads to speed-ups in even carefully-optimized dependency parsers

• Negligible loss in accuracy

• Download today: http://code.google.com/p/arcfilter/
Thanks

- Ryan McDonald & MST coders, Google
- Terry Koo, MIT (now Google)
- Dekang Lin, Google
- Peter Turney, NRC