Faster Dependency Parsing, More Accurate Unsupervised Parsing

Shay Cohen
ILCC, School of Informatics
University of Edinburgh

May 11, 2022
Did Aristotle Have a Cellphone?

Leslie Valiant (2021)
But... ”There is no data like more data”

Maybe... There is no model like a bigger model?
This Talk

Part 1:
Show how to make dependency parsers faster

Part 2:
Show how to make unsupervised parsers more accurate
This Talk

Part 1:
Show how to make dependency parsers faster

Part 2:
Show how to make unsupervised parsers more accurate
From the Stanza parser:

- Parsers are not perfect yet!
- The Universal Dependencies Project aims at finding a common dependency formalism for many languages.
Why Dependency Parsing?

- It gives relations directly between words, no “deep” structure which is sometimes redundant
- This also makes it a better theory for free-word order languages
- Flexible at modelling non-projectivity
In languages with *free word order*, phrase structure (constituency) grammars don’t make as much sense.

- E.g., we would need both $S \rightarrow NP \ VP$ and $S \rightarrow VP \ NP$, etc. Not very informative about what’s really going on.

In contrast, the dependency relations stay constant (in Russian, “Sasha gave a book to the girl”):
STAGE 1: Start with a complete graph, all edges connected to each other with some weight.
Graph Parsing for Dependency Parsing

- **STAGE 1:** Start with a complete graph, all edges connected to each other with some weight.

- **STAGE 2:** Run a maximum spanning tree algorithm to find the highest scoring tree.
Where is Time Spent in Parsing?

Time spent on STAGE 1 and STAGE 2:

- Calculated from the Stanza parser
- Most time is spent on inference
The Universal Dependency Project (Nivre et al., 2018) documentation states that (Zmigrod et al. 2020):

There should be just one node with the root dependency relation in every tree [...] 

https://universaldependencies.org/u/dep/root.html

We ask: Can we optimally decode a dependency tree with a single-root constraint? (Stanojević and Cohen, 2021)

Answer: Yes. Run asymptotically in quadratic time with an empirical low constant
## Single-Root Dependency Parsing Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Appears in</th>
<th>Current implementation worst-case</th>
<th>Claimed worst-case dense graph</th>
<th>Average-case dense graph</th>
<th>Claimed worst-case sparse graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gabow-Tarjan</td>
<td>Gabow &amp; Tarjan (1984) Zmigrod et al. (2020)</td>
<td>$O(n^2 \log n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(m \log n)$</td>
</tr>
<tr>
<td>Naïve</td>
<td>mentioned in Zmigrod et al. (2020)</td>
<td>n/a</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(mn + n^2 \log n)$</td>
</tr>
<tr>
<td>Preselection</td>
<td>code of some parsers (undocumented)</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(mn + n^2 \log n)$</td>
</tr>
<tr>
<td>Reweighting</td>
<td>this work</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(m + n \log n)$</td>
</tr>
</tbody>
</table>

In the above, we might use as a subroutine an unconstrained parsing algorithm.

For example: $O(n^2)$, using our implementation of the Chu-Liu-Edmonds algorithm (CLE) based on data structures from Tarjan (1977).
# Single-Root Dependency Parsing Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Appeared in</th>
<th>Current Implementation Worst-Case</th>
<th>Claimed Worst-Case Dense Graph</th>
<th>Average-Case Dense Graph</th>
<th>Claimed Worst-Case Sparse Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gabow-Tarjan</td>
<td>Gabow &amp; Tarjan (1984) Zmigrod et al. (2020)</td>
<td>$O(n^2 \log n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(m \log n)$</td>
</tr>
<tr>
<td>Naïve</td>
<td>mentioned in Zmigrod et al. (2020)</td>
<td>n/a</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(mn + n^2 \log n)$</td>
</tr>
<tr>
<td>Preselection</td>
<td>code of some parsers (undocumented)</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(mn + n^2 \log n)$</td>
</tr>
<tr>
<td>Reweighting</td>
<td>this work</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(m + n \log n)$</td>
</tr>
</tbody>
</table>

In the above, we might use as a subroutine an unconstrained parsing algorithm.

For example: $O(n^2)$, using our implementation of the Chu-Liu-Edmonds algorithm (CLE) based on data structures from Tarjan (1977).
In the above, we might use as a subroutine an unconstrained parsing algorithm

For example: $\mathcal{O}(n^2)$, using our implementation of the Chu-Liu-Edmonds algorithm (CLE) based on data structures from Tarjan (1977)
## Single-Root Dependency Parsing Algorithms

<table>
<thead>
<tr>
<th>algorithm</th>
<th>appeared in</th>
<th>current implementation worst-case</th>
<th>claimed worst-case dense graph</th>
<th>average-case dense graph</th>
<th>claimed worst-case sparse graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gabow-Tarjan</td>
<td>Gabow &amp; Tarjan (1984) Zmigrod et al. (2020)</td>
<td>$O(n^2 \log n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(m \log n)$</td>
</tr>
<tr>
<td>Naïve</td>
<td>mentioned in Zmigrod et al. (2020)</td>
<td>n/a</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(mn + n^2 \log n)$</td>
</tr>
<tr>
<td>Preselection</td>
<td>code of some parsers (undocumented)</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(mn + n^2 \log n)$</td>
</tr>
<tr>
<td>Reweighting</td>
<td>this work</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(m + n \log n)$</td>
</tr>
</tbody>
</table>

In the above, we might use as a subroutine an unconstrained parsing algorithm.

For example: $O(n^2)$, using our implementation of the Chu-Liu-Edmonds algorithm (CLE) based on data structures from Tarjan (1977).
In the above, we might use as a subroutine an unconstrained parsing algorithm

For example: $\mathcal{O}(n^2)$, using our implementation of the Chu-Liu-Edmonds algorithm (CLE) based on data structures from Tarjan (1977)
Consider the complete graph we start with our inference. If we subtract $c$ from all edges $ROOT \rightarrow w_i$:

- We subtract from all trees the weight $k \cdot c$ where $k$ is the number of $ROOT$ edges
- We do not change the weight order of trees with the same number of $ROOT$ edges
- We may change weight order of trees with different number of $ROOT$ edges

**Question:** Can we find $c$ such that we make the best single-root tree come at the top compared to all other multiple root trees?
Question: Can we find \( c \) such that we make the best single-root tree come at the top compared to all other multiple root trees?

Answer: Yes! Choose \( c^* = 1 + n(\max_e w(e) - \min_e w(e)) \) where \( w(e) \) is the weight of edge \( e \) in the complete graph.

The Reweighting Algorithm:

- Subtract \( c^* \) from all \( \text{ROOT} \rightarrow * \) edges in the initial complete graph.
- Run an unconstrained tree inference algorithm on the new complete graph.
Side Note: the ArcMax Trick

Zhang et al. (2017) show that choosing for each word the highest scoring edge going into that word as parent often gives a valid tree.

Use this as a trick: (a) check if this gives tree (this tree is optimal), if so, don't run any costly inference – we will go back to that!
Experiment 1A: Unconstrained Inference Speed

With trained weights (unconstrained):

- The CLE algorithm includes an ArcMax-like step in the beginning (by design)
- Tarjan with ArcMax catches up...
Experiment 1B: Unconstrained Inference Speed

With random weights (unconstrained):

- ArcMax-like step no longer works, so Tarjan shines
Experiment 2: Single-Root Inference Speed without ArcMax

(a) Trained English input

(b) Random input
Experiment 3: Single-Root Inference Speed with ArcMax

- Random weights not shown - same as without ArcMax
Summary of Part 1

To get optimal complexity for single root, just use:

```python
def reweighting(scores, mst_func):
    scores2 = np.where(np.isinf(scores), np.nan, scores)
    n = scores.shape[0] - 1  # number of words
    scores[:, 0] -= 1 + n*(np.nanmax(scores2) - np.nanmin(scores2))
    return mst_func(scores)
```

Our recommendation for optimal single-root dependency parsing (regardless of learning): **ArcMax + Reweighting + Tarjan**
This Talk

Part 1:
Show how to make dependency parsers faster

Part 2:
Show how to make unsupervised parsers more accurate
Unsupervised Parsing
L-PCFGs, the Supervised Case

At node **VP**:  

Outside tree $o =$  

```
S
 /     
|      |
NP VP  
|      |
D N V P
|      |
the dog saw him
```

Inside tree $t =$  

```
S
 |     
NP VP
 |     
D N
 |   |
the dog
V P
 | |
saw him
```

Conditionally independent given the label and the connecting nonterminal

$$p(o, t|VP) = p(o|VP) \times p(t|VP)$$
• The yield of the orange part is “inside string”

• The yield of the blue part is “outside string”

• We will bootstrap a classifier to predict if a node dominates a pair of (inside, outside) strings
Co-training (Yarowsky, 1995; Blum and Mitchell, 1998)

Roughly! Repeats the following steps (given unlabeled data $U$ and labeled data $L$):

- Train a classifier with one “view” on $L$
- Train a classifier with another “view” on $L$
- Take some (confident) predictions on $U$ from both classifiers and add to $L$

Co-training works best when the two views are conditionally independent given the label
And In Our Case...

This leads to:

- Take all sentences, and break them all possible ways into inside/outside strings
- Get neural representations for each pair \((i, o)\) as two “views”
- Take a subset of these \(L\), and label them with some heuristics - the label is whether \((i, o)\) has a node that connects them
- Do co-training
What would be the seed dataset to start the co-training process?

- All \((\text{start}, \text{end})\) spans for a given sentence have label 1 (are constituents)
- All \((\text{start}, \text{end} - i)\) for \(i = 1, 2, \ldots, 6\) have label 0
- Another simple heuristic that relies on casing
How Do We Use These Labels?

Let

- $p_i(e)$ be the confidence of the classifier for an inside of span $e$
- $p_o(e)$ be the confidence of the classifier for an outside of span $e$

Then, for three different types of scores:

- $score(e) = p_i(e)$
- $score(e) = p_o(e)$
- $score(e) = p_i(e) \cdot p_o(e)$

find the tree $t^* = \arg\max_t \sum_{e \in t} score(e)$

Also referred to as span decoding
● Results with the inside score
Results: Penn Treebank (English)

- Results with self-training of the inside score
• Results with co-training for inside × outside scores
Results: Chinese Treebank

- Left Branching (LB)
- Random Trees
- Right Branching (RB)
- PRPN (Shen et al., 2018)
- ON (Shen et al., 2019)
- Neural PCFG (Kim et al., 2019a)
- Compound PCFG (Kim et al., 2019a)
- Ours

Legend:
- UP Models
- inside
- inside w/ self-training
- inside-outside w/ co-training
Results: Korean Treebank

Unlabeled Sentence-level F1

- Left Branching (LB)
- Right Branching (RB)
- PRPN (Shen et al., 2018)
- URNNG (Kim et al., 2019b)
- DIORA (Drozdov et al., 2019)
- DIORA-all (Hong et al., 2020)
- Ours

KTB-Full

KTB-10

Legend:
- UP Models
- inside
- inside w/ self-training
- inside-outside w/ co-training
Unsupervised constituency parsing can be viewed as a co-training problem.

See the paper for more examples, score functions and error analysis!
Conclusion

- Language is manifested through symbols. Computational systems in general are often symbolic in nature.

- Its intermediate representation, however - can be continuous or symbolic.

- Symbolic: interpretable; Continuous: have a gradient.

- Both have their role. Both can co-exist.
Thank You!

Any questions?

Collaborators

Milos Stanojević

Nickil Maveli