A Transition-Based Directed Acyclic Graph Parser for Universal Conceptual Cognitive Annotation

Daniel Hershcovich, Omri Abend and Ari Rappoport

University of Washington
July 26, 2017
TUPA — Transition-based UCCA Parser

The **first parser** to support the combination of three properties:

1. **Non-terminal nodes** — entities and events over the text
TUPA — Transition-based UCCA Parser

The **first parser** to support the combination of three properties:

1. **Non-terminal nodes** — entities and events over the text
2. **Reentrancy** — allow argument sharing

You want to take a long bath
The first parser to support the combination of three properties:
1. Non-terminal nodes — entities and events over the text
2. Reentrancy — allow argument sharing
3. Discontinuity — conceptual units are split
   — needed for many semantic schemes (e.g. AMR, UCCA).
Introduction
Linguistic Structure Annotation Schemes

- Syntactic dependencies (Nivre, 2005)
- Semantic dependencies (Oepen et al., 2016)
- AMR (Banarescu et al., 2013)
- UCCA (Abend and Rappoport, 2013)
- Other semantic representation schemes

Abstract away from syntactic detail that does not affect meaning:

\[ \ldots \text{bathed} = \ldots \text{took a bath} \]

\(^1\text{See recent survey (Abend and Rappoport, 2017)}\)
Syntactic Dependencies

- Bilexical tree: syntactic structure representation.
- Fast and accurate parsers (e.g. *transition-based*).

You want to take a long bath

Non-projectivity (discontinuity) is a challenge (Nivre, 2009).
Semantic Dependencies

- Bilexical graph: predicate-argument representation.
- Derived from theories of syntax-semantics interface.

You want to take a long bath

DELPH-IN MRS-derived bi-lexical dependencies (DM).

After graduation, Joe moved to Paris

Prague Dependency Treebank tectogrammatical layer (PSD).
The UCCA Semantic Representation Scheme
After graduation, Joe moved to Paris.
The UCCA Semantic Representation Scheme

- Cross-linguistically applicable (Abend and Rappoport, 2013).
- Stable in translation (Sulem et al., 2015).
- Fast and intuitive to annotate (Abend et al., 2017).
- Facilitates MT human evaluation (Birch et al., 2016).
Graph Structure

UCCA generates a directed acyclic graph (DAG): **no parser yet.** Text tokens are terminals, complex units are **non-terminal nodes.** *Remote edges* enable **reentrancy** for argument sharing. Phrases may be **discontinuous** (e.g., multi-word expressions).

You want to take a long bath

--- primary edge

- - - remote edge
Transition-based UCCA Parsing
Transition-Based Parsing

First used for dependency parsing (Nivre, 2004).
Parse text $w_1 \ldots w_n$ to graph $G$ incrementally by applying transitions to the parser state: stack, buffer and constructed graph.
Transition-Based Parsing

First used for dependency parsing (Nivre, 2004).
Parse text $w_1 \ldots w_n$ to graph $G$ incrementally by applying transitions to the parser state: stack, buffer and constructed graph.

Initial state:

stack

You    want   to    take    a    long    bath

buffer

$\bullet$
Transition-Based Parsing

First used for dependency parsing (Nivre, 2004). Parse text $w_1 \ldots w_n$ to graph $G$ incrementally by applying transitions to the parser state: stack, buffer and constructed graph.

Initial state:

stack  

buffer

You want to take a long bath

TUPA transitions:

$\{\text{Shift}, \text{Reduce}, \text{Node}_X, \text{Left-Edge}_X, \text{Right-Edge}_X, \text{Left-Remote}_X, \text{Right-Remote}_X, \text{Swap}, \text{Finish}\}$

Support non-terminal nodes, reentrancy and discontinuity.
Example

⇒ SHIFT

stack

buffer

You

want to take a long bath
Example

⇒ \text{RIGHT-EDGE}_A
Example

⇒ **SHIFT**

stack

- You
- want

buffer

- to
- take
- a
- long
- bath

graph

- You
Example

⇒ SWAP

stack

buffer

<table>
<thead>
<tr>
<th>●</th>
<th>want</th>
</tr>
</thead>
</table>

| You | to | take | a | long | bath |

You want to take a long bath

graph

You

A
Example

$\Rightarrow$ \textsc{Right-Edge}_P

stack

buffer

\begin{align*}
&\bullet & \text{want} \\
\text{You} & \to & \text{take} & a & \text{long} & \text{bath} \\
\end{align*}

graph

You \quad \text{want}
Example

⇒ REDUCE

stack

buffer

to take a long bath

graph

You want

A

P
Example

⇒ \textbf{SHIFT}

stack

\begin{array}{cc}
\bullet & \text{You} \\
\end{array}

buffer

\begin{array}{cccc}
to & take & a & long & bath \\
\end{array}

graph

\begin{tikzpicture}
    \node (A) at (0,0) {You};
    \node (B) at (1,1) {P};
    \draw[->] (A) -- (B);
    \node (C) at (1,-1) {want};
    \draw[->] (B) -- (C);
    \end{tikzpicture}
Example

⇒ **SHIFT**

![Graph showing the relationship between stack and buffer]

- Stack: You to take a long bath
- Buffer: take a long bath

The graph illustrates the relationship between the stack and the buffer.
Example

⇒ NODE_F

stack

You

to

buffer

take

a

long

bath

graph

You

want

A

P

F

to
Example

⇒ REDUCE

stack

buffer

<table>
<thead>
<tr>
<th>●</th>
<th>You</th>
</tr>
</thead>
</table>

| ● | take | a | long | bath |

graph

You 🔳

want 🔳

P 🔳

A 🔳

to 🔳

F 🔳
Example

⇒ **SHIFT**

stack

| ● | You | ● |

buffer

| take | a | long | bath |

You take a long bath

**graph**

You want to

F

to
Example

⇒ \textbf{SHIFT}

stack

\begin{tabular}{ccc}
  \textbullet & You & \textbullet & \textbf{take} \\
\end{tabular}

buffer

\begin{tabular}{ccc}
a & long & bath \\
\end{tabular}

graph

\begin{itemize}
  \item You
  \item want
  \item to
\end{itemize}

\begin{itemize}
  \item A
  \item P
  \item F
\end{itemize}
Example

⇒ NODEC

stack

buffer

<table>
<thead>
<tr>
<th></th>
<th>You</th>
<th>take</th>
</tr>
</thead>
</table>

|   | a    | long | bath |

graph

You \(\xrightarrow{A} P\) want

You \(\xrightarrow{F} to\) C \(\xrightarrow{C} take\)
Example

⇒ REDUCE

stack

You

buffer

a long bath

graph

You

want

take

A P

F

to

C
Example

⇒ Shift

stack

buffer

You  a  long  bath

You  want  to  take

A  P  F  C

diagram:
Example

⇒ RIGHT-EDGE_P

stack

<table>
<thead>
<tr>
<th></th>
<th>You</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

buffer

|   | a    | long | bath |

graph

A  P

You want

take
Example

\[ \Rightarrow \text{SHIFT} \]

<table>
<thead>
<tr>
<th>stack</th>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="graph.png" alt="Graph" /></td>
<td><img src="graph.png" alt="Graph" /></td>
</tr>
</tbody>
</table>

You want to take a long bath.
Example

\[ \Rightarrow \text{RIGHT-EDGE}_F \]

stack

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>♦</td>
<td>You</td>
<td>a</td>
</tr>
</tbody>
</table>

buffer

| long | bath |

graph

![Graph Diagram]

1. You
2. You want to
3. Take a
Example

$\Rightarrow$ REDUCE

\[
\begin{array}{ccc}
\text{stack} & \quad & \text{buffer} \\
\text{You} & \quad & \text{long} \\
\text{long bath} & \quad & \text{long bath}
\end{array}
\]

\begin{array}{c}
\text{graph} \\
A \\
You \\
f \quad \quad P \\
want \\
to \\
\text{take} \\
a \\
f \\
f \\
F \\
P \\
C \\
P \\
to \\
A
\end{array}
Example

⇒ \textbf{SHIFT}

stack

buffer

Graph

You \rightarrow P \rightarrow A

long

bath

take a to F P F C

You want
Example

\[ \Rightarrow SWAP \]

stack

<table>
<thead>
<tr>
<th></th>
<th>You</th>
<th>long</th>
</tr>
</thead>
</table>

buffer

| bath |

graph

You \[ \rightarrow \] want

You \[ \rightarrow \] to \[ \rightarrow \] take

<table>
<thead>
<tr>
<th>A</th>
<th>P</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>P</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>C</th>
</tr>
</thead>
</table>
Example

$\Rightarrow \text{RIGHT-EDGE}_D$

You

You want to take a long bath while being out.
Example

$\Rightarrow$ REDUCE

stack

buffer

A

w

P

F

C

d

You

want

take

a

long

bath
Example

⇒ SWAP

stack

buffer

You

bath

graph

You want

A P

to
d

D

C F

take a long
Example

\[ \Rightarrow \text{RIGHT-EDGE}_A \]

stack

buffer

You

bath

diagram

graph

You

want

to

take

a

long

A

P

A

P

F

F

C
Example

⇒ REDUCE

graph

You

want

A

P

A

D

to

take

a

long

stack

buffer

You

bath

You

A

P

C

F

F

P
Example

⇒ REDUCE

stack

buffer

You | bath

graph

You

want

to

take

a

long

A

P

A

F

P

C

F

D
Example
\[ \Rightarrow \text{SHIFT} \]

stack

buffer

You

You

You

want

take

a

long

to

F

P

C

F

D

graph

You

A

P

A

P

F

P

F

Red
Example

⇒ **SHIFT**

<table>
<thead>
<tr>
<th>stack</th>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>You</td>
<td>bath</td>
</tr>
</tbody>
</table>

**Graph**

- **You**
  - **A**
  - **P**
- **want**
  - **A**
- **to**
  - **C**
  - **F**
- **take**
  - **a**
- **long**
  - **D**
Example

$\Rightarrow \text{LEFT-REMOTE}_A$

stack

buffer

You

bath

graph

You

want

take

a

long

D

C

F

P

A

A

P

A
Example

⇒ SHIFT

stack

buffer

You | bath

You A want P A

A to F P D

A F C F

take a long
Example

⇒ \text{RIGHT-EDGE}_C

stack

You \quad \bullet \quad \text{bath}

buffer

cgraph

You \quad \rightarrow \quad \text{want} \quad \rightarrow \quad \text{to} \quad \rightarrow \quad \text{take} \quad \rightarrow \quad \text{a} \quad \rightarrow \quad \text{long} \quad \rightarrow \quad \text{bath}
Example

⇒ FINISH

stack

buffer

You | bath

You want to take a long bath.

A
 P
 A

A

F
 P

D
 C

C

A

F

A

graph
Training

An oracle provides the transition sequence given the correct graph:

You want to take a long bath

\[
\text{Shift, Right-Edge}_A, \text{ Shift, Swap, Right-Edge}_P, \text{ Reduce, Shift, Shift, Node}_F, \text{ Reduce, Shift, Node}_C, \text{ Reduce, Shift, Right-Edge}_P, \text{ Shift, Right-Edge}_F, \text{ Reduce, Shift, Swap, Right-Edge}_D, \text{ Reduce, Swap, Right-Edge}_A, \text{ Reduce, Reduce, Shift, Shift, Left-Remote}_A, \text{ Shift, Right-Edge}_C, \text{ Finish}
\]
TUPA Model

Learn to greedily predict transition based on current state.
Experimenting with three classifiers:

- **Sparse**: Perceptron with sparse features.
- **MLP**: Embeddings + feedforward NN classifier.
- **BiLSTM**: Embeddings + deep bidirectional LSTM + MLP (Kiperwasser and Goldberg, 2016).

Features: words, POS, syntactic dependencies, existing edge labels from the stack and buffer + parents, children, grandchildren; ordinal features (height, number of parents and children)
TUPA Model

Learn to greedily predict transition based on current state. Experimenting with three classifiers:

- **Sparse**  Perceptron with sparse features.
- **MLP**  Embeddings + feedforward NN classifier.
- **BiLSTM**  Embeddings + deep bidirectional LSTM + MLP (Kiperwasser and Goldberg, 2016).

Effective “lookahead” encoded in the representation.
TUPA Model

Learn to greedily predict transition based on current state. Experimenting with three classifiers:

**Sparse** Perceptron with sparse features.

**MLP** Embeddings + feedforward NN classifier.

**BiLSTM** Embeddings + deep bidirectional LSTM + MLP (Kiperwasser and Goldberg, 2016).

Effective “lookahead” encoded in the representation.
TUPA Model

Learn to greedily predict transition based on current state.
Experimenting with three classifiers:

**Sparse** Perceptron with sparse features.
**MLP** Embeddings + feedforward NN classifier.
**BiLSTM** Embeddings + deep bidirectional LSTM + MLP (Kiperwasser and Goldberg, 2016).

Effective “lookahead” encoded in the representation.
**TUPA Model**

Learn to greedily predict transition based on current state. Experimenting with three classifiers:

- **Sparse** Perceptron with sparse features.
- **MLP** Embeddings + feedforward NN classifier.
- **BiLSTM** Embeddings + **deep bidirectional LSTM** + MLP (Kiperwasser and Goldberg, 2016).

```
You want to take a long bath
```
You want to take a long bath.
Experiments
Experimental Setup

- UCCA Wikipedia corpus ($4268 + 454 + 503$ sentences).
Baselines

No existing UCCA parsers $\Rightarrow$ conversion-based approximation.

Bilexical DAG parsers (allow reentrancy):

- DAGParser (Ribeyre et al., 2014): transition-based.

Tree parsers (all transition-based):

- MaltParser (Nivre et al., 2007): bilexical tree parser.
- Stack LSTM Parser (Dyer et al., 2015): bilexical tree parser.
- uparse (Maier, 2015): allows non-terminals, discontinuity.

UCCA bilexical DAG approximation (for tree, delete remote edges).
Bilexical Graph Approximation

1. Convert UCCA to bilexical dependencies.
2. Train bilexical parsers and apply to test sentences.
3. Reconstruct UCCA graphs and compare with gold standard.

After graduation, Joe moved to Paris.
Evaluation

Comparing graphs over the same sequence of tokens,

- Match edges by their terminal yield and label.
- Calculate **labeled precision, recall and F1 scores**.
- Separate primary and remote edges.

### Primary:

<table>
<thead>
<tr>
<th>LP</th>
<th>LR</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{6}{9} = 67%$</td>
<td>$\frac{6}{10} = 60%$</td>
<td>$64%$</td>
</tr>
</tbody>
</table>

### Remote:

<table>
<thead>
<tr>
<th>LP</th>
<th>LR</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{2} = 50%$</td>
<td>$\frac{1}{1} = 100%$</td>
<td>$67%$</td>
</tr>
</tbody>
</table>
## Results

**TUPA BiLSTM** obtains the highest F-scores in all metrics:

<table>
<thead>
<tr>
<th></th>
<th>Primary edges</th>
<th>Remote edges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
</tr>
<tr>
<td><strong>TUPA Sparse</strong></td>
<td>64.5</td>
<td>63.7</td>
</tr>
<tr>
<td><strong>TUPA MLP</strong></td>
<td>65.2</td>
<td>64.6</td>
</tr>
<tr>
<td><strong>TUPA BiLSTM</strong></td>
<td>74.4</td>
<td>72.7</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAGParser</td>
<td>61.8</td>
<td>55.8</td>
</tr>
<tr>
<td>TurboParser</td>
<td>57.7</td>
<td>46</td>
</tr>
<tr>
<td>Bilexical tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaltParser</td>
<td>62.8</td>
<td>57.7</td>
</tr>
<tr>
<td>Stack LSTM</td>
<td>73.2</td>
<td>66.9</td>
</tr>
<tr>
<td>Tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPARSE</td>
<td>60.9</td>
<td>61.2</td>
</tr>
</tbody>
</table>

Results on the Wiki test set.
## Results

Comparable on out-of-domain test set:

<table>
<thead>
<tr>
<th></th>
<th>Primary edges</th>
<th></th>
<th>Remote edges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
<td>LF</td>
<td>LP</td>
</tr>
<tr>
<td>TUPA&lt;sub&gt;Sparse&lt;/sub&gt;</td>
<td>59.6</td>
<td>59.9</td>
<td>59.8</td>
<td>22.2</td>
</tr>
<tr>
<td>TUPA&lt;sub&gt;MLP&lt;/sub&gt;</td>
<td>62.3</td>
<td>62.6</td>
<td>62.5</td>
<td>20.9</td>
</tr>
<tr>
<td>TUPA&lt;sub&gt;BiLSTM&lt;/sub&gt;</td>
<td>68.7</td>
<td>68.5</td>
<td>68.6</td>
<td>38.6</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td></td>
<td></td>
<td>(91.3)</td>
<td></td>
</tr>
<tr>
<td>DAGParser</td>
<td>56.4</td>
<td>50.6</td>
<td>53.4</td>
<td></td>
</tr>
<tr>
<td>TurboParser</td>
<td>50.3</td>
<td>37.7</td>
<td>43.1</td>
<td>100</td>
</tr>
<tr>
<td>Bilexical tree</td>
<td></td>
<td></td>
<td>(91.3)</td>
<td></td>
</tr>
<tr>
<td>MaltParser</td>
<td>57.8</td>
<td>53</td>
<td>55.3</td>
<td></td>
</tr>
<tr>
<td>Stack LSTM</td>
<td>66.1</td>
<td>61.1</td>
<td>63.5</td>
<td></td>
</tr>
<tr>
<td>Tree</td>
<td></td>
<td></td>
<td>(100)</td>
<td></td>
</tr>
<tr>
<td>UPARSE</td>
<td>52.7</td>
<td>52.8</td>
<td>52.8</td>
<td></td>
</tr>
</tbody>
</table>

Results on the 20K Leagues out-of-domain set.
Discussion
Fine-Grained Analysis

Evaluation of TUPA_{BiLSTM} per edge type:

- Adverbial: 76.9, 72.1, 74.4
- Center: 76.9, 77.3, 77.1
- Connector: 90, 76.8, 82.9
- Elaborator: 69.8, 73.5, 71.6
- Function: 69.3, 81.2, 74.7

- Linker: 69.6, 71.7, 70.6
- Linked scene: 61.2, 59.3, 60.2
- Participant: 69.3, 65.6, 67.4
- Process: 73.4, 71.1, 72.2
- Relator: 91.8, 92.4, 92.1
- State: 43, 26.9, 33.1

Legend: LP, LR, LF
Online Demo


Input text:
Paul went home after working for eight hours.

Parse

Legend

P  process
S  state
A  participant
L  linker
H  linked scene
C  center
E  elaborator
D  adverbial
R  relator
N  connector
U  punctuation
F  function
G  ground
Error Analysis

Copular clauses tend to be parsed as identity.

But, from the guidelines:\(^2\)

\[
\text{John}_A \left[ \text{is}_F \left[ \left[ \text{six}_E \text{years}_C \right] \text{old}_C \right]_C \right]_S
\]

\(^2\)http://www.cs.huji.ac.il/~oabend/ucca/guidelines.pdf
Future Work
Future Work: UCCA

Already annotated in UCCA, but not yet handled by TUPA:

- Linkage: inter-scene relations (see example).
- Implicit units: units not mentioned at all in the text.
- Inter-sentence relations: discourse structure.

UCCA graph with a Linkage relation.
Future Work: AMR

Similar in structure and content, but poses several challenges:

- Node labels: not just edges, not also nodes are labeled.
- Partial alignment: orphan tokens, implicit concepts.

AMR graph in UCCA format.
Future Work: SDP

Similar structure, but without non-terminal nodes. By applying bilexical conversion in reverse, TUPA can be used.

Semantic dependency graph (DM) in UCCA format.
Conclusion

- UCCA’s semantic distinctions require a graph structure including non-terminals, reentrancy and discontinuity.
- TUPA is an accurate transition-based UCCA parser, and the first to support UCCA and any DAG over the text tokens.
- Outperforms strong conversion-based baselines.

Code: https://github.com/danielhers/tupa
Corpora: http://www.cs.huji.ac.il/~oabend/ucca.html
Conclusion

- UCCA’s semantic distinctions require a graph structure including non-terminals, reentrancy and discontinuity.
- TUPA is an accurate transition-based UCCA parser, and the first to support UCCA and any DAG over the text tokens.
- Outperforms strong conversion-based baselines.

Future Work:
- More languages (German corpus construction is underway).
- Broad coverage UCCA parsing.
- Parsing other schemes, such as AMR.
- Text simplification, MT evaluation and other applications.

Code: https://github.com/danielhers/tupa
Corpora: http://www.cs.huji.ac.il/~oabend/ucca.html
Conclusion

- UCCA’s semantic distinctions require a graph structure including non-terminals, reentrancy and discontinuity.
- TUPA is an accurate transition-based UCCA parser, and the first to support UCCA and any DAG over the text tokens.
- Outperforms strong conversion-based baselines.

Future Work:

- More languages (German corpus construction is underway).
- Broad coverage UCCA parsing.
- Parsing other schemes, such as AMR.
- Text simplification, MT evaluation and other applications.

Code: https://github.com/danielhers/tupa
Corpora: http://www.cs.huji.ac.il/~oabend/ucca.html

Thank you!
References I


References II

Simple and accurate dependency parsing using bidirectional LSTM feature representations.
*TACL*, 4:313–327.

Discontinuous incremental shift-reduce parsing.

Incrementality in deterministic dependency parsing.

Dependency grammar and dependency parsing.
Technical Report MSI 05133, Växjö University, School of Mathematics and Systems Engineering.

Non-projective dependency parsing in expected linear time.

MaltParser: A language-independent system for data-driven dependency parsing.

Towards comparability of linguistic graph banks for semantic parsing.
In *LREC*. 
Alpage: Transition-based semantic graph parsing with syntactic features.
In Proc. of SemEval, pages 97–103.

Conceptual annotations preserve structure across translations: A French-English case study.
In Proc. of S2MT, pages 11–22.
Backup
### UCCA Corpora

<table>
<thead>
<tr>
<th></th>
<th>Wiki Train</th>
<th>Wiki Dev</th>
<th>Wiki Test</th>
<th>20K Leagues</th>
</tr>
</thead>
<tbody>
<tr>
<td># passages</td>
<td>300</td>
<td>34</td>
<td>33</td>
<td>154</td>
</tr>
<tr>
<td># sentences</td>
<td>4268</td>
<td>454</td>
<td>503</td>
<td>506</td>
</tr>
<tr>
<td># nodes</td>
<td>298,993</td>
<td>33,704</td>
<td>35,718</td>
<td>29,315</td>
</tr>
<tr>
<td>% terminal</td>
<td>42.96</td>
<td>43.54</td>
<td>42.87</td>
<td>42.09</td>
</tr>
<tr>
<td>% non-term.</td>
<td>58.33</td>
<td>57.60</td>
<td>58.35</td>
<td>60.01</td>
</tr>
<tr>
<td>% discont.</td>
<td>0.54</td>
<td>0.53</td>
<td>0.44</td>
<td>0.81</td>
</tr>
<tr>
<td>% reentrant</td>
<td>2.38</td>
<td>1.88</td>
<td>2.15</td>
<td>2.03</td>
</tr>
<tr>
<td># edges</td>
<td>287,914</td>
<td>32,460</td>
<td>34,336</td>
<td>27,749</td>
</tr>
<tr>
<td>% primary</td>
<td>98.25</td>
<td>98.75</td>
<td>98.74</td>
<td>97.73</td>
</tr>
<tr>
<td>% remote</td>
<td>1.75</td>
<td>1.25</td>
<td>1.26</td>
<td>2.27</td>
</tr>
<tr>
<td>Average per non-terminal node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># children</td>
<td>1.67</td>
<td>1.68</td>
<td>1.66</td>
<td>1.61</td>
</tr>
</tbody>
</table>

Corpus statistics.
Evaluation

*Mutual edges* between predicted graph $G_p = (V_p, E_p, \ell_p)$ and gold graph $G_g = (V_g, E_g, \ell_g)$, both over terminals $W = \{w_1, \ldots, w_n\}$:

$$M(G_p, G_g) = \{(e_1, e_2) \in E_p \times E_g \mid y(e_1) = y(e_2) \land \ell_p(e_1) = \ell_g(e_2)\}$$

The yield $y(e) \subseteq W$ of an edge $e = (u, v)$ in either graph is the set of terminals in $W$ that are descendants of $v$. $\ell$ is the edge label.

Labeled precision, recall and F-score are then defined as:

$$LP = \frac{|M(G_p, G_g)|}{|E_p|}, \quad LR = \frac{|M(G_p, G_g)|}{|E_g|},$$

$$LF = \frac{2 \cdot LP \cdot LR}{LP + LR}.$$  

Two variants: one for primary edges, and another for remote edges.