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

Daniel Hershcovich, Omri Abend and Ari Rappoport

ACL 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
2. **Reentrancy** — allow argument sharing
3. **Discontinuity** — conceptual units are split — needed for many semantic schemes (e.g. AMR, UCCA).

You want to take a long bath
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
- Semantic dependencies (Oepen et al., 2016)

**Syntactic (UD)**

You \(\rightarrow\) want \(\rightarrow\) to \(\rightarrow\) take \(\rightarrow\) a \(\rightarrow\) long \(\rightarrow\) bath

- nsubj
- xcomp
- mark
- dobj
- det
- amod
- ARG1
- ARG2
- top

**Semantic (DM)**

Bilexical dependencies.
Linguistic Structure Annotation Schemes

- Syntactic dependencies
- Semantic dependencies (Oepen et al., 2016)
- Semantic role labeling (PropBank, FrameNet)
- AMR (Banarescu et al., 2013)
- UCCA (Abend and Rappoport, 2013)
- Other semantic representation schemes\(^1\)

Semantic representation schemes attempt to abstract away from syntactic detail that does not affect meaning:

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

\(^1\)See recent survey (Abend and Rappoport, 2017)
The UCCA Semantic Representation Scheme
Universal Conceptual Cognitive Annotation (UCCA)

Cross-linguistically applicable (Abend and Rappoport, 2013). Stable in translation (Sulem et al., 2015).

English

IBM happened to choose a company with a crucial vulnerability, despite vetting.

Hebrew

IBM שבדקה תזמורת פיגוע מחוד שحضر ב-חברה בסעודה ב- nackt be-xevra me'od pgi'a lamrot še-badka ota meroš
Universal Conceptual Cognitive Annotation (UCCA)

Rapid and intuitive annotation interface (Abend et al., 2017).
Usable by non-experts. ucca-demo.cs.huji.ac.il

Facilitates semantics-based human evaluation of machine translation (Birch et al., 2016). ucca.cs.huji.ac.il/mteval

William Bradley Pitt was born in Shawnee, Oklahoma, to William Alvin Pitt, who ran a trucking company, and Jane Etta (née Hillhouse), a school counselor. The family soon moved to Springfield, Missouri, where he lived together with his younger siblings, Douglas (born 1966) and Julie Neal (born 1969). Born into a conservative household, he was raised as Southern Baptist, but has since stated that he does not have a great relationship with religion and that he oscillates between agnosticism and atheism. Pitt has described Springfield as "Mark Twain country", Jesse James country, having grown up with "a lot of hills, a lot of lakes".
Graph Structure

UCCA generates a directed acyclic graph (DAG). 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
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          buffer

●       You want to take a long bath
```
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
...
```

TUPA transitions:

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

Support non-terminal nodes, reentrancy and discontinuity.
Example

⇒ SHIFT

stack

buffer

- You

want to take a long bath

graph
Example

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

stack  buffer

You | want to take a long bath

graph

You
Example

⇒ $\text{SHIFT}$

stack

- You
- want

buffer

- to
- take
- a
- long
- bath

graph

- A

- You
Example

⇒ SWAP

You want to take a long bath

graph

You

A

stack

buffer

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

You to take a long bath
Example

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

stack

\[ \begin{array}{c}
\bullet \\
\text{want}
\end{array} \quad \begin{array}{c}
\text{buffer}
\end{array} \]

You \quad to \quad take \quad a \quad long \quad bath

graph

You \quad want

\[ \begin{array}{c}
A \\
P
\end{array} \]
Example

⇒ REDUCE

stack

buffer

to take a long bath

graph

You want

A P
Example

⇒ \textbf{SHIFT}

\begin{itemize}
  \item \textbf{stack}
  \begin{itemize}
    \item \textbf{buffer}
    \begin{itemize}
      \item \textbf{You}
      \begin{itemize}
        \item \textbf{You}
        \begin{itemize}
          \item \textbf{want}
          \begin{itemize}
            \item \textbf{to take a long bath}
          \end{itemize}
        \end{itemize}
      \end{itemize}
    \end{itemize}
  \end{itemize}
\end{itemize}
\end{itemize}
Example

⇒ **SHIFT**

stack

- **You** to

buffer

- take a long bath

graph

- You want

A

P
Example

$\Rightarrow \text{NODE}_F$

stack

buffer

You to

take a long bath

graph

You want to

to
Example

⇒ REDUCE

<table>
<thead>
<tr>
<th>stack</th>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="black" alt="You" /></td>
<td>take a long bath</td>
</tr>
</tbody>
</table>

graph

- You
- want
- to

A
P

F
Example

 ⇒ SHIFT

You take a long bath

You want to

graph

stack

buffer

A P

F

to
Example

\[ \Rightarrow \text{SHIFT} \]

stack

\[ \bullet \quad \text{You} \quad \bullet \quad \text{take} \]

buffer

\[ \begin{array}{c} \text{a} \\ \text{long} \\ \text{bath} \end{array} \]

graph

\[ \begin{array}{c} \text{You} \\ \text{want} \\ \text{to} \end{array} \]
Example

⇒ NODEC

stack

buffer

*You* take

You

take

A

P


want

F

to

C

a long bath
Example

⇒ REDUCE

stack

buffer

You

a

long

bath

c

take

guest

A

P

F

to

C

want

You
Example

⇒ **SHIFT**

stack

You

buffer

a  long  bath

graph

You  want

C  to  take

A  P  F
Example

⇒ RIGHT-EDGE_P

stack

buffer

You

buffer

a long bath

tagh

You

want

You

take

want

A

P

F

to

C

A

P

P

Example

$\Rightarrow$ Shift

stack

You | a

buffer

long bath

graph

You

want

to

take

A

P

F

P

C
Example

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

stack

\[
\begin{array}{c|c|c|c}
\text{buffer} & \text{long} & \text{bath} \\
\hline
\bullet & \text{You} & \bullet & \bullet & \text{a} \\
\hline
\end{array}
\]

graph

- You
  - want
    - You
    - to
      - take
        - a
        - C
          - F
    - P
      - A
        - P
          - F

- You
  - want
Example

\[ \Rightarrow \text{REDUCE} \]

stack

 buffer

<table>
<thead>
<tr>
<th>●</th>
<th>You</th>
<th>●</th>
<th>●</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>bath</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

graph

You → want

A

P

You

want

F

P

to

C

F

take

a
Example

\[ \Rightarrow \text{SHIFT} \]

stack

- You
- long

buffer

- bath

graph

- You
- want

- to
- take
- a

- P
- A
- F C P F
Example

⇒ SWAP

stack

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

buffer

| bath |

graph

```
A
    /
   /  
/    
You

P

want
```

to

```
F
    /
   /  
/    
|    
|    
to

to

take

take
```

A
    /
   /  
/    
C

F

P

P

F

F

A

C

P

F
Example

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

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

graph

- You
- want
- take a long
- to
- F
- P
- D

You want to take a long bath.
Example

⇒ REDUCE

stack

buffer

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

|    | bath |

graph

A
P

You

want

F
P

to

C
F

D

long

take

a
Example

⇒ SWAP

graph

stack

buffer

You

bath
Example

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

stack

$\bullet$ $\bullet$

buffer

You

You want to take a long bath

graph
Example

⇒ REDUCE

stack

buffer

You

bath

graph

You

want

You

A

P

A

F

P

to

D

take

a

long

A

F

C

F
Example

⇒ REDUCE

stack

buffer

You

bath

graph

You

want

A

P

A

to

C

F

D

P

F

a

take

long
Example

$\Rightarrow \text{SHIFT}$

stack

buffer

You

You

bath

A

want

A

F

to

P

take

long

D
Example

⇒ \textbf{SHIFT}

stack

You [red]

buffer

bath

diagram:

You \quad \text{want} \quad \text{to} \quad \text{take} \quad \text{a} \quad \text{long}

A \quad P \quad A \quad F \quad P \quad D

You \quad want \quad to \quad take \quad a \quad long

A \quad F \quad C \quad F
Example

\[ \Rightarrow \text{LEFT-REMOTE}_A \]
Example

⇒ **SHIFT**

stack

| You | red bath |

buffer

graph
Example

⇒ $\text{RIGHT-EDGE}_C$

stack

You bath

buffer

You want to take a long bath

diagram

graph

You P A A P F F P D C C A A C
Example

⇒ FINISH

stack

buffer

You | bath

graph

You

want

to

take

a

long

bath
An *oracle* provides the transition sequence given the correct graph:

\[
\downarrow
\]

**Shift, Right-Edge\(_A\), Shift, Swap, Right-Edge\(_P\), Reduce, Shift, Shift, Node\(_F\), Reduce, Shift, Shift, Node\(_C\), Reduce, Shift, Right-Edge\(_P\), Shift, Right-Edge\(_F\), Reduce, Shift, Swap, Right-Edge\(_D\), Reduce, Swap, Right-Edge\(_A\), Reduce, Reduce, Shift, Shift, Left-Remote\(_A\), Shift, Right-Edge\(_C\), Finish**
TUPA Model

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

**Sparse**  Perceptron with sparse features (Zhang and Nivre, 2011).
**MLP**  Embeddings + feedforward NN (Chen and Manning, 2014).
**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 (Zhang and Nivre, 2011).
**MLP** Embeddings + feedforward NN (Chen and Manning, 2014).
**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 (Zhang and Nivre, 2011).
- **MLP**: Embeddings + feedforward NN (Chen and Manning, 2014).
- **BiLSTM**: Embeddings + deep bidirectional LSTM + MLP (Kiperwasser and Goldberg, 2016).
Learn to greedily predict transition based on current state. Experimenting with three classifiers:

**Sparse** Perceptron with sparse features (Zhang and Nivre, 2011).

**MLP** Embeddings + feedforward NN (Chen and Manning, 2014).

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

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

**Sparse**  Perceptron with sparse features (Zhang and Nivre, 2011).
**MLP**  Embeddings + feedforward NN (Chen and Manning, 2014).
**BiLSTM**  Embeddings + **deep bidirectional LSTM** + MLP (Kiperwasser and Goldberg, 2016).
You want to take a long bath.

Diagram:

Stack:
- You
- take

Buffer:
- a
- long
- bath

Graph:
- You
- want
- to
- take

MLP Node:
- NODEC

LSTMs:
- You
- want
- to
- take
- a
- long
- bath
Experiments
Experimental Setup

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

No existing UCCA parsers ⇒ conversion-based approximation.

Bilexical DAG parsers (allow \textit{reentrancy}):\footnote{In computer science, \textit{reentrancy} refers to a program's ability to safely return to a previous state. In the context of parsing, it allows for repeated processing of a sentence in different ways.} 

- 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.
- \textsc{uparse} (Maier, 2015): allows non-terminals, \textit{discontinuity}.

You want to take a long bath

\textit{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.
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></th>
<th>LP</th>
<th>LR</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>gold</td>
<td>$\frac{6}{9} = 67%$</td>
<td>$\frac{10}{9} = 60%$</td>
<td>64%</td>
</tr>
<tr>
<td>predicted</td>
<td>$\frac{1}{2} = 50%$</td>
<td>$\frac{1}{1} = 100%$</td>
<td>67%</td>
</tr>
</tbody>
</table>
## Results

**TUPA\textsubscript{BiLSTM}** obtains the highest F-scores in all metrics:

<table>
<thead>
<tr>
<th></th>
<th>Primary edges</th>
<th></th>
<th>Remote edges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
<td>LF</td>
</tr>
<tr>
<td>TUPA\textsubscript{Sparse}</td>
<td>64.5</td>
<td>63.7</td>
<td>64.1</td>
</tr>
<tr>
<td>TUPA\textsubscript{MLP}</td>
<td>65.2</td>
<td>64.6</td>
<td>64.9</td>
</tr>
<tr>
<td>TUPA\textsubscript{BiLSTM}</td>
<td>74.4</td>
<td>72.7</td>
<td>73.5</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td></td>
<td></td>
<td>(91)</td>
</tr>
<tr>
<td>DAGParser</td>
<td>61.8</td>
<td>55.8</td>
<td>58.6</td>
</tr>
<tr>
<td>TurboParser</td>
<td>57.7</td>
<td>46</td>
<td>51.2</td>
</tr>
<tr>
<td>Bilexical tree</td>
<td></td>
<td></td>
<td>(91)</td>
</tr>
<tr>
<td>MaltParser</td>
<td>62.8</td>
<td>57.7</td>
<td>60.2</td>
</tr>
<tr>
<td>Stack LSTM</td>
<td>73.2</td>
<td>66.9</td>
<td>69.9</td>
</tr>
<tr>
<td>Tree</td>
<td></td>
<td></td>
<td>(100)</td>
</tr>
<tr>
<td>UPARSE</td>
<td>60.9</td>
<td>61.2</td>
<td>61.1</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>Remote edges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
</tr>
<tr>
<td>$TUPA_{Sparse}$</td>
<td>59.6</td>
<td>59.9</td>
</tr>
<tr>
<td>$TUPA_{MLP}$</td>
<td>62.3</td>
<td>62.6</td>
</tr>
<tr>
<td>$TUPA_{BiLSTM}$</td>
<td>68.7</td>
<td>68.5</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td>(91.3)</td>
<td></td>
</tr>
<tr>
<td>DAGParser</td>
<td>56.4</td>
<td>50.6</td>
</tr>
<tr>
<td>TurboParser</td>
<td>50.3</td>
<td>37.7</td>
</tr>
<tr>
<td>Bilexical tree</td>
<td>(91.3)</td>
<td></td>
</tr>
<tr>
<td>MaltParser</td>
<td>57.8</td>
<td>53</td>
</tr>
<tr>
<td>Stack LSTM</td>
<td>66.1</td>
<td>61.1</td>
</tr>
<tr>
<td>Tree</td>
<td>(100)</td>
<td></td>
</tr>
<tr>
<td>uparse</td>
<td>52.7</td>
<td>52.8</td>
</tr>
</tbody>
</table>

Results on the 20K Leagues out-of-domain set.
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: github.com/danielhers/tupa
Demo: bit.ly/tupademo
Corpora: 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).
- Parsing other schemes, such as AMR.
- Compare semantic representations through conversion.
- Text simplification, MT evaluation and other applications.

Code: github.com/danielhers/tupa
Demo: bit.ly/tupademo
Corpora: 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).
- Parsing other schemes, such as AMR.
- Compare semantic representations through conversion.
- Text simplification, MT evaluation and other applications.

Code: github.com/danielhers/tupa
Demo: bit.ly/tupademo
Corpora: cs.huji.ac.il/~oabend/ucca.html

Thank you!
Universal Conceptual Cognitive Annotation (UCCA).
In Proc. of ACL, pages 228–238.

The state of the art in semantic representation.
In Proc. of ACL.
to appear.

UCCAApp: Web-application for syntactic and semantic phrase-based annotation.
to appear.

Lisbon: Evaluating TurboSemanticParser on multiple languages and out-of-domain data.
In Proc. of SemEval, pages 970–973.

Abstract Meaning Representation for sembanking.
In Proc. of the Linguistic Annotation Workshop.


In Proc. of EMNLP, pages 740–750.
References II

Transition-based dependency parsing with stack long short-term memory.

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

Discontinuous incremental shift-reduce parsing.

Incrementality in deterministic dependency parsing.
In Keller, F., Clark, S., Crocker, M., and Steedman, M., editors, Proceedings of the ACL Workshop
Association for Computational Linguistics.

MaltParser: A language-independent system for data-driven dependency parsing.
Natural Language Engineering, 13(02):95–135.

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.

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>
</tbody>
</table>

Average per non-terminal node

<table>
<thead>
<tr>
<th># children</th>
<th>Wiki Train</th>
<th>Wiki Dev</th>
<th>Wiki Test</th>
<th>20K Leagues</th>
</tr>
</thead>
<tbody>
<tr>
<td></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:

$$LP = \frac{|M(G_p, G_g)|}{|E_p|}, \quad LR = \frac{|M(G_p, G_g)|}{|E_g|}, \quad LF = \frac{2 \cdot LP \cdot LR}{LP + LR}.$$ 

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