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

Daniel Hershcovich, Omri Abend and Ari Rappoport

Tel Aviv University
January 9, 2018
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

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

Diagram:

```
  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
3. **Discontinuity** — conceptual units are split
   — needed for many semantic schemes (e.g. AMR, UCCA).

![Diagram of a parse tree for the sentence "You want to take a long bath"]
Introduction
Linguistic Structure Annotation Schemes

- Syntactic dependencies
- 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:

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

\(^1\)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).

```
A hearing is scheduled on the issue today
```
Semantic Dependencies

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

![Diagram of semantic dependencies]

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

![Diagram of semantic dependencies]

**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). 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

You want
want
to
to

—– primary edge
- - - remote edge

You want to take a long bath

<table>
<thead>
<tr>
<th>P</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>participant</td>
</tr>
<tr>
<td>C</td>
<td>center</td>
</tr>
<tr>
<td>D</td>
<td>adverbial</td>
</tr>
<tr>
<td>F</td>
<td>function</td>
</tr>
</tbody>
</table>
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

$\bullet$ | 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

●          You want to take a long bath
```

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

| • | You |

buffer

| want | to | take | a | long | bath |

graph

| • |
Example

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

stack

```
You
```

buffer

```
want to take a long bath
```

graph

```
You
```

\[ A \]
Example

⇒ \textbf{SHIFT}

stack

You \textbf{want} to take a long bath

buffer

graph

You

\textbf{A}
Example

⇒ SWAP

stack

|   ●   | want |

buffer

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

graph

|          | A     |

You

|          |       |       |

You
Example

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

stack

You want to take a long bath

graph

You want

buffer
Example

⇒ REDUCE

stack

.buffer

to take a long bath

graph

You want

A P
Example

$\Rightarrow$ \textbf{SHIFT}

stack

buffer

You

You

want

to take a long bath

A

P
Example

⇒ SHIFT

stack

You to

buffer

take a long bath

graph

A P

You want
Example

\[ \Rightarrow \text{NODE}_F \]

stack

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

buffer

<table>
<thead>
<tr>
<th></th>
<th>take</th>
<th>a</th>
<th>long</th>
<th>bath</th>
</tr>
</thead>
</table>

graph

- You → A
- You → P
- P → F
- F → to
- to → want
- want → A
- A → P

You want to take a long bath.
Example

⇒ REDUCE

stack

buffer

- You
- take a long bath

graph

You want to P A

F to
Example

⇒ \underline{SHIFT}

You \rightarrow \text{buffer}

You want to take a long bath
Example

⇒ \textit{SHIFT}

stack

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

buffer

| a | long | bath |

graph

You \quad \text{want} \quad \text{to} \quad \text{P} \quad \text{A} \quad \text{F}
Example

\[ \Rightarrow \text{NODEC} \]

stack

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

buffer

| a | long | bath |

graph

- You
- want
- to
- take

A
P
F
C
Example

⇒ REDUCE

stack

buffer

You

a

long

bath

You want

to take

A

P

F

C

graph
Example

⇒ \textbf{SHIFT}

stack

\begin{tabular}{|c|c|c|}
\hline
• & You & • \color{red}{\textbullet} \\
\hline
\end{tabular}

buffer

\begin{tabular}{|c|c|c|}
\hline
a & long & bath \\
\hline
\end{tabular}

diagram:

\begin{itemize}
\item You
\item want
\item to
\item take
\item \textbf{A}
\item \textbf{P}
\item \textbf{F}
\item \textbf{C}
\end{itemize}
Example

⇒ \textbf{RIGHT-EDGE}_P

stack

- \textcolor{black}{\textbullet} \hspace{0.5cm} \textcolor{blue}{You} \hspace{0.5cm} \textcolor{red}{P}

buffer

- \textcolor{black}{a} \hspace{0.5cm} \textcolor{blue}{long} \hspace{0.5cm} \textcolor{red}{bath}

graph

- \textcolor{black}{You} \hspace{1cm} \textcolor{blue}{want} \hspace{1cm} \textcolor{red}{take}

- \textcolor{black}{A} \hspace{1cm} \textcolor{red}{P} \hspace{1cm} \textcolor{blue}{F} \hspace{1cm} \textcolor{red}{C}

- \textcolor{black}{to} \hspace{1cm} \textcolor{red}{P}
Example

⇒ $\text{SHIFT}$

stack

buffer

You  a

long  bath

graph

You  want

You  want

take
Example

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

```
stack

● You ● a

buffer

long bath
```

```
graph

You \(\downarrow\) want

\(\downarrow\) F P

to

\(\downarrow\) C F

take a
```
Example

⇒ REDUCE

stack

buffer

A

P

F

long bath

You

want

to
take

C

F

P

A

You
Example

\[ \Rightarrow \text{SHIFT} \]

stack

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

buffer

bath

graph

You \quad want

You \quad want

C \quad F

take \quad a
Example

⇒ SWAP

stack

You

long

buffer

bath

diagram

You

want

A

P

You

take

a

F

P

to

C

F
Example

⇒ \text{RIGHT-EDGE}_D

\begin{array}{c}
\text{stack} \\
\begin{array}{c|c|c}
& \text{You} & \text{long} \\
\bullet & \text{blue} & \text{red} \\
\end{array}
\end{array}

\begin{array}{c}
\text{buffer} \\
\text{red} & \text{bath} \\
\end{array}

\begin{tikzpicture}[thick]
  \node (you) at (0,0) {You};
  \node (want) at (2,0) {want};
  \node (long) at (2,-1) {long};
  \node (take) at (-2,-1) {take};
  \node (a) at (1,-2) {A};
  \node (p) at (1,-3) {P};
  \node (f) at (1,-4) {F};
  \node (c) at (-1,-4) {C};
  \node (d) at (3,-4) {D};

  \draw[->] (you) -- (a);
  \draw[->] (a) -- (p);
  \draw[->] (p) -- (f);
  \draw[->] (f) -- (c);
  \draw[->] (c) -- (take);
  \draw[->] (d) to[out=-90,in=-90,looseness=1.5] (long);
\end{tikzpicture}
Example

$\Rightarrow$ REDUCE

stack

- You
- buffer

bath

graph

- A
- P
- You
- want

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

You want to take a long bath.
Example

⇒ SWAP

stack

buffer

You

bath

take

a

long

to

F

C

P

D

P

F

You

want

A

graph
Example

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

stack

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

buffer

| You | bath |

graph

A \quad P \quad A

You \quad want

A

F

P

to

take

D

long
Example

⇒ REDUCE

stack

buffer

You

bath

diagram:

graph

You

want

to

take

a

long

You

P

A

P

A

D

C

F

F

P
Example

⇒ REDUCE

You want to take a long bath

Graph

Stack

Buffer

You

bath
Example

⇒ SHIFT

stack

You

buffer

You

A

want

P

A

to

F

P

D

take

long

A

C

F
Example

⇒ SHIFT

stack

buffer

You

You

want

take a

to

A

P

A

F

P

C

F

D

long
Example

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

stack

buffer

You

You

want

long

take

A

P

A

F

P

D

C

F

A

bath
Example

⇒ **SHIFT**

You want to take a long bath.

![Graph diagram]

**Stack**

- You
- bath

**Buffer**

A

**Graph**

- You
- want
- to
- take
- a
- long
Example

⇒ $\text{RIGHT-EDGE}_C$

stack

buffer

You \[\bullet\] bath

A

P

A

F

P

D

You

want

to

take

a

long

bath
Example

⇒ Finish

You want to take a long bath.
Training

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

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

\[
\begin{align*}
\text{Shift, Right-Edge}_A, \quad \text{Shift, Swap, Right-Edge}_P, \quad \text{Reduce, Shift, Shift, Node}_F, \quad \text{Reduce, Shift, Right-Edge}_P, \quad \text{Shift, Right-Edge}_F, \quad \text{Reduce, Shift, Swap, Right-Edge}_D, \quad \text{Reduce, Swap, Right-Edge}_A, \quad \text{Reduce, Reduce, Shift, Shift, Left-Remote}_A, \quad \text{Shift, Right-Edge}_C, \quad \text{Finish}
\end{align*}
\]
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.

![Diagram of LSTM network]
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).

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).
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 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.

```
<table>
<thead>
<tr>
<th></th>
<th>LP</th>
<th>LR</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary:</td>
<td>6/9 = 67%</td>
<td>6/10 = 60%</td>
<td>64%</td>
</tr>
<tr>
<td>Remote:</td>
<td>1/2 = 50%</td>
<td>1/1 = 100%</td>
<td>67%</td>
</tr>
</tbody>
</table>
```
Results

$\text{TUPA}_{\text{BiLSTM}}$ 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>$\text{TUPA}_{\text{Sparse}}$</td>
<td>64.5</td>
<td>63.7</td>
<td>64.1</td>
</tr>
<tr>
<td>$\text{TUPA}_{\text{MLP}}$</td>
<td>65.2</td>
<td>64.6</td>
<td>64.9</td>
</tr>
<tr>
<td>$\text{TUPA}_{\text{BiLSTM}}$</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></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\textsubscript{Sparse}</td>
<td>59.6</td>
<td>59.9</td>
<td>59.8</td>
<td>22.2</td>
</tr>
<tr>
<td>TUPA\textsubscript{MLP}</td>
<td>62.3</td>
<td>62.6</td>
<td>62.5</td>
<td>20.9</td>
</tr>
<tr>
<td>TUPA\textsubscript{BiLSTM}</td>
<td>68.7</td>
<td>68.5</td>
<td>\textbf{68.6}</td>
<td>38.6</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td></td>
<td>(91.3)</td>
<td></td>
<td>(43.4)</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>(91.3)</td>
<td></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>(100)</td>
<td></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: LP 76.9, LR 72.1, LF 74.4
- center: LP 76.9, LR 77.3, LF 77.1
- connector: LP 90, LR 76.8, LF 82.9
- elaborator: LP 69.8, LR 73.5, LF 71.6
- function: LP 69.3, LR 81.2, LF 74.7

- linker: LP 69.6, LR 71.7, LF 70.6
- linked scene: LP 61.2, LR 59.3, LF 60.2
- participant: LP 69.3, LR 65.6, LF 67.4
- process: LP 73.4, LR 71.1, LF 72.2
- relator: LP 91.8, LR 92.4, LF 92.1
- state: LP 43, LR 26.9, LF 33.1
Online Demo


Input text:

This is a demo for TUPA, a transition-based UCCA parser.

Parse

Download
Error Analysis

Copular clauses tend to be parsed as identity.

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

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

\(^2\text{http://www.cs.huji.ac.il/~oabend/ucca/guidelines.pdf}\)
Error Analysis

The *participant* category is used when *adverbial* should be.
Future Work
Broad-Coverage UCCA Parsing

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.
AMR Parsing

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.
AMR Parsing

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.
Semantic Dependency Parsing

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

SDP graph (in the DM formalism).

After graduation , John moved to Paris
Semantic Dependency Parsing

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

SDP graph 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: 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).
- Broad coverage UCCA parsing.
- Parsing other schemes, such as AMR and SDP.
- 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).
- Broad coverage UCCA parsing.
- Parsing other schemes, such as AMR and SDP.
- 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.

Uccaapp: Web-application for syntactic and semantic phrase-based annotation.

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
References III

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.

Transition-based dependency parsing with rich non-local features.
Backup
## UCCA Corpora

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>20K Leagues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Dev</td>
</tr>
<tr>
<td># passages</td>
<td>300</td>
<td>34</td>
</tr>
<tr>
<td># sentences</td>
<td>4268</td>
<td>454</td>
</tr>
<tr>
<td># nodes</td>
<td>298,993</td>
<td>33,704</td>
</tr>
<tr>
<td>% terminal</td>
<td>42.96</td>
<td>43.54</td>
</tr>
<tr>
<td>% non-term.</td>
<td>58.33</td>
<td>57.60</td>
</tr>
<tr>
<td>% discont.</td>
<td>0.54</td>
<td>0.53</td>
</tr>
<tr>
<td>% reentrant</td>
<td>2.38</td>
<td>1.88</td>
</tr>
<tr>
<td># edges</td>
<td>287,914</td>
<td>32,460</td>
</tr>
<tr>
<td>% primary</td>
<td>98.25</td>
<td>98.75</td>
</tr>
<tr>
<td>% remote</td>
<td>1.75</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Average per non-terminal node

<table>
<thead>
<tr>
<th># children</th>
<th>Wiki</th>
<th>20K Leagues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.67</td>
<td>1.68</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 $W = \{w_1, \ldots, w_n\}$:

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

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.