Online Graph Planarisation for Synchronous Parsing of Semantic and Syntactic Dependencies

Ivan Titov
University of Illinois at Urbana-Champaign

James Henderson, Paola Merlo, Gabriele Musillo
University of Geneva
Motivation / Problem Statement

- NLP applications will require a shallow representation of meaning
  - Often shallow semantic structures can be regarded as labeled directed graphs

Can we apply methods for syntactic dependency parsing to predict these trees?
Motivation / Problem Statement

- NLP applications will require a shallow representation of meaning
  - Often shallow semantic structures can be regarded as labeled directed graphs

Syntactic structure
- Many crossing arcs (‘non-planarity’)

Semantic structure
- Often little or no crossing links

Sequa makes and repairs jet engines
Motivation / Problem Statement

- NLP applications will require a shallow representation of meaning
  - Often shallow semantic structures can be regarded as labeled directed graphs

  - How can we deal with more general graphs representing semantic structures?
  - How can we construct an effective semantic parser on the basis of an existing syntactic parser?
Outline

- Motivation / Problem Statement
- Background
  - Dependency parsing
  - Properties of dependency graphs
- Non-Planar Parsing using Swapping
- Synchronous Parsing of Semantic and Syntactic Dependencies
  - Synchronization
  - Statistical Model
- Experiments
- Conclusions and Future Directions
Dependency Parsing Problem

- Parsing: given sentence \( x \in X \) predict structure \( y \in Y \):
  \[
  \hat{y} = \arg\max_y F(y, x|w)
  \]

- Graph-based methods: assume that features factorize over subgraphs of \( y \) (e.g., edges) [Eisner, 96; McDonald et al., 05]
  
- Transition-based methods [Yamada and Matsumoto, 03; Nivre et al., 04]:
  - Define derivation order for structures, i.e. mapping from \( y \) to sequences of decisions \( (d_1, \ldots, d_{n(y)}) \)
  - Learn to score individual decision given preceding decisions:
    \[
    F(y, x|w) = \sum_{i=1}^{n(y)} f(d_i|d_1, \ldots, d_{i-1}, x, w)
    \]
  - Decode greedily:
    \[
    \hat{d}_i = \arg\max_{d_i} f(d_i|\hat{d}_1, \ldots, \hat{d}_{i-1}, x, w)
    \]
  - Model estimated on a labeled dataset (treebank)
  - Beam-search can be used instead of greedy decoding
  - That will be the main focus of the talk
Properties of Semantic Structures

- Semantic structures are not trees
  - Graph-based (GB) methods based on maximum spanning tree algorithms are not directly applicable
- Semantic structures are not planar
  - Definition: planar graphs can be drawn in the semi-plane above the sentence without any two arcs crossing and without changing the order of words

```
Sequa makes and repairs jet engines
```
Properties of Semantic Structures

- Semantic structures are not trees
  - Graph-based (GB) methods based on maximum spanning tree algorithms are not directly applicable
- Semantic structures are not planar
  - **Definition**: planar graphs can be drawn in the semi-plane above the sentence without any two arcs crossing and without changing the order of words
  - Most transition-based (TB) algorithms handle only planar graphs
- Related work:
  - [Attardi, 06]: TB method with extended derivation order to handle non-planarity
  - [Nivre, 08]: Assumes that a structure can be made planar by changing order of words (not true for general non-tree graphs)
  - ...
Properties of Semantic Structures

- Semantic structures are not trees
  - Graph-based (GB) methods based on maximum spanning tree algorithms are not directly applicable.
- Semantic structures are not planar
  - Definition: planar graphs can be drawn in the semi-plane above the sentence without any two arcs crossing and without changing the order of words.
- Most transition-based (TB) algorithms handle only planar graphs

Related work:
- [Attardi, 06]: TB method with extended derivation order to handle non-planarity
- [Nivre, 08]: Assumes that a structure can be made planar by changing order of words (not true for general non-tree graphs)
- ...

We will propose a very simple technique to extend standard TB methods to handle non-planar and not-tree structured graphs.
Outline

- Motivation / Problem Statement
- Background
  - Dependency parsing
  - Properties of dependency graphs
- Non-Planar Parsing using Swapping
- Synchronous Parsing of Semantic and Syntactic Dependencies
  - Synchronization
  - Statistical Model
- Experiments
- Conclusions and Future Directions
Derivation order \([Nivre, 04]\)

- **State of the parser after steps** \((d_1, \ldots, d_{i-1})\) is characterized by:
  - current stack \(S\) (\(w_j\) – word on top of the stack)
  - a queue \(I\) of remaining input words (\(w_k\) – next input word)
  - partial dependency structure defined by \((d_1, \ldots, d_{i-1})\)

- **New decision** \(d_i\) can be
  - \(\text{LeftArc}_r\) - adds a labeled dependency arc \(w_j \xrightarrow{r} w_k\)
  - \(\text{RightArc}_r\) - adds a labeled dependency arc \(w_j \xrightarrow{r} w_k\)
  - \(\text{Shift}\) - moves \(w_k\) from queue to the stack
  - \(\text{Reduce}\) - remove \(w_j\) from the stack

- Terminates when the queue is empty
Handling non-planar structures

- \[\text{[Nivre, 04]}\] order cannot handle non-planar structures:
  - All the arcs are created between top of the stack and front of queue
  - Words are stored in the stack in the same order as they appear in the sentence

- A single new decision:
  - Swap - swaps 2 top words in the stack
    
    Stack before:  \[S = [\ldots, w_m, w_j]\]  
    Stack after:  \[S = [\ldots, w_j, w_m]\]
Example

Example: Sequa makes and repairs jet engines.
Example

- Partial Structure:

  Sequa makes and repairs jet engines

  $S = \left[ \right]$

  $l = \left[ \text{Sequa makes and } .... \right]$

- Next action: Shift
Example

- Partial Structure:

\[ \text{Sequa makes and repairs jet engines} \]

- \[ S = [ \text{Sequa} ] \]

- \[ l = [ \text{makes and repairs ....} ] \]

- Next action: \( \text{LeftArc}_\text{AGENT} \)
Example

- Partial Structure:

  > Sequa makes and repairs jet engines

  - $S = \text{[ Sequa ]}$
  - $l = \text{[ makes and repairs .... ]}$

- Next action: *Shift*
Example

- Partial Structure:

Sequa makes and repairs jet engines

- S = [ Sequa makes ]
- l = [ and repairs jet engines ]

- Next action: Shift
Example

- Partial Structure:

Sequa makes and repairs jet engines

- $S = [\text{Sequa makes and}]$
- $l = [\text{repairs jet engines}]$

- Next action: Reduce
Example

- Partial Structure:

\[ S = [ \text{Sequa makes } ] \]

\[ I = [ \text{repairs jet engines } ] \]

- Next action: Swap

Stalled if without swap:
- \text{repairs} needs an arc to Sequa
- but \text{makes} cannot be removed from stack
Example

- Partial Structure:

  - $S = [\text{makes Sequa}]$
  - $I = [\text{repairs jet engines}]$

- Next action: $\text{LeftArc}_{\text{AGENT}}$
Example

- Partial Structure:

  \[ S = \text{[makes Sequa]} \]
  \[ l = \text{[repairs jet engines]} \]

- Next action: Reduce
Example

- Partial Structure:

  - $S = \text{[ makes ]}$
  - $I = \text{[ repairs jet engines ]}$

- Next action: Shift
Example

- Partial Structure:

  - $S = \{ \text{makes repairs} \}$
  - $I = \{ \text{jet engines} \}$

- Next action: *Shift*
Example

- Partial Structure:

- $S = [\text{makes repairs jet}]$

- $I = [\text{engines}]$

- Next action: Reduce
Example

- Partial Structure:

- $S = \text{[ makes repairs ]}$

- $l = \text{[ engines ]}$

- Next action: $\text{RightArc}_{\text{PATIENT}}$
Example

- Partial Structure:

  - **S = [ makes repairs ]**
  - **l = [ engines ]**

  - Next action: *Reduce*
Example

- Partial Structure:

- \( S = [ \text{makes} ] \)

- \( l = [ \text{engines} ] \)

- Next action: \( \text{RightArc}_{\text{PATIENT}} \)
Example

- Partial Structure:

  - $S = [\text{makes}]$
  - $I = [\text{engines}]$

- Next action: Shift
Example

- Partial Structure:

- $S = [\text{makes engines}]$

- $l = []$

- Next action: *Stop*
Canonical orders

- The algorithm allows the same structure to be parsed multiple ways
  - Summing over all the possible derivation when parsing is not feasible
  - Instead, models are trained to produce derivations in a canonical way
- In other words, canonical orders define how derivations should be produced for the training set
- We consider two canonical derivations which differ in when swapping is done
  - last-resort: Swap is used as a last resort when no other operation is possible
  - exhaustive: Swap is used pre-emptively
Last-Resort ordering

- Swap is used as a last resort when no operation is possible
- Drawback: not all the structures parsable with swapping have a last-resort derivation
  - in CoNLL-2008 dataset 2.8% fewer structures are parsable with last-resort ordering
  - An example of such a structure:

Suddenly CDC and DEC have products

- Advantage: this canonical derivation is predictable and, therefore, supposedly easier to learn
Exhaustive ordering

- **Algorithm for preemptive swapping:**
  - Ordering follows standard planar parser ordering until no other operation except *Shift* and *Swap* are possible
  - Compute the ordered list of positions of words in the queue to which current top of the stack \( w_j \) will be connected
  - Compute a similar list for word \( w_m \) under the top of the stack
  - Swap if \( w_m \)'s list precedes \( w_j \)'s list in their lexicographical order

\[ S = [ \text{Suddenly} \ CDC ] \]
\[ I = [ \text{DEC} \ldots ] \]

List for ‘Suddenly’: \( \{5\} \)

List for ‘CDC’: \( \{5, 6\} \implies \text{Swap} \]
Exhaustive ordering

- **Algorithm for preemptive swapping:**
  - Ordering follows standard planar parser ordering until no other operation except *Shift* and *Swap* are possible
  - Compute the ordered list of positions of words in the queue to which current top of the stack $w_j$ will be connected
  - Compute a similar list for word $w_m$ under the top of the stack
  - Swap if $w_m$’s list precedes $w_j$’s list in their lexicographical order

- **Theorem**  *If the graph is parsable with the defined set of operations then the exhaustive ordering is guaranteed to find a derivation*
  - See the paper for the proof sketch
Not all the non-planar graphs are parsable
- In CoNLL-2008 ST dataset only 1% of semantic structures are not parsable whereas 44% are not planar (i.e., require swapping)

Among common linguistic structures requiring Swap are coordinations
- E.g., “Sequa makes, repairs and sells engines”

A frequent example of an unparsable structure:
- Funds also might buy and sell
  - 2 predicates sharing 3 arguments
Structures Parsable with Swapping

- Any structures with isolated pairs of crossing arcs are parsable but they are more powerful than that

**Theorem** A graph cannot be parsed with the defined set of parsing operations iff the graph contains at least one of the subgraphs presented below:

- the unspecified arc end points can be anywhere strictly following those specified
- circled pairs of endpoints can be either a single word or two distinct words

See the paper for the proof sketch
Outline

- Motivation / Problem Statement
- Background
  - Dependency parsing
  - Properties of dependency graphs
- Non-Planar Parsing using Swapping
- Synchronous Parsing of Semantic and Syntactic Dependencies
  - Synchronization
  - Statistical Model
- Experiments
- Conclusions and Future Directions
Synchronization [Henderson et al., 2008]

- We define **two separate derivations**: one for semantics, one for syntax
- Instead of using pipelines we synchronize these two derivations
  - joint learning and joint inference
Example:

Sequa makes and repairs jet engines
Synchronization

- Example:

Sequa **makes** and **repairs** jet engines
Synchronization

- Example:

Sequa makes and repairs jet engines

Semantic structure

Syntactic structure
Synchronization

Example:

Sequa makes and repairs jet engines

$D^2_{syn}$
Synchronization

Example:

Sequa makes and repairs jet engines

Semantics:

Syntactics:
Synchronization

Example:

Sequa makes and repairs jet engines

Semantic structure

Syntactic structure
Synchronization

Example:

Sequa makes and repairs jet engines
Synchronization

Example:

Sequa makes and repairs jet engines
Synchronization

- Example:

Sequa makes and repairs jet engines

**Semantic structure**

**Syntactic structure**
Synchronization

Example:

Sequa makes and repairs jet engines
Synchronization

Example:

Sequa makes and repairs jet engines
Following [Henderson, et al., 08], the synchronous derivations are modeled with Incremental Sigmoid Belief Networks (ISBNs) [Titov and Henderson, 07]

- previously successfully applied to constituent and dependency syntactic parsing

- Vectors of latent variables are associated with each parsing decision

- Each vector is connected with previous vectors by a pattern of interconnections determined by the previous decisions
Outline

- Motivation / Problem Statement
- Background
  - Dependency parsing
  - Properties of dependency graphs
- Non-Planar Parsing using Swapping
- Synchronous Parsing of Semantic and Syntactic Dependencies
  - Synchronization
  - Statistical Model
- Experiments
- Conclusions and Future Directions
Empirical Evaluation

- CoNLL-2008 Shared Task data [Surdeanu et al., 08], merged
  - dependency transformation of Penn Treebank WSJ (syntax)
  - dependency representation of Propbank and Nombank (semantics)

- Data: 39,279/1,334/2,824 sentences for training/development/testing

- Systems
  - Our models
    - exhaustive order
    - last-resort order
    - planar order (can only process projective parts of derivations)
  - [Henderson et al., 08] planarisation: a modification of [Nivre and Nilsson, 05] method for semantic graphs (HEAD)
    - Crossing links are removed and encoded in labels of remaining arcs
  - Model structure and estimation methods are kept constant
## Results on Development Set

<table>
<thead>
<tr>
<th>Technique</th>
<th>CoNLL Measures</th>
<th>Crossing Paris (Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syn LAS</td>
<td>Sem FI</td>
</tr>
<tr>
<td>Last resort</td>
<td>86.6</td>
<td>76.2</td>
</tr>
<tr>
<td>Exhaustive</td>
<td>86.8</td>
<td>76.0</td>
</tr>
<tr>
<td>HEAD</td>
<td>86.7</td>
<td>73.3</td>
</tr>
<tr>
<td>Planar</td>
<td>85.9</td>
<td>72.8</td>
</tr>
</tbody>
</table>
## Results on Development Set

<table>
<thead>
<tr>
<th>Technique</th>
<th>CoNLL Measures</th>
<th>Crossing Paris (Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syn LAS</td>
<td>Sem FI</td>
</tr>
<tr>
<td>Last resort</td>
<td>86.6</td>
<td>76.2</td>
</tr>
<tr>
<td>Exhaustive</td>
<td>86.8</td>
<td>76.0</td>
</tr>
<tr>
<td>HEAD</td>
<td>86.7</td>
<td>73.3</td>
</tr>
<tr>
<td>Planar</td>
<td>85.9</td>
<td>72.8</td>
</tr>
</tbody>
</table>
## Results on Development Set

<table>
<thead>
<tr>
<th>Technique</th>
<th>CoNLL Measures</th>
<th>Crossing Pairs (Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syn LAS</td>
<td>Sem F1</td>
</tr>
<tr>
<td>Last resort</td>
<td>86.6</td>
<td>76.2</td>
</tr>
<tr>
<td>Exhaustive</td>
<td>86.8</td>
<td>76.0</td>
</tr>
<tr>
<td>HEAD</td>
<td>86.7</td>
<td>73.3</td>
</tr>
<tr>
<td>Planar</td>
<td>85.9</td>
<td>72.8</td>
</tr>
</tbody>
</table>
## Results on Development Set

<table>
<thead>
<tr>
<th>Technique</th>
<th>CoNLL Measures</th>
<th>Crossing Pairs (Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syn LAS</td>
<td>Sem F1</td>
</tr>
<tr>
<td>Last resort</td>
<td>86.6</td>
<td>76.2</td>
</tr>
<tr>
<td>Exhaustive</td>
<td>86.8</td>
<td>76.0</td>
</tr>
<tr>
<td>HEAD</td>
<td>86.7</td>
<td>73.3</td>
</tr>
<tr>
<td>Planar</td>
<td>85.9</td>
<td>72.8</td>
</tr>
</tbody>
</table>

The model is generative and, therefore, decisions are not conditioned on future words – a likely reason why no improvement from using exhaustive strategy.
Test Set Results

<table>
<thead>
<tr>
<th>Model</th>
<th>CoNLL Measures</th>
<th>Crossing Pairs (Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syn LAS</td>
<td>Sem FI</td>
</tr>
<tr>
<td>Johanssen</td>
<td>89.3</td>
<td>81.6</td>
</tr>
<tr>
<td>Ciaramita</td>
<td>87.4</td>
<td>78.0</td>
</tr>
<tr>
<td>Che</td>
<td>86.7</td>
<td>78.5</td>
</tr>
<tr>
<td>Zhao</td>
<td>87.7</td>
<td>76.7</td>
</tr>
<tr>
<td>This paper</td>
<td>87.5</td>
<td>76.1</td>
</tr>
<tr>
<td>Henderson+</td>
<td>87.6</td>
<td>73.1</td>
</tr>
<tr>
<td>Lluis</td>
<td>85.8</td>
<td>70.3</td>
</tr>
</tbody>
</table>

- 3% improvement over the baseline on semantics graphs
- However, does not outperform reranking or ensemble techniques
## Test Set Results

<table>
<thead>
<tr>
<th>Model</th>
<th>CoNLL Measures</th>
<th>Crossing Pairs (Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syn LAS</td>
<td>Sem F1</td>
</tr>
<tr>
<td>Johanssen</td>
<td>89.3</td>
<td>81.6</td>
</tr>
<tr>
<td>Ciaramita</td>
<td>87.4</td>
<td>78.0</td>
</tr>
<tr>
<td>Che</td>
<td>86.7</td>
<td>78.5</td>
</tr>
<tr>
<td>Zhao</td>
<td>87.7</td>
<td>76.7</td>
</tr>
<tr>
<td>This paper</td>
<td>87.5</td>
<td>76.1</td>
</tr>
<tr>
<td>Henderson+</td>
<td>87.6</td>
<td>73.1</td>
</tr>
<tr>
<td>Lluis</td>
<td>85.8</td>
<td>70.3</td>
</tr>
</tbody>
</table>

- Recent results: 3\textsuperscript{rd} result in CoNLL-2009 Shared task (7 languages)
Conclusions

- Proposed a simple modification to handle semantic graphs with transition-based parsers
  - though not powerful enough to process all the semantic graphs it is able to handle vast majority
  - proposed and analyzed two algorithms for canonical derivation induction
  - theoretically characterized the class of parsable structures

- Showed improvements over previous methods

- Demonstrated state-of-the-art results without ensemble techniques or pipelines

- Future directions: applying this approach to parsing of language with highly non-planar syntactic structures