Structured Prediction and NLP

**Structured prediction**: a machine learning framework for predicting structured, constrained, and interdependent outputs.

**NLP** deals with *structured* and *ambiguous* textual data:

- machine translation
- speech recognition
- syntactic parsing
- semantic parsing
- information extraction
- ...
Constituent/Phrase-Structure Parsing

S --> NP VP
NP --> Det Adj N
VP --> V NP Adv
Adj --> minimal
Adv --> here
Det --> a
N --> logic
N --> role
V --> plays

Example extracted from the Penn Treebank.

(Magerman, 1995; Charniak, 1996; Johnson, 1998; Collins, 1999; Klein and Manning, 2003)
Dependency Parsing

Map **sentences** to their **syntactic structure**.

- A lexicalized syntactic formalism
- Grammar functions represented as lexical relationships (dependencies)

(Eisner, 1996; McDonald et al., 2005; Nivre et al., 2006; Koo et al., 2007)
Outline

1 Turbo Parsers

2 Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
What is a Turbo Parser?

A parser that runs inference in factor graphs, ignoring global effects caused by loops (Martins et al., 2010). Name inspired from turbo decoders (Berrou et al., 1993).

André Martins (Unbabel/IT)
What is a Turbo Parser?

- A parser that runs inference in factor graphs, ignoring global effects caused by loops (Martins et al., 2010)
- name inspired from *turbo* decoders (Berrou et al., 1993)
Examples of Turbo Parsers

- Exponential-sized ILP formulation (Riedel and Clarke, 2006)
- Polynomial-sized ILP formulation with multi-commodity flows (Martins et al., 2009)
- Belief propagation decoder (Smith and Eisner, 2008; Martins et al., 2010)
- Dual decomposition decoder (Koo et al., 2010)
- AD$^3$ decoder (Martins et al., 2011, 2013)
An Important Distinction

- A projective tree:

  * Logic plays a minimal role here

- A non-projective tree:

  * We learned a lesson in 1987 about volatility

Non-projective trees are suitable for languages with flexible word order (Dutch, German, Czech,...).
An Important Distinction

- A projective tree:

  * Logic plays a minimal role here

- A non-projective tree:

  * We learned a lesson in 1987 about volatility

Non-projective trees are suitable for languages with flexible word order (Dutch, German, Czech,...).
First-Order Scores for Arcs

We learned a lesson in 1987 about volatility
Second-Order Scores for Consecutive Siblings

* We learned a lesson in 1987 about volatility
Second-Order Scores for Grandparents

* We learned a lesson in 1987 about volatility
Scores for Arbitrary Siblings

* We learned a lesson in 1987 about volatility
We learned a lesson in 1987 about volatility.
Third-Order Scores for Grand-siblings

* We learned a lesson in 1987 about volatility
Third-Order Scores for Tri-siblings

* We learned a lesson in 1987 about volatility
Decoding

- arc
- consecutive siblings
- grandparent
- all siblings
- directed path
- head bigram
- nonprojective arc

- grand-siblings
- tri-siblings

- How to deal with all these parts?
Decoding

- How to deal with all these parts?
- Beyond arc-factored models, non-projective parsing is **NP-hard** (McDonald and Satta, 2007)—**need to embrace approximations!**
Decoding

- How to deal with all these parts?
- Beyond arc-factored models, non-projective parsing is \textbf{NP-hard} (McDonald and Satta, 2007)—\textit{need to embrace approximations!}

<table>
<thead>
<tr>
<th></th>
<th>parser</th>
<th>AF</th>
<th>CS</th>
<th>G</th>
<th>AS</th>
<th>DP</th>
<th>HB</th>
<th>NPA</th>
<th>GS</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald et al. (2006)</td>
<td>projective + greedy loopy BP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smith et al. (2008)</td>
<td>loopy BP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martins et al. (2010)</td>
<td>LP solver</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Koo et al. (2010)</td>
<td>dual decomp.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martins et al. (2011)</td>
<td>AD$^3$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Martins et al. (2013)</td>
<td>AD$^3$ &amp; active set</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
For each input $x \in X$: a large set of candidate outputs $y(x)$

Decoding problem:

$$\hat{y} = \arg \max_{y \in y(x)} F_w(x, y)$$
Factor Graph Representations

- For each input $x \in X$: a large set of candidate outputs $y(x)$
- Decoding problem:

$$\hat{y} = \arg \max_{y \in y(x)} F_w(x, y)$$

- Key assumption: $F_w$ decomposes into (overlapping) parts

$$F_w(x, y) := \sum_s f_s(y_s)$$

Examples: HMMs, CRFs, PCFGs, general graphical models
Factor Graph Representations

- For each input $x \in \mathcal{X}$: a large set of candidate outputs $y(x)$
- Decoding problem:

$$\hat{y} = \arg \max_{y \in y(x)} F_w(x, y)$$

- Key assumption: $F_w$ decomposes into (overlapping) parts

$$F_w(x, y) := \sum_s f_s(y_s)$$

- Examples: HMMs, CRFs, PCFGs, general graphical models
Factors as Machines

\[ Y_1 \rightarrow Y_2 \rightarrow Y_3 \rightarrow Y_4 \rightarrow Y_5 \]
Factors as Machines

\[ Y_1 \quad Y_2 \quad Y_3 \quad Y_4 \quad Y_5 \]
Alternating Directions Dual Decomposition (AD$^3$)

A general purpose algorithm, suitable for many scenarios in NLP and IR.
Alternating Directions Dual Decomposition (\(\text{AD}^3\))

A general purpose algorithm, suitable for many scenarios in NLP and IR.

- Problems with factor graph representations
Alternating Directions Dual Decomposition (AD³)

A general purpose algorithm, suitable for many scenarios in NLP and IR.

- Problems with factor graph representations
- Statements in FOL
Alternating Directions Dual Decomposition (AD$^3$)

A general purpose algorithm, suitable for many scenarios in NLP and IR.

- Problems with factor graph representations
- Statements in FOL
- Budget/knapsack constraints
Alternating Directions Dual Decomposition (AD³)

A general purpose algorithm, suitable for many scenarios in NLP and IR.

- Problems with factor graph representations
- Statements in FOL
- Budget/knapsack constraints
- Combination of structured models
Alternating Directions Dual Decomposition (AD³)

A general purpose algorithm, suitable for many scenarios in NLP and IR.

- Problems with factor graph representations
- Statements in FOL
- Budget/knapsack constraints
- Combination of structured models

**High level idea:**

- Decompose a complex problem into local subproblems (factors), constrained to be globally consistent
- Iterate between solving the local subproblems and penalizing the global disagreements (via Lagrange multipliers)
- FOL/knapsack constraints: the local subproblems correspond to projections onto “hard constraint” polytopes
All projections can be computed in linear time (Martins et al., 2015)

- **Applications**: Markov logic networks (Richardson and Domingos, 2006), constrained conditional models (Roth and Yih, 2004), summarization (Almeida and Martins, 2013), ...
Some Problems in Which AD³ Have Been Applied

- Dependency parsing (Martins et al., 2011, 2013)
- Frame semantics (Das et al., 2012)
- Broad-coverage semantic parsing (Martins and Almeida, 2014)
- Compressive summarization (Almeida and Martins, 2013)
- Coreference resolution (Almeida et al., 2014)

Could be a great fit to many other applications!!
Literature Pointers

- André F. T. Martins.
  “AD³: A Fast Decoder for Structured Prediction.”
  Book chapter of *Advanced Structured Prediction*,
  Sebastian Nowozin, Peter V. Gehler, Jeremy Jancsary, and Christoph H. Lampert (Editors),

  “AD³: Alternating Directions Dual Decomposition for MAP Inference in Graphical Models.”
  JMLR 2015.

More details: EMNLP 2014 tutorial on “LP Decoders for NLP.”
SOTA accuracies for the largest non-projective datasets (CoNLL-2006 and CoNLL-2008):

<table>
<thead>
<tr>
<th>Language</th>
<th>Method</th>
<th>Accuracy</th>
<th>TPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN (English)</td>
<td>Koo et al. (2011)</td>
<td>92.57</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>Martins et al. (2011b)</td>
<td>92.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>93.22</td>
<td>785 toks/sec</td>
</tr>
<tr>
<td>NL (Dutch)</td>
<td>Koo et al. (2011)</td>
<td>85.81</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>Martins et al. (2011b)</td>
<td>85.53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>86.19</td>
<td>599 toks/sec</td>
</tr>
<tr>
<td>DE (German)</td>
<td>Martins et al. (2011b)</td>
<td>91.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rush &amp; Petrov (2012)</td>
<td>90.8</td>
<td>2,880</td>
</tr>
<tr>
<td></td>
<td>Zhang &amp; McDonald (2012)</td>
<td>91.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>92.41</td>
<td>965 toks/sec</td>
</tr>
<tr>
<td>CZ (Czech)</td>
<td>Martins et al. (2010)</td>
<td>88.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Martins et al. (2011b)</td>
<td>89.46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>90.32</td>
<td>501 toks/sec</td>
</tr>
</tbody>
</table>
Extension: Broad-Coverage Semantic Parsing

Same idea applied to **semantic role labeling**.

Best results in the SemEval 2014 shared task:

Try It Yourself: AD$^3$ Toolkit

Freely available at: http://www.ark.cs.cmu.edu/AD3

Implemented in C++, includes a Python wrapper (thanks to Andy Mueller)

Many built-in factors: logic, knapsack, dense, and some structured factors

You can implement your own factor (only need to write a local MAP decoder!)

Toy examples included (parsing, coreference, Potts models)
Try It Yourself: TurboParser

- Freely available at: http://www.ark.cs.cmu.edu/TurboParser
- Implemented in C++, includes a Python wrapper
- Not just parsing, but a full NLP pipeline now!
- Includes multilingual POS tagging, dependency parsing, semantic role labeling, entity recognition, coreference resolution (all trainable on any dataset).
Outline

1 Turbo Parsers

2 Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
Constituent parsers are slow (heavy grammar constant)
Dependency parsers are faster, but their output is less informative
How to get the best of both worlds?
Constituent parsers are slow (heavy grammar constant)
Dependency parsers are faster, but their output is less informative
How to get the best of both worlds?

Our proposal: a reduction of constituent parsing to dependency parsing
In a Nutshell (Fernández-González and Martins, 2015, ACL)

- Constituent parsers are slow (heavy grammar constant)
- Dependency parsers are faster, but their output is less informative
- How to get the best of both worlds?

**Our proposal:** a reduction of constituent parsing to dependency parsing

- Rooted in a novel formalism: **head-ordered dependency trees**
- Works for **any out-of-the-box dependency parser**
- Competitive for English and morphologically rich languages
- Results above the state of the art for **discontinuous parsing**
Outline

1 Turbo Parsers

2 Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
Outline

1 Turbo Parsers

2 Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
Continuous and Discontinuous C-Trees

- CFG generate **continuous** trees, LCFRS generate **discontinuous** trees (Vijay-Shanker et al., 1987)
- ... but existing discontinuous parsers are too slow and inaccurate!
Continuous and discontinuous c-trees “project” respectively to **projective** and **non-projective** d-trees (Gaifman, 1965)

Non-projectiveness is suitable for languages with flexible word order (Dutch, German, Czech, etc.)
Projecting C-Trees onto D-Trees...

The public is still cautious.
Projecting C-Trees onto D-Trees...

1. apply set of head rules:

   - $S \rightarrow NP \ VP$
   - $NP \rightarrow DT \ NN$
   - $VP \rightarrow VBZ \ ADVP \ ADJP$
   - ...

The public is still cautious.
Projecting C-Trees onto D-Trees...

1. apply set of head rules:
   
   \[
   \begin{align*}
   S & \rightarrow NP \ VP \\
   NP & \rightarrow DT \ NN \\
   VP & \rightarrow VBZ \ ADVP \ ADJP \\
   \end{align*}
   \]

2. lexicalize

The public is still cautious.
Projecting C-Trees onto D-Trees...

```
The public is still cautious.
```

1. apply set of head rules:
   
   \[
   \begin{align*}
   S & \rightarrow \text{NP VP} . \\
   \text{NP} & \rightarrow \text{DT NN} \\
   \text{VP} & \rightarrow \text{VBZ ADVP ADJP} \\
   \end{align*}
   \]

2. lexicalize
3. drop constituent nodes
Projecting C-Trees onto D-Trees...

The public is still cautious.
... And Back?

The public is still cautious.
... And Back?

This paper: formal equivalence results to "invert" this projection.

André Martins (Unbabel/IT)

Turbo Parser Redux

LxMLS, Lisboa, 26/07/16
... And Back?

left-branch? right-branch? flat?
... And Back?

This paper: formal equivalence results to “invert” this projection.
Related Work

- Store structural information in the dependency labels (Hall and Nivre, 2008)
- Manual transformation rules toward multi-representational treebanks (Xia and Palmer, 2001; De Marneffe et al., 2006; Xia et al., 2008)
- Apply second-stage constituent parser (Kong et al., 2015)
- Joint dependency and constituent parsing (Carreras et al., 2008; Rush et al., 2010)
Outline

1 Turbo Parsers

2 Parsing as Reduction

- Dependencies and Constituents
- Head-Ordered Dependency Trees
- Reduction-Based Constituent Parsers
- Experiments
- Conclusions
Strictly Ordered D-Trees

Key idea: endow d-trees with additional structure, by making each head attach its modifiers in a particular order
Strictly Ordered D-Trees

**Key idea:** endow d-trees with additional structure, by making each head attach its modifiers in a particular order

```
NN
  ┌─VBZ─JJ─public─is─cautious
  │   │         │    │    │    ┌─NN─VBZ─JJ─public─is─cautious
```
Strictly Ordered D-Trees

**Key idea:** endow d-trees with additional structure, by making each head attach its modifiers in a particular order.

```
    public is cautious
    \  /     \  /
   VBZ    JJ
   \         /   \\
  NN       VP#1
```

```
    public is cautious
    \  /     \  /
   NN    VBZ JJ
```

Proposition: Binary c-trees = strictly ordered d-trees

Same number of symbols (dependency alphabet = phrasal alphabet)
Strictly Ordered D-Trees

**Key idea:** endow d-trees with additional structure, by making each head attach its modifiers in a particular order

```
S
   /\  \
NN  VP
   /   /
VBZ  JJ
```

```
S#2
   /
public
   /
is
   /
cautious
```

```
VP#1
   /
NN
   /
VBZ
   /
JJ
```
Strictly Ordered D-Trees

Key idea: endow d-trees with additional structure, by making each head attach its modifiers in a particular order

Proposition

Binary c-trees = strictly ordered d-trees
Strictly Ordered D-Trees

**Key idea:** endow d-trees with additional structure, by making each head attach its modifiers in a particular order

![D-Tree Diagram]

**Proposition**

**Binary c-trees = strictly ordered d-trees**

- Same number of symbols (dependency alphabet = phrasal alphabet)
The Spinal View

The order is given by the attachment position in the spine (Carreras et al., 2008)
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)

```
  NN
    | VBZ  | RB  | JJ
  public   is   still   cautious
```

Proposition

Unaryless c-trees = weakly ordered d-trees
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)

\[
\text{NN} \quad \text{VP} \\
\quad \text{VBZ} \quad \text{RB} \quad \text{JJ} \\
\text{public} \quad \text{is} \quad \text{still} \quad \text{cautious}
\]
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)

![Diagram of a weakly ordered D-Tree]

- Can every c-tree be represented like this?
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)

- Can every c-tree be represented like this? **No: unaries are lost.**
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)

![Diagram of a weakly ordered D-tree]

- Can every c-tree be represented like this? **No: unaries are lost.**
Weakly Ordered D-Trees

Same, but allow **simultaneous** events (as long as the d-label is consistent)

Can every c-tree be represented like this? **No: unaries are lost.**

**Proposition**

*Unaryless c-trees = weakly ordered d-trees*
What About Projective Trees?

A head-ordered d-tree has the **nesting property** if, on each side of every head, closer modifiers are attached first.
A head-ordered d-tree has the **nesting property** if, on each side of every head, closer modifiers are attached first.
What About Projective Trees?

A head-ordered d-tree has the **nesting property** if, on each side of every head, closer modifiers are attached first.

**Proposition**

*Unaryless continuous c-trees = nested-weakly ordered projective d-trees*
Projective, but not nested:

\[
\begin{align*}
\text{Nested, but not projective:}
\end{align*}
\]
Outline

1 Turbo Parsers

2 Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
Reduction-Based Constituent Parsers

1. Convert c-treebank to head-ordered d-treebank.
2. Encode head-orders in the d-labels, yielding a d-treebank.
3. Train a d-parser on the d-treebank.
4. Run the d-parser on new sentences.
5. Convert the predicted d-trees into unaryless c-trees.
6. Recover unary nodes.
Reduction-Based Constituent Parsers

1. Convert c-treebank to head-ordered d-treebank. ✔
2. Encode head-orders in the d-labels, yielding a d-treebank.
3. Train a d-parser on the d-treebank.
4. Run the d-parser on new sentences. ✔
5. Convert the predicted d-trees into unaryless c-trees. ✔
6. Recover unary nodes.
Label Encoding Strategies

direct encoding

The public is still cautious.

NP#1  S#2  VP#1  S#2  VP#1
DT    NN   VBZ  RB   JJ   .
Label Encoding Strategies

**direct encoding**

```
The public is still cautious.
```

**delta encoding**

```
The public is still cautious.
```

```
NP#1   S#2     VP#1       S#2
DT     NN      VBZ        RB    JJ
```

```
NP#1   S#2     VP#1       S#0+1
DT     NN      VBZ        RB    JJ
```
Label Encoding Strategies

direct encoding

```
NP#1  S#2
The   public
DT    NN

VP#1
is   still  cautious
VBZ   RB   JJ

S#2
```

delta encoding

```
NP#1  S#2
The   public
DT    NN

VP#1
is   still  cautious
VBZ   RB   JJ

S#+1
```

H&N encoding (Hall and Nivre, 2008)

```
$  NP#2
The  public
DT   NN

*#1
is   still  cautious
VBZ   RB   JJ

*#2
```

```
ADJP#1
```

```
ADVP#1
```
Impact of Label Encoding

- Evaluated on the English PTB §22 (Marcus et al., 1993).

<table>
<thead>
<tr>
<th></th>
<th># labels</th>
<th>dep (LAS)</th>
<th>const (F₁)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H&amp;N encoding</td>
<td>731</td>
<td>87.86</td>
<td>89.39</td>
</tr>
<tr>
<td>Direct encoding</td>
<td>75</td>
<td>91.99</td>
<td>90.89</td>
</tr>
<tr>
<td><strong>Delta encoding</strong></td>
<td>69</td>
<td><strong>92.00</strong></td>
<td><strong>90.94</strong></td>
</tr>
</tbody>
</table>

- H&N encoding overgenerates labels, leading to a loss in accuracy
- Delta encoding performs consistently better than direct encoding on other datasets (see paper)
Reduction-Based Constituent Parsers

1. Convert c-treebank to head-ordered d-treebank.
2. Encode head-orders in the d-labels, yielding a d-treebank.
3. Train a d-parser on the d-treebank.
4. Run the d-parser on new sentences.
5. Convert the predicted d-trees into unaryless c-trees.
6. Recover unary nodes.
Choice of Dependency Parser

- Evaluated on the English PTB §22 (Marcus et al., 1993).

<table>
<thead>
<tr>
<th>Dependency Parser</th>
<th>Dep (LAS)</th>
<th>Const (F₁)</th>
<th># toks/s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaltParser</td>
<td>88.95</td>
<td>86.87</td>
<td>5,392</td>
</tr>
<tr>
<td>MSTParser</td>
<td>89.86</td>
<td>87.93</td>
<td>363</td>
</tr>
<tr>
<td>ZPar</td>
<td>91.28</td>
<td>89.50</td>
<td>1,022</td>
</tr>
<tr>
<td>TurboParser-Basic</td>
<td>90.23</td>
<td>87.63</td>
<td>2,585</td>
</tr>
<tr>
<td>TurboParser-Standard</td>
<td>91.58</td>
<td>90.41</td>
<td>1,658</td>
</tr>
<tr>
<td>TurboParser-Full</td>
<td>91.70</td>
<td>90.53</td>
<td>959</td>
</tr>
<tr>
<td>TurboParser-Full + Labeler</td>
<td>92.00</td>
<td>90.94</td>
<td>912</td>
</tr>
</tbody>
</table>

- Best results: separate stages for d-parser and d-labeler
- The d-labeler is a simple sequence model for each head (see paper)
Reduction-Based Constituent Parsers

1. Convert c-treebank to head-ordered d-treebank.
2. Encode head-orders in the d-labels, yielding a d-treebank.
3. Train a d-parser on the d-treebank.
4. Run the d-parser on new sentences.
5. Convert the predicted d-trees into unaryless c-trees.
6. Recover unary nodes.
Recovery of Unary Nodes

- We run independent classifiers at each c-node
- Each class is either **NULL** (no unary node pre-appended) or a concatenation of labels (e.g., \textit{S$\rightarrow$ADJP} for a node \textit{JJ})
- To speed-up: only observed classes are considered (9.9 classes per node in PTB §22)
- A tiny fraction of the time is spent on this post-processing (<2%), with $F_1$-score of 99.43% in PTB §22
Outline

1. Turbo Parsers

2. Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
Experiments: English PTB

Results on the English PTB §23 (Marcus et al., 1993).

<table>
<thead>
<tr>
<th>Parser</th>
<th>LR</th>
<th>LP</th>
<th>F1</th>
<th>#Toks/s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klein and Manning (2003)</td>
<td>85.3</td>
<td>86.5</td>
<td>85.9</td>
<td>143</td>
</tr>
<tr>
<td>Hall et al. (2014)</td>
<td>88.4</td>
<td>88.8</td>
<td>88.6</td>
<td>12</td>
</tr>
<tr>
<td>Socher et al. (2013)</td>
<td>89.1</td>
<td>89.7</td>
<td>89.4</td>
<td>70</td>
</tr>
<tr>
<td>Charniak (2000)</td>
<td>89.5</td>
<td>89.9</td>
<td>89.5</td>
<td>–</td>
</tr>
<tr>
<td>Stanford Shift-Reduce (2014)</td>
<td>89.1</td>
<td>89.1</td>
<td>89.1</td>
<td>655</td>
</tr>
<tr>
<td>Petrov and Klein (2007)</td>
<td>90.0</td>
<td>90.3</td>
<td>90.1</td>
<td>169</td>
</tr>
<tr>
<td><strong>This work</strong></td>
<td>89.9</td>
<td>90.4</td>
<td>90.2</td>
<td>957</td>
</tr>
<tr>
<td>Zhu et al. (2013)</td>
<td>90.3</td>
<td>90.6</td>
<td>90.4</td>
<td>1,290</td>
</tr>
<tr>
<td>Carreras et al. (2008)</td>
<td>90.7</td>
<td>91.4</td>
<td>91.1</td>
<td>–</td>
</tr>
<tr>
<td><strong>Zhu et al. (2013)</strong></td>
<td>91.1</td>
<td>91.5</td>
<td>91.3</td>
<td>–</td>
</tr>
<tr>
<td>Charniak and Johnson (2005)</td>
<td>91.2</td>
<td>91.8</td>
<td>91.5</td>
<td>84</td>
</tr>
</tbody>
</table>

*Grayed parsers* are ensemble/reranking/semi-supervised systems.
Experiments: Morphologically Rich Languages

- Results on SPMRL14 shared task datasets (Seddah et al., 2014).

<table>
<thead>
<tr>
<th>Parser</th>
<th>Bas</th>
<th>Fre</th>
<th>Ger</th>
<th>Heb</th>
<th>Hun</th>
<th>Kor</th>
<th>Pol</th>
<th>Swe</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berkeley</td>
<td>70.50</td>
<td>80.38</td>
<td>78.30</td>
<td>86.96</td>
<td>81.62</td>
<td>71.42</td>
<td>79.23</td>
<td>79.19</td>
<td>78.45</td>
</tr>
<tr>
<td>Berkeley Tagged</td>
<td>74.74</td>
<td>79.76</td>
<td>78.28</td>
<td>85.42</td>
<td>85.22</td>
<td>78.56</td>
<td>86.75</td>
<td>80.64</td>
<td>81.17</td>
</tr>
<tr>
<td>Crabbé and Seddah (2014)</td>
<td>85.35</td>
<td>79.68</td>
<td>77.15</td>
<td>86.19</td>
<td>87.51</td>
<td>79.35</td>
<td>91.60</td>
<td>82.72</td>
<td>83.69</td>
</tr>
<tr>
<td>Hall et al. (2014)</td>
<td>83.39</td>
<td>79.70</td>
<td>78.43</td>
<td>87.18</td>
<td>88.25</td>
<td>80.18</td>
<td>90.66</td>
<td>82.00</td>
<td>83.72</td>
</tr>
<tr>
<td>This work</td>
<td>85.90</td>
<td>78.75</td>
<td>78.66</td>
<td>88.97</td>
<td>88.16</td>
<td>79.28</td>
<td>91.20</td>
<td>82.80</td>
<td>84.22</td>
</tr>
<tr>
<td>Björkelund et al. (2014)</td>
<td>88.24</td>
<td>82.53</td>
<td>81.66</td>
<td>89.80</td>
<td>91.72</td>
<td>83.81</td>
<td>90.50</td>
<td>85.50</td>
<td>86.72</td>
</tr>
</tbody>
</table>
Experiments: Discontinuous Parsing

- Results on the discontinuous TIGER treebank (Brants et al., 2002).

<table>
<thead>
<tr>
<th>TIGER-SPMRL, $L \leq 70$</th>
<th>$F_1$</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>gold tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Versley (2014b)</td>
<td>76.46</td>
<td>41.05</td>
</tr>
<tr>
<td>This work</td>
<td>80.98</td>
<td>43.44</td>
</tr>
<tr>
<td>pred. tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Versley (2014b)</td>
<td>73.90</td>
<td>37.00</td>
</tr>
<tr>
<td>This work</td>
<td>77.72</td>
<td>38.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIGER-H&amp;N, $L \leq 40$</th>
<th>$F_1$</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>gold tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hall and Nivre (2008)</td>
<td>79.93</td>
<td>37.78</td>
</tr>
<tr>
<td>Versley (2014a)</td>
<td>74.23</td>
<td>37.32</td>
</tr>
<tr>
<td>This work</td>
<td>85.53</td>
<td>51.21</td>
</tr>
<tr>
<td>pred. tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hall and Nivre (2008)</td>
<td>75.33</td>
<td>32.63</td>
</tr>
<tr>
<td>van Cranenburgh and Bod (2013)</td>
<td>78.8–</td>
<td>40.8–</td>
</tr>
<tr>
<td>This work</td>
<td>82.57</td>
<td>45.93</td>
</tr>
</tbody>
</table>
Experiments: Discontinuous Parsing

- Results on the discontinuous NEGRA treebank (Skut et al., 1997).

<table>
<thead>
<tr>
<th></th>
<th>NEGRA, $L \leq 40$</th>
<th>$F_1$</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>gold tags</td>
<td>van Cranenburgh (2012)</td>
<td>72.33</td>
<td>33.16</td>
</tr>
<tr>
<td></td>
<td>van Cranenburgh and Bod (2013)</td>
<td>76.8–</td>
<td>40.5–</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>81.08</td>
<td>48.04</td>
</tr>
<tr>
<td>pred. tags</td>
<td>van Cranenburgh and Bod (2013)</td>
<td>74.8–</td>
<td>38.7–</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>77.93</td>
<td>44.83</td>
</tr>
</tbody>
</table>

- We parse all sentences (regardless of length) in 27.1 seconds in a single core (618 toks/sec)
- Orders of magnitude faster than van Cranenburgh and Bod (2013)
- Similar speed as the easy-first system of Versley (2014a), but much higher accuracy
Outline

1 Turbo Parsers

2 Parsing as Reduction
   - Dependencies and Constituents
   - Head-Ordered Dependency Trees
   - Reduction-Based Constituent Parsers
   - Experiments
   - Conclusions
Conclusions

- We proposed a **reduction technique** that allows to implement a constituent parser when only a dependency parser is available.
- The technique is very **simple** and **flexible**: applicable to any dependency parser, regardless of its nature or kind.
- If the dependency parser is non-projective, we can predict **discontinuous constituent trees**.
- We showed empirically that the reduction leads to highly-competitive constituent parsers for English and 8 morphologically rich languages.
- We surpassed the state of the art in discontinuous parsing of German by a wide margin.
We’re Hiring!

Excited about MT, crowdsourcing and Lisbon? ⇒ jobs@unbabel.com.
Acknowledgments

- Spanish Ministry of Economy and Competitiveness and FEDER (project TIN2010-18552-C03-01)
- Ministry of Education (FPU Grant Program) and Xunta de Galicia (projects R2014/029 and R2014/034)
- Fundação para a Ciência e Tecnologia, grants UID/EEA/50008/2013 and PTDC/EEI-SII/2312/2012.
- Priberam: QREN/POR Lisboa (Portugal), EU/FEDER programme, Intelligo project, contract 2012/24803.
References I


References II


