Dependency Parsing: Past, Present, and Future

Wenliang Chen, Zhenghua Li, Min Zhang
{wlchen, zhli13, minzhang}@suda.edu.cn
Soochow University, China
Recent Events of Dependency Parsing

- Shared tasks
  - CoNLL 2006/2007 shared tasks on multilingual dependency parsing
  - CoNLL 2008/2009 shared tasks on joint parsing of syntactic and semantic dependencies
  - SANCL 2012 shared task organized by Google (parsing the web)
  - SemEval 2014/2015: broad-coverage semantic dependency parsing (SDP)
Recent Events of Dependency Parsing

• Tutorials
  • COLING-ACL06: “Dependency Parsing” by Joakim Nivre and Sandra Kübler
  • NAACL10: “Recent Advances in Dependency Parsing” by Qin Iris Wang and Yue Zhang
  • IJCNLP13: Ours
  • EACL14: “Recent Advances in Dependency Parsing” by Ryan McDonald and Joakim Nivre
  • ACL14: “Syntactic Processing Using Global Discriminative Learning and Beam-Search Decoding” by Yue Zhang, Meishan Zhang, and Ting Liu

• Books
  • “Dependency Parsing” by Sandra Kübler, Joakim Nivre, and Ryan McDonald, 2009
Outline

- Part A: dependency parsing and supervised approaches
- Part B: semi-supervised dependency parsing
- Part C: Parsing the web and domain adaptation
- Part D: Multilingual dependency parsing
- Part E: Conclusion and open problems
Part A: Dependency Parsing and Supervised approaches
A dependency tree is a tree structure composed of the input words and meets a few constraints:
- Single-head
- Connected
- Acyclic
Projective dependency trees

- Informally, “projective” means the tree does not contain any crossing arcs.
Non-projective dependency trees

- A non-projective dependency tree contains crossing arcs.

Example from “Dependency Parsing” by Kübler, Nivre, and McDonald, 2009
Dependency Tree

- The basic unit is a link (dependency, arc) from the head to the modifier.

- **obj**
  - **eat**
    - **Head**
    - **Governor**
    - **Parent**
  - **fish**
    - **Modifier**
    - **Dependent**
    - **Child**

- **eat**
  - **obj**
  - **fish**

- **Label**
- **Relation**
- **Type**
Dependency Tree

- A bilingual example
Evaluation Metrics

- **Unlabeled attachment score (UAS)**
  - The percent of words that have the correct heads

- **Labeled attachment score (LAS)**
  - The percent of words that have the correct heads and labels.

- **Root Accuracy (RA)**

- **Complete Match rate (CM)**
Formalism of Dependency Parsing

\[ Y^* = \arg\max_{Y \in \Phi(X)} \text{score}(X, Y) \]

\[ X = x_1 x_2 \ldots x_n \rightarrow \text{the input sentence} \]

\[ (h, m) \rightarrow \text{a link from the head } x_h \text{ to the modifier } x_m \]

\[ Y = \{(h, m) : 0 \leq h \leq n, 0 < m \leq n\} \rightarrow \text{a candidate tree} \]

\[ \Phi(X) \rightarrow \text{The set of all possible dependency trees over } X \]
Supervised Approaches for Dependency Parsing

- Graph-based
- Transition-based
- Hybrid (ensemble)
- Other methods
Graph-based Dependency Parsing

- Find the highest scoring tree from a complete dependency graph.

\[ Y^* = \arg \max_{Y \in \Phi(X)} \text{score}(X, Y) \]
Two problems

- The search problem
  - Given the score of each link, how to find $Y^*$?
- The learning problem
  - Given an input sentence, how to determine the score of each link? $w \cdot f$
  - How to learn $w$ using a treebank (supervised learning)?

$$Y^* = \arg\max_{Y \in \Phi(X)} \text{score}(X, Y)$$
First-order as an example

- The first-order graph-based method assumes that the dependencies in a tree are independent from each other (arc-factorization)

\[
score(X, Y) = \sum_{(h,m) \in Y} score(X, h, m)
\]
Search problem for first-order model

- Eisner (2000) described a dynamic programming based decoding algorithm for bilexical grammar.
- McDonald+ (2005) applied this algorithm to the search problem of the first-order model.
- Time complexity $O(n^3)$
Eisner algorithm data structure

- Basic data structure
  - Incomplete spans
Eisner algorithm data structure

- Basic data structure
  - Complete spans
Eisner algorithm operations

- Basic Operations

1. \( s \quad r \quad r+1 \quad t \)

2. \( s \quad r \quad r \quad t \)

3. \( s \quad r \quad r \quad t \)
Eisner algorithm

- Initialization: complete spans (width=1)
Eisner algorithm

- Build incomplete span (width = 2)
Eisner algorithm

- Build complete spans (width = 2)
Eisner algorithm

- Build incomplete span (width = 3)

\[ \text{score}(\text{does} \rightarrow \text{here}) = 3.3 \]
Eisner algorithm

- Build incomplete spans (width = 3)

\[ \text{score} \left( \text{does} \rightarrow \text{here} \right) = 3.3 \]
Eisner algorithm

- Build complete spans (width = 3)
- Build incomplete spans (width = 4)
- ...
- The best parse is stored in the complete span from “$” to “here”: $C(0,4)$

$0 \quad He_1 \quad does_2 \quad it_3 \quad here_4$
Eisner algorithm

- The search path for the correct parse tree
The learning problem

- Given an input sentence, how to determine the score of each link?

  \[ \text{score}(2,4) = ? \]

- Feature based representation: a link is represented as a feature vector \( \mathbf{f}(2,4) \)

  \[ \text{score}(2,4) = \mathbf{w} \cdot \mathbf{f}(2,4) \]
Features for one dependency

Example from slides of Rush and Petrov (2012)
How to learn w? (supervised)

- Use a treebank
  - Each sentence has a manually annotated dependency tree.
- Online training (Collins, 2002; Crammer and Singer, 2001; Crammer+, 2003)
  - Initialize $w = 0$
  - Go though the treebank for a few (10) iterations.
    - Use one instance to update the weight vector.
Online learning $w$

Gold-standard parse $Y^+$

\[ w^{(k+1)} = w^{(k)} + f(X, Y^+) - f(X, Y^-) \]
Quick summarization

• The search problem
  • Dynamic programming based decoding algorithms (Eisner algorithm) to find the best parse tree.

• The learning problem
  • Online training algorithms to learn the feature weight vector $\mathbf{w}$ under the supervision of a treebank.
Recent advances (graph-based method)

- Explore features of more non-local subtrees
  - Second-order (McDonald & Pereira 06; Carreras 07)
  - Third-order (Koo+, 2010)
  - Higher-order with beam search (Zhang & McDonald 12)
Transition-based Dependency Parsing

• Gradually build a tree by applying a sequence of transition actions. (Yamada and Matsumoto, 2003; Nivre, 2003)

• The score of the tree is equal to the summation of the scores of the actions.

\[ score(X, Y) = \sum_{i=0}^{m} score(X, h_i, a_i) \]

- \(a_i\) → the action adopted in step \(i\)
- \(h_i\) → the partial results built so far by \(a_0...a_{i-1}\)
- \(Y\) → the tree built by the action sequence \(a_0...a_m\)
Transition-based Dependency Parsing

- The goal of a transition-based dependency parser is to find the highest scoring action sequence that builds a legal tree.

\[
Y^* = \text{arg max}_{Y \in \Phi(X)} \text{score}(X, Y)
\]

\[
= \text{arg max} \sum_{a_0 \ldots a_m \rightarrow Y} \text{score}(X, h_i, a_i)
\]
Two problems for Transition-based DP

- The search problem
  - Assuming the model parameters (feature weights) are known, how to find the optimal action sequence that leads to a legal tree for an input sentence?
  - Greedy search as an example

- The learning problem
  - How to learn the feature weights?
  - Global online training
    - Before 2008: Locally training classifiers
Components of Transition-based DP

- A stack to store the processed words and the partial trees
- A queue to store the unseen input words
- Transition actions
  - Gradually build a dependency tree according to the contexts (history) in the stack and the queue.

Which action should be applied?
An arc-eager transition-based parser

- Four actions
  - Shift
  - Left-arc
  - Right-arc
  - Reduce
An arc-eager transition-based parser

Stack

$0$

Input queue

He$_1$ does$_2$ it$_3$ here$_4$

↓ Shift

$0$ He$_1$

does$_2$ it$_3$ here$_4$
An arc-eager transition-based parser

\[ \begin{array}{c|c|c|c|c} 
$0$ & $H_1$ & \hline 
\hline 
\end{array} \quad \begin{array}{c} 
does_2 \quad it_3 \quad here_4 
\end{array} \]

$\downarrow$ Left-arc

\[ \begin{array}{l|c|c|c|c} 
$0$ & \hline 
\hline 
\end{array} \quad \begin{array}{c} 
does_2 \quad it_3 \quad here_4 
\end{array} \]

$H_1$
An arc-eager transition-based parser
An arc-eager transition-based parser
An arc-eager transition-based parser

He₁ does₂ it₃

Reduce

He₁ does₂ it₃

here₄

Reduced

here₄
An arc-eager transition-based parser

$0$ does$_2$

He$_1$  it$_3$

here$_4$

Right-arc

$0$ does$_2$ here$_4$

He$_1$  it$_3$
An arc-eager transition-based parser

$0 \quad \text{does}_2 \quad \text{here}_4$

$\text{He}_1 \quad \text{it}_3$

Reduce

$0 \quad \text{does}_2$

$\text{He}_1 \quad \text{it}_3 \quad \text{here}_4$
An arc-eager transition-based parser

Complete!
Recent advances (transition-based)

- Explore richer features using the partially built structures (Zhang and Nivre, 2011).
- Enlarge the search space during decoding
  - Beam search (Duan+, 2007; Zhang and Nivre, 2011)
  - Beam search with state merging via dynamic programming (Huang and Sagae, 2010)
  - Dynamic programming based decoding (Kuhlmann+, 2011)
- Better online learning
  - Dynamic oracles (Goldberg and Nivre, 2014)
Other methods

• Easy-first non-directional dependency parsing (Goldberg and Elhadad, 2010)
  • Iteratively select easiest (highest-scoring) pair of neighbors to build a dependency.

• Constituent-based dependency parsing (Sun and Wan, 2013)
  • Method 1: convert the outputs of a constituent parser into dependency structures.
  • Method 2: convert dependency trees into context-free-grammar structures.
Ensemble methods

- Different dependency parsers have different advantages.
  - The graph-based MSTParser performs better on long-distance dependencies.
  - The transition-based MaltParser performs better on short-distance dependencies.
- Ensemble methods try to leverage the complementary strengths of different parsing approaches.
  - Re-parsing (Sagae and Lavie, 2006; Surdeanu and Manning, 2010)
  - Stacking (McDonald and Nivre, 2011; Martins+, 2008)
Ensemble via re-parsing

- Sagae and Lavie (2006); Surdeanu and Manning (2010)
  - Separately train M different parsers
  - For a test sentence, the M parsers produce M parses.
  - Combine the M parses to build a partial dependency graph.
  - Reparse to find the best result from the dependency graph using the standard MST parsing algorithm.
Ensemble via re-parsing

- Sagae and Lavie (2006); Surdeanu and Manning (2010)

Example from the slides of Wang and Zhang (2010)
Ensemble via stacking

- Joakim and McDonald (2008); Martins+ (2008)
  - Combine the graph-based and transition-based parsers.
  - Use one parser to guide or help the other one.
    - Train the level-1 parser first (Parser1)
    - Let the level-2 parser (Parser2) consult Parser1 during both training and test.
  - Two directions
    - \(\text{MST}_{\text{malt}}\) (MaltParser for level-1; MSTParser as level-2)
    - \(\text{Malt}_{\text{MST}}\) (verse)
Non-projective dependency parsing

- Pseudo-projective (Nivre and Nilsson, 2005)
- Graph-based methods (McDonald+, 2005, 2006; Pilter, 2014)
- Transition-based methods (Nivre, 2009)

Example from “Dependency Parsing” by Kübler, Nivre, and McDonald, 2009
Non-projective dependency parsing

- Non-projectivity in natural languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Trees</th>
<th>Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>11.2%</td>
<td>0.4%</td>
</tr>
<tr>
<td>Basque</td>
<td>26.2%</td>
<td>2.9%</td>
</tr>
<tr>
<td>Czech</td>
<td>23.2%</td>
<td>1.9%</td>
</tr>
<tr>
<td>Danish</td>
<td>15.6%</td>
<td>1.0%</td>
</tr>
<tr>
<td>Greek</td>
<td>20.3%</td>
<td>1.1%</td>
</tr>
<tr>
<td>Russian</td>
<td>10.6%</td>
<td>0.9%</td>
</tr>
<tr>
<td>Slovene</td>
<td>22.2%</td>
<td>1.9%</td>
</tr>
<tr>
<td>Turkish</td>
<td>11.6%</td>
<td>1.5%</td>
</tr>
</tbody>
</table>

Table from invited talks by Prof. Joakim Niver: Beyond MaltParser - Advances in Transition-Based Dependency Parsing
Non-projective dependency parsing

- Pseudo-projective (Nivre and Nilsson, 2005)
  - Pre-processing
    - Convert non-projective trees into projective ones, with dependency labels encoding the transform process.

  ![Diagram](image)

- Projective dependency parsing for both training and test.
- Post-processing
  - Recover the non-projective trees from the projective outputs with the help of the labels.
Non-projective dependency parsing

- Graph-based methods
  - First-order (McDonald+, 2005)
    - Chu-Liu-Edmond decoding algorithm, $O(n^2)$
  - Second-order (McDonald and Pereira, 2006)
    - Greedy approximate decoding, $O(n^3)$
  - Third-order (Pitler, 2014)
    - Dynamic programming decoding, $O(n^4)$
Non-projective dependency parsing

- Transition-based methods
  - Extended with a SWAP action (Nivre, 2009)

Example from Wang and Zhang (2010)
Probabilistic dependency parsing

- Log-linear models (CRF)
  - Projective: inside-outside (Paskin, 2001)
  - Non-projective: matrix-tree theorem (Smith and Smith, 2007; McDonald and Satta, 2007; Koo+, 2007)

- Generative models
  - Spectral learning (Luque+, 2012; Dhillon+, 2012)
Improving parsing efficiency

- Coarse-to-fine parsing
- Fast feature generation
- Parallel techniques
Coarse-to-fine parsing

- Use a coarse and fast model to prune the search space for more complex models.
  - Charniak and Johnson (2005) apply this method to fast n-best constituent parsing.
  - Use a CRF-based first-order dependency parser to prune the search space before third-order parsing (Koo and Collins, 2010)
  - Vine pruning (Rush and Petrov, 2012)
    - Zero-order => first-order => second-order => third-order
Fast feature generation

- When only considering positive features, current graph-based dependency parsers usually incorporate tens of millions of different features.
- Feature generation based on standard hash tables needs ~90% of the total parsing time (Bohnet, 2010)
  - Feature string generation
  - Feature index mapping
Feature strings for one dependency

Example from slides of Rush and Petrov (2012)
Feature-index mapping

- Map different feature strings into different indices (feature space).

\[
\begin{align*}
\text{[went, VERB, As, IN]} & \rightarrow 1023 \\
\text{[VERB, As, IN, left]} & \rightarrow 526
\end{align*}
\]

- Feature weight
  - Each feature in the feature space has a weight.
Fast feature generation

  - Map each feature string into an index using a hash function with pure numerical calculation.
  - Hash collision is OK.
- Qian + (2010) propose a 2D trie structure for fast feature generation.
  - Complex feature templates are extension of simple ones.
Parallel techniques

• Parsing efficiency can be largely improved by exploring multiple CPU cores via multi-thread programming.
  • Graph-based parser (Bohnet, 2010)
    • Parallel feature generation
    • Parallel decoding algorithms
  • Transition-based parser (Hatori+, 2011)
    • Parallel beam-search decoding
• Both parsers are publicly available.
Quick summarization

Supervised dependency parsing

- Graph-based
- Transition-based
- Easy-first constituent-based
- Non-projective
- Probabilistic models
- Parsing efficiency

Parser ensemble: reparsing, stacking
Joint morphological analysis and dependency parsing

• Motivation
  • Due to the intrinsic difficulty of NLP, a cascaded framework is commonly adopted.
    • Morphology (lexicon) → Syntax → Semantics
  • Two problems
    • Error propagation
    • Fail to explore the connections between different levels of processing tasks which may help those related tasks.
Pipeline example: Chinese POS tagging and dependency parsing

Dependency Parsing

POS tagging

Words

# NN NT NN P NN PU

$0$ 欧文_1 现在_2 效力_3 于_4 利物浦队_5 。

Owen now plays for in Liverpool.
Joint Chinese POS tagging and dependency parsing

**Words**

Owen now plays for Liverpool.

**POS tag lattice**

- $0$ (None)
- $1$ (NR)
- $2$ (JJ)
- $3$ (VV)
- $4$ (JJ)
- $5$ (NR)

**Joint Parsing and Tagging**

- #
- NT
- P
- PU
Graph-based joint POS tagging and dependency parsing (Li+, 2011)

- Formally, the pipeline method
  - Step 1: POS tagging
    \[ T^* = \arg \max_{T \in \Phi_1(X)} \text{score}_{pos}(X, T) \]
  - Step 2: dependency parsing
    \[ Y^* = \arg \max_{Y \in \Phi_2(X)} \text{score}_{syn}(X, T^*, Y) \]

```
$0\ 1\ \text{Owen} \ 2\ \text{now} \ 3\ \text{palys}\ 4\ \text{for}\ 5\ \text{Liverpool} 6.
```

```
# \quad \text{NN} \quad \text{NN} \quad \text{NN} \quad \text{NN} \quad \text{NN} \quad \text{PU}
\$0\ \quad \text{欧文} \quad \text{现在} \quad \text{效力} \quad \text{于} \quad \text{利物浦} \quad \text{。}
Owen \quad \text{now} \quad \text{palys for} \quad \text{in} \quad \text{Liverpool} \quad \text{.}
```
The joint method tries to solve the two tasks simultaneously.

\[
\langle T^*, Y^* \rangle = \arg \max_{T \in \Phi_1(X), Y \in \Phi_2(X)} \text{score}_{\text{joint}}(X, T, Y)
\]

Graph-based joint POS tagging and dependency parsing (Li+ 2011)

Owen now plays for Liverpool.
Graph-based joint POS tagging and dependency parsing (Li+, 2011)

\[ score_{\text{joint}}(X, T, Y) \]

\[ = score_{\text{pos}}(X, T) + score_{\text{syn}}(X, T, Y) \]

\[ = w_{\text{pos}} \cdot f_{\text{pos}}(X, T) + w_{\text{syn}} \cdot f_{\text{syn}}(X, T, Y) \]

\[ = w_{\text{pos} \oplus \text{syn}} \cdot f_{\text{pos} \oplus \text{syn}}(X, T, Y) \]

\[ = w_{\text{joint}} \cdot f_{\text{joint}}(X, T, Y) \]

Under the joint model, the POS tagging features and the syntactic features can interact with each other in order to find an optimal joint solution.
Graph-based joint POS tagging and dependency parsing (Li+, 2011)

- The search problem
  - Given the feature weights $w_{joint}$, how to efficiently find the optimal joint result from a huge search space?
  - Dynamic programming based decoding algorithms: direct extension of the decoding algorithms for dependency parsing
Dynamic programming based decoding algorithms (Li+, 2011)

- Product of two dynamic programming based decoding algorithms
- Augment partial parses (spans) with POS tags.
- Time complexity $O(n^3q^4)$ ($q=1.4$)
The learning problem (Li+, 2011)

- How to learn the feature weights $w_{joint}$?
- Online training
  - Averaged perceptron (AP)
  - Margin infused relaxed algorithm (MIRA)
  - Passive-aggressive algorithm (PA)
Separately passive-aggressive (SPA) learning (Li+, 2012)

- Use separate update steps for the POS tagging features and syntactic features.
- Can better balance the discriminative power of both tagging and parsing features.
- Lead to better tagging and parsing accuracy.
Results of graph-based joint models

On Chinese Data: CTB5
Transition-based joint Chinese POS tagging and dependency parsing

- A direct extension of transition-based dependency parsing by adding a tagging action (Hatori+, 2011; Bohnet and Nivre, 2012)
  - An arc-standard version (Hatori+, 2011)
    - Shift($t$): shift a word in the queue into the stack and assign tag $t$ to it. (SH)
    - Reduce-left (RL)
    - Reduce-right (RR)
Transition-based joint Chinese POS tagging and dependency parsing

• An example from the slides of Hatori+ (2011)

<table>
<thead>
<tr>
<th>#</th>
<th>Act.</th>
<th>Stack S</th>
<th>Queue Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>$\phi$</td>
<td>我/?? 想/?? 把/?? …</td>
</tr>
<tr>
<td>1</td>
<td>SH(PN)</td>
<td>我/PN</td>
<td>想/?? 把/?? 这/?? …</td>
</tr>
<tr>
<td>2</td>
<td>SH(VV)</td>
<td>我/PN 想/VV</td>
<td>把/?? 这/?? 个/?? …</td>
</tr>
<tr>
<td>3</td>
<td>SH(BA)</td>
<td>我/PN 想/VV 把/BA</td>
<td>这/?? 个/?? 句子/?? …</td>
</tr>
<tr>
<td>4</td>
<td>SH(DT)</td>
<td>我/PN 想/VV 把/BA 这/DT</td>
<td>个/?? 句子/?? 翻译/?? …</td>
</tr>
<tr>
<td>5</td>
<td>SH(M)</td>
<td>我/PN 想/VV 把/BA 这/DT 个/M</td>
<td>句子/?? 翻译/?? 成/?? …</td>
</tr>
<tr>
<td>6</td>
<td>RL</td>
<td>我/PN 想/VV 把/BA 这/DT ~[个/M]</td>
<td>句子/?? 翻译/?? 成/?? …</td>
</tr>
<tr>
<td>7</td>
<td>SH(NN)</td>
<td>我/PN 想/VV 把/BA 这/DT ~[个/M] 句子/NN</td>
<td>翻译/?? 成/?? 英语/?? $</td>
</tr>
<tr>
<td>8</td>
<td>RR</td>
<td>我/PN 想/VV 把/BA [这/DT~[…] ~句子/NN</td>
<td>翻译/?? 成/?? 英语/?? $</td>
</tr>
<tr>
<td>9</td>
<td>RL</td>
<td>我/PN 想/VV 把/BA ~[…] ~句子/NN</td>
<td>翻译/?? 成/?? 英语/?? $</td>
</tr>
<tr>
<td>10</td>
<td>SH(VV)</td>
<td>我/PN 想/VV 把/BA ~[…] ~句子/NN 翻译/VV</td>
<td>成/?? 英语/?? $</td>
</tr>
</tbody>
</table>
Other work on joint morphological analysis and parsing

- Easy-first joint Chinese POS tagging and dependency parsing (Ma+, 2012)
- Transition-based joint Chinese word segmentation, POS tagging, and dependency parsing (Hatori+, 2012; Li and Zhou, 2012)
- Joint Chinese word segmentation, POS tagging, and phrase-structure parsing
  - Zhang+ (2013): a character-level transition-based system
Other work on joint morphological analysis and parsing

- Dual decomposition (DD) (Rush+, 2010)
  - Integrating different NLP subtasks at the test phase
    - a phrase-structure parser and a dependency parser
    - a phrase-structure parser and a POS tagger

- Loopy belief propagation (LBP) (Lee+, 2011)
  - Joint morphological disambiguation and dependency parsing for morphologically-rich languages including Latin, Czech, Ancient Greek, and Hungarian.

- Comparison of DD and LBP (Auli and Lopez, 2011)
  - Joint CCG supertagging and parsing
References


## References


References

References

- Yoav Goldberg and Joakim Nivre. 2014. Training Deterministic Parsers with Non-Deterministic Oracles. In TACL.
References

- Emily Pitler. 2014. A Crossing-Sensitive Third-Order Factorization for Dependency Parsing. In TACL
- Xian Qian, Qi Zhang, Xuangjing Huang, Lide Wu. 2010. 2D Trie for Fast Parsing. In Proc. of COLING, pp904-912
References

End of Part A