Joint Parsing and Disfluency Detection in Linear Time

Mohammad Sadegh Rasooli  Joel Tetreault

Columbia University

Nuance Communications
Overview

1. Introduction
   - Speech Disfluency
   - Parsing Disfluent Sentences

2. Joint Dependency Parsing and Disfluency Detection
   - Arc-Eager Parsing
   - Additional Transitions for Handling Disfluencies
   - Learning Model

3. Evaluation
   - Disfluency Detection
   - Parser Evaluation
Speech Disfluency

- Speech text is mostly disfluent
- Disfluency types:
  - Filled pauses; e.g. *uh, um*
  - Discourse markers and parentheticals; e.g. *I mean, you know*
  - Reparandum (edited phrase)

I want a flight to Boston *uh I mean* to Denver

- Reparandum
- FP
- Prn
- Repair
Most prior approaches focus solely on disfluency detection.

Why not parse the disfluent sentence at the same time as disfluency detection?

✓ This has the potential to speed-up spoken language processing in dialogue systems.
Parsing Disfluent Sentences

- Parsing spoken language is harder than written text. Disfluencies make it much harder
- How about joint parsing?
  Studies that only focus on disfluency detection vastly outperform joint model approaches by 20 F score or more.
Our Approach: Joint Parsing and Disfluency Detection

- Parsing and disfluency detection with high accuracy and processing speed.
  
  I want a flight to Boston uh I mean to Denver

  I want a flight to Denver

This is the real output of our system!
○ Parsing and disfluency detection with high accuracy and processing speed.

I want a flight to Boston uh I mean to Denver
I want a flight to Denver

This is the real output of our system!
1 Introduction

Speech Disfluency
Parsing Disfluent Sentences

2 Joint Dependency Parsing and Disfluency Detection

Arc-Eager Parsing
Additional Transitions for Handling Disfluencies
Learning Model

3 Evaluation

Disfluency Detection
Parser Evaluation
Arc-Eager Parsing [Nivre, 2004]

- **Goal**: Finding the best dependency tree
- **Parser State**: Buffer of words, stack of already processed words and set of already made dependency arcs.
- **Initialization**: Buffer with sentence words, stack and arc-set are empty.
- **Final State**: Stack and buffer are empty and arc-set has a set of arcs.
Arc-Eager Parsing

Actions in an arc-eager algorithms are:

- **Shift**: \([... \ j]_S [i \ k ...]_B \rightarrow [... \ j \ i]_S [k ...]_B\)
- **Right-arc**: \([... \ j]_S [i \ k ...]_B \rightarrow [... \ j \ i]_S [k ...]_B + \text{add-arc}(j,i)\)
- **Left-arc**: \([... \ h \ j]_S [i \ k ...]_B \rightarrow [...h]_S [i \ k ...]_B + \text{add-arc}(i,j)\)
- **Reduce**: \([... \ h \ j]_S [i \ k ...]_B \rightarrow [...h]_S [i \ k ...]_B\)

- Are these actions **ENOUGH** for disfluency detection?
Arc-Eager Parsing

Actions in an arc-eager algorithms are:

- **Shift:** \([... \ j]_S [i \ k \ ...]_B \rightarrow [... \ j \ i]_S [k \ ...]_B\)
- **Right-arc:** \([... \ j]_S [i \ k \ ...]_B \rightarrow [... \ j \ i]_S [k \ ...]_B + \text{add-arc}(j,i)\)
- **Left-arc:** \([... \ h \ j]_S [i \ k \ ...]_B \rightarrow [...h]_S [i \ k \ ...]_B + \text{add-arc}(i,j)\)
- **Reduce:** \([... \ h \ j]_S [i \ k \ ...]_B \rightarrow [...h]_S [i \ k \ ...]_B\)
- **Are these actions ENOUGH for disfluency detection?**
Additional Transitions for Handling Disfluencies

- Three additional actions:

  \textbf{Intj}[i]: Remove the first \( i \) words from the buffer and tag them as \textit{interjection} (\textbf{Intj}).

\begin{align*}
\text{[ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S & \quad \rightarrow \quad \text{Next action is Intj}[1] \\
\text{[ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S & \quad \rightarrow \quad \text{[uh}_7, \text{i}_8, \text{mean}_9, \text{to}_10, \text{Denver}_{11}]_B \\
\text{[I}_8, \text{mean}_9, \text{to}_10, \text{Denver}_{11}]_B
\end{align*}
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \texttt{Intj[i]}: Remove the first $i$ words from the buffer and tag them as \textit{interjection} (\texttt{Intj}).

  \[
  \begin{array}{l}
  \text{Next action is } \texttt{Intj[1]} \\
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \rightarrow [\text{uh}_7, \text{i}_8, \text{mean}_9, \text{to}_{10}, \text{Denver}_{11}]_B \\
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \rightarrow [\text{i}_8, \text{mean}_9, \text{to}_{10}, \text{Denver}_{11}]_B
  \end{array}
  \]
Three additional actions:

- **Intj[i]**: Remove the first $i$ words from the buffer and tag them as *interjection* (Intj).

  \[
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad \text{[uh}_7, \text{l}_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B
  \]

  → **Next action is Intj[1]**

  \[
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{l}_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B
  \]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - $\text{Intj}[i]$: Remove the first $i$ words from the buffer and tag them as *interjection* ($\text{Intj}$).

  \[
  \begin{align*}
  &\text{[ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{uh}_7, \text{l}_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B \\
  \rightarrow & \quad \text{Next action is Intj}[1] \\
  &\text{[ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{l}_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B
  \end{align*}
  \]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \textbf{Prn}[i]: Remove the first $i$ words from the buffer and tag them as \textit{discourse marker} (Prn).

  \[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6 \] \_S
  \[\text{I}_8, \text{mean}_9, \text{to}_10, \text{Denver}_{11} \] \_B

  → Next action is Prn[2]

  \[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6 \] \_S
  \[\text{to}_10, \text{Denver}_{11} \] \_B
Three additional actions:

- **Prn[i]**: Remove the first $i$ words from the buffer and tag them as *discourse marker* (**Prn**).

  \[
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \rightarrow \text{Next action is Prn}[2]
  \]

  \[
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \rightarrow [\text{to}_10, \text{Denver}_11]_B
  \]
Three additional actions:

\( \text{Prn}[i] \): Remove the first \( i \) words from the buffer and tag them as *discourse marker* (Prn).

\[
[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [l_8, \text{mean}_9, \text{to}_10, \text{Denver}_{11}]_B
\]

Next action is \( \text{Prn}[2] \)

\[
[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{to}_10, \text{Denver}_{11}]_B
\]
Introduction
Joint Dependency Parsing and Disfluency Detection
Evaluation

Arc-Eager Parsing
Additional Transitions for Handling Disfluencies
Learning Model

Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \text{Prn}[i]: Remove the first $i$ words from the buffer and tag them as \textit{discourse marker (Prn)}.

\[
\begin{align*}
\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6 \rangle_S \rightarrow & \text{Next action is Prn}[2] \\
\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6 \rangle_S & [l_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B \\
\end{align*}
\]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \textbf{RP[i:j]}: From the words outside the buffer, remove un-removed words \textit{i} to \textit{j} and tag them as \textit{reparandum} (RP).

\[
\text{Candidates: RP[6:6], RP[5:6], RP[4:6], RP[3:6], ..., Intj[1], Intj[2], ..., Prn[1], Prn[2], ..., Shift, Reduce, Left-arc, Right-arc}
\]

\[
\rightarrow \text{Next action is RP[5:6]}
\]

\[
[\text{ROOT}_0, \text{want}_2, \text{flight}_4]_S \quad [\text{to}_{10}, \text{Denver}_{11}]_B
\]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \( \text{RP}[i:j] \): From the words outside the buffer, remove un-removed words \( i \) to \( j \) and tag them as \textit{reparandum} (RP).

\[
[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{to}_{10}, \text{Denver}_{11}]_B
\]

Candidates: \( \text{RP}[6:6], \text{RP}[5:6], \text{RP}[4:6], \text{RP}[3,6], ..., \text{Intj}[1], \text{Intj}[2], ..., \text{Prn}[1], \text{Prn}[2], ..., \text{Shift}, \text{Reduce}, \text{Left-arc}, \text{Right-arc} \)

\( \rightarrow \) Next action is \( \text{RP}[5:6] \)

\[
[\text{ROOT}_0, \text{want}_2, \text{flight}_4]_S \quad [\text{to}_{10}, \text{Denver}_{11}]_B
\]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \( \text{RP}[i:j] \): From the words outside the buffer, remove un-removed words \( i \) to \( j \) and tag them as \textit{reparandum} (RP).

\[
\text{[ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad \text{[to}_{10}, \text{Denver}_{11}]_B
\]

Candidates: \( \text{RP}[6:6], \text{RP}[5:6], \text{RP}[4:6], \text{RP}[3,6], ..., \text{Intj}[1], \text{Intj}[2], ..., \text{Prn}[1], \text{Prn}[2], ..., \text{Shift}, \text{Reduce}, \text{Left-arc}, \text{Right-arc} \)

→ Next action is \( \text{RP}[5:6] \)

\[
\text{[ROOT}_0, \text{want}_2, \text{flight}_4]_S \quad \text{[to}_{10}, \text{Denver}_{11}]_B
\]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - **RP[i:j]**: From the words outside the buffer, remove un-removed words $i$ to $j$ and tag them as *reparandum* (RP).

  \[
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{to}_{10}, \text{Denver}_{11}]_B
  \]

  Candidates: RP[6:6], RP[5:6], RP[4:6], RP[3,6], ..., Intj[1], Intj[2], ..., Prn[1], Prn[2], ..., Shift, Reduce, Left-arc, Right-arc

  → Next action is RP[5:6]

  \[
  [\text{ROOT}_0, \text{want}_2, \text{flight}_4]_S \quad [\text{to}_{10}, \text{Denver}_{11}]_B
  \]
Additional Transitions for Handling Disfluencies

- Three additional actions:
  - \( \text{RP}[i:j] \): From the words outside the buffer, remove un-removed words \( i \) to \( j \) and tag them as *reparandum* (RP).

\[
\begin{align*}
\text{Candidates: } & \text{RP}[6:6], \text{RP}[5:6], \text{RP}[4:6], \text{RP}[3:6], \ldots, \text{Intj}[1], \text{Intj}[2], \ldots, \text{Prn}[1], \text{Prn}[2], \ldots, \text{Shift, Reduce, Left-arc, Right-arc} \\
\rightarrow & \text{Next action is } \text{RP}[5:6] \\
\end{align*}
\]

\[
\begin{align*}
\text{Candidates: } & \text{RP}[6:6], \text{RP}[5:6], \text{RP}[4:6], \text{RP}[3:6], \ldots, \text{Intj}[1], \text{Intj}[2], \ldots, \text{Prn}[1], \text{Prn}[2], \ldots, \text{Shift, Reduce, Left-arc, Right-arc} \\
\rightarrow & \text{Next action is } \text{RP}[5:6] \\
\end{align*}
\]

\[
\begin{align*}
\text{Candidates: } & \text{RP}[6:6], \text{RP}[5:6], \text{RP}[4:6], \text{RP}[3:6], \ldots, \text{Intj}[1], \text{Intj}[2], \ldots, \text{Prn}[1], \text{Prn}[2], \ldots, \text{Shift, Reduce, Left-arc, Right-arc} \\
\rightarrow & \text{Next action is } \text{RP}[5:6] \\
\end{align*}
\]
Let’s Practice

\[ \text{ROOT}_0, \text{want}_2, \text{flight}_4 \]_S \quad [\text{to}_5, \text{Boston}_6, \text{uh}_7, \text{I}_8, \text{mean}_9, \text{to}_{10}, \text{Denver}_{11}]_B

Next action is \textit{right-arc:prep}

[\textit{Root}] \quad \text{I want a flight to Boston uh I mean to Denver}
Let’s Practice

\[ [\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5]_S \quad [\text{Boston}_6, \text{uh}_7, \text{I}_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B \]

Next action is **right-arc:pobj**

\[ [\text{Root}] \text{I want a flight to Boston uh I mean to Denver} \]
Let’s Practice

\[
\begin{align*}
&\text{[ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \\
&\text{[uh}_7, \text{I}_8, \text{mean}_9, \text{to}_10, \text{Denver}_11]_B
\end{align*}
\]

Next action is \textbf{Intj}[1]

\[
\text{[Root]} \quad \text{I want a flight to Boston uh I mean to Denver}
\]
Let’s Practice

[ROOT\_0, want\_2, flight\_4, to\_5, Boston\_6]_S  [l\_8, mean\_9, to\_10, Denver\_11]_B

Next action is Prn[2]

[Root] I want a flight to Boston I mean to Denver
Let’s Practice

\[
[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad [\text{to}_{10}, \text{Denver}_{11}]_B
\]

Next action is \textbf{RP}[5:6]

\[
[\text{Root}] \quad \text{I want a flight to Boston to Denver}
\]
Let’s Practice

\[[\text{ROOT}_0, \text{want}_2, \text{flight}_4, \text{to}_5, \text{Boston}_6]_S \quad \text{[to}_{10}, \text{Denver}_{11}]_B\]

Deleting words and dependencies

\[\text{[Root]} \quad \text{I want a flight to Boston to Denver}\]
Let’s Practice

\[ \text{[ROOT}_0, \text{want}_2, \text{flight}_4]_S \quad \text{[to}_10, \text{Denver}_11]_B \]

Next action is \textbf{right-arc:prep}

\[ \text{[Root]} \quad \text{I want a flight to Denver} \]
Let’s Practice

Let’s practice parsing the sentence: "I want a flight to Denver"

[ROOT$_0$, want$_2$, flight$_4$, to$_{10}$]$_S$  [Denver$_{11}$]$_B$

Next action is **right-arc:pobj**

[I want a flight to Denver]
Two Classifiers for Learning the Model

- Instead of having one complete joint model, we have two classifiers that each classifier has its own features and label set.

```
State
   \downarrow
  C1
   \downarrow
DM[i] Parse RP[i:j] IJ[i]
```

```
   \downarrow
  C2
   \downarrow
SH LA RA R
```

5.2 Learning
Averaged Structured Perceptron (Collins, 2002) is a discriminative supervised learning method which empirically converges very quickly. We use this algorithm for learning the weights of our features for all of our classifiers.

One of the main issues with detecting reparandum is sparsity: speech repairs happen X% of the time in the Switchboard corpus. Qian and Liu (2013) try to change the update weight for reparandum misclassification in training and see that this would improve the results. We also realize that changing of the weight for reparandum candidate for the cases where a "reparandum" is wrongly recognized as another label. This method is similar to weighted Perceptron (Cavallanti et al., 2007) and we call the modified version "weighted averaged Perceptron (WAP)". We can see that this minor modification to the algorithm improves the results significantly.

6 Experiments and Evaluation
6.1 Data Preparation
Most previous work on disfluency detection has used the Switchboard corpus (Godfrey et al., 1992) with the train/dev/test splits from (Johnson and Charniak, 2004). Unfortunately, comparability in this task is hampered by the fact that there are two data formats in Switchboard: `dps` and `mrg`. There are twice as many `dps` files as `mrg` files; i.e., `mrg` files are a subset of `dps` files. `Dps` files have part of speech tag information in addition to disfluencies tags, while `mrg` files have part of speech tags and bracketed trees. Furthermore, there is no complete one-to-one mapping between `dps` and `mrg` files, since the sentence boundaries differ slightly, and `dps` annotation of parenthetical disfluencies is more fine-grained. There are
Features

- We use two kinds of features for the first classifier: *local* and *global*.

| Global Features |
|-----------------|
| First n words inside/outside buffer (n=1:4) |
| First n POS i/o buffer (n=1:6) |
| Are n words i/o buffer equal? (n=1:4) |
| Are n POS i/o buffer equal? (n=1:4) |
| n last FG transitions (n=1:5) |
| n last transitions (n=1:5) |
| n last FG transitions + first POS in the buffer (n=1:5) |
| n last transitions + first POS in the buffer (n=1:5) |
| (n+m)-gram of m/n POS i/o buffer (n,m=1:4) |
| Refined (n+m)-gram of m/n POS i/o buffer (n,m=1:4) |
| Are n first words of i/o buffer equal? (n=1:4) |
| Are n first POS of i/o buffer equal? (n=1:4) |
| Number of common words i/o buffer words (n=1:6) |

| Local Features |
|----------------|
| First n words of the candidate phrase (n=1:4) |
| First n POS of the candidate phrase (n=1:6) |
| Distance between the candidate and first word in the buffer |
We experimented with two learning algorithms [Collins, 2002]:

- We use averaged Perceptron [Collins, 2002] with mostly binary features (AP).
- Changing weight updates from 1 to 2 for misclassification of reparandum shows improvement (WAP).
1 Introduction
   Speech Disfluency
   Parsing Disfluent Sentences

2 Joint Dependency Parsing and Disfluency Detection
   Arc-Eager Parsing
   Additional Transitions for Handling Disfluencies
   Learning Model

3 Evaluation
   Disfluency Detection
   Parser Evaluation
Evaluation Data and Measures

- **Data**
  - We use Switchboard parsed section (mrg files) with the same train/dev/test split as [Johnson and Charniak, 2004]

- **Metric**
  - **Disfluency detection**: F-score of detecting reparandum.
  - **Parsing**: F-score of finding correct parents for fluent words.
## Disfluency Detection

| Model                                      | Model Description          | F-Score |
|--------------------------------------------|----------------------------|---------|
| [Miller and Schuler, 2008]                 | Joint + PCFG parsing       | 30.6    |
| [Lease and Johnson, 2006]                  | Joint + PCFG parsing       | 62.4    |
| [Kahn et al., 2005]                        | TAG + LM rerank.           | 78.2    |
| [Qian and Liu, 2013]                       | IOB tagging                | 81.4    |
| [Qian and Liu, 2013]–Opt.                  | IOB tagging                | **82.1**|
| Our Model                                  | AP                         | 80.9    |
| Our Model                                  | WAP                        | **81.4**|
Parser Evaluation

Table 1: Parsing results. UB = upperbound (parsing clean sentences), LB = lowerbound (parsing disfluent sentences without disfluency correction). UAS is unlabeled attachment score (accuracy), Pr. is precision, Rec. is recall and F1 is f-score.

|      | UAS | LB  | UB  | Pr. | Rec. | F1  |
|------|-----|-----|-----|-----|-----|-----|
| AP   | 88.6| 70.7| 90.2| 86.8| 88.0| 87.4|
| WAP  | 88.1| 70.7| 90.2| 87.2| 88.0| 87.6|

Table 2: Disfluency results. Pr. is precision, Rec. is recall and F1 is f-score. KL = (Kahn et al., 2005), LJ = (Lease and Johnson, 2006), MS = (Miller and Schuler, 2008) and QL = (Qian and Liu, 2013).

As we see in Table 2, WAP works better than the original method. As mentioned before, the numbers are not completely comparable to others except for (Kahn et al., 2005; Lease and Johnson, 2006; Miller and Schuler, 2008) which we outperform. For the sake of comparing to the state of the art, the best result for this task (Qian and Liu, 2013) is replicated from their available software on the portion of dps files. As we see, WAP works better than the original method. As mentioned before, the numbers are not completely comparable to others except for (Kahn et al., 2005; Lease and Johnson, 2006; Miller and Schuler, 2008) which we outperform. For a fairer comparison, we also optimized the number of training iterations of (Qian and Liu, 2013) for the mrg set based on dev data (10 iterations instead of 30 iterations). As shown in the results, our model accuracy is slightly less than the state-of-the-art (which focuses solely on the disfluency detection task and does no parsing), but we believe that the performance can be improved through better features and by changing the model. Another characteristic of our model is that it operates at a very high precision, though at the expense of some recall.

Acknowledgements
We would like to thank anonymous reviewers for their helpful comments on the paper. Additionally, we were aided by researchers by their prompt responses to our many questions: Mark Core, Luciana Ferrer, Kallirroi Georgila, Mark Johnson, Jeremy Kahn, Yang Liu, Xian Qian, Kenji Sagae, and Wen Wang. Finally, this work was conducted during the first author's summer internship at the Nuance Sunnyvale Research Lab. We would like to thank the researchers in the group for the helpful discussions and assistance on different aspects of the problem. In particular, we would like to thank Chris Brew, Ron Kaplan, Deepak Ramachandran and Adwait Ratnaparkhi.
Our experiments show that our model is close to the state-of-the-art.

There are still many avenues of improving accuracy:

- Better structure: completely joint model
- Better features: prosodic features
- K-beam training and decoding
- Classifier ensemble
Any Question?

Thanks [for]$_{Rp}$. [uh]$_{Intj}$ for your attention
Bohnet, B. (2010).
Very high accuracy and fast dependency parsing is not a contradiction.
In *COLING*, pages 89–97.

Collins, M. (2002).
Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms.
In *ACL*, pages 1–8.

Godfrey, J. J., Holliman, E. C., and McDaniel, J. (1992).
Switchboard: Telephone speech corpus for research and development.
In *ICASSP*, volume 1, pages 517–520.

Johnson, M. and Charniak, E. (2004).
A TAG-based noisy channel model of speech repairs.
In *ACL*, pages 33–39.

Kahn, J. G., Lease, M., Charniak, E., Johnson, M., and Ostendorf, M. (2005).
Effective use of prosody in parsing conversational speech.
In *EMNLP*, pages 233–240.
References II

- Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency parsing.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

- Lease, M. and Johnson, M. (2006). Early deletion of fillers in processing conversational speech. In *NAACL-HLT*, pages 73–76.

- Levy, R. and Andrew, G. (2006). Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *LREC*, pages 2231–2234.

- Miller, T. and Schuler, W. (2008). A unified syntactic model for parsing fluent and disfluent speech. In *ACL-HLT*, pages 105–108.

- Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57.
Qian, X. and Liu, Y. (2013). Disfluency detection using multi-step stacked learning. In *NAACL-HLT*, pages 820–825.

Zwarts, S. and Johnson, M. (2011). The impact of language models and loss functions on repair disfluency detection. In *ACL*, pages 703–711.