Joint Parsing and Disfluency Detection in Linear Time

Mohammad Sadegh Rasooli*
Department of Computer Science
Columbia University, New York, NY
rasooli@cs.columbia.edu

Joel Tetreault
Nuance Communications, Inc.
Sunnyvale, CA
joel.tetreault@nuance.com

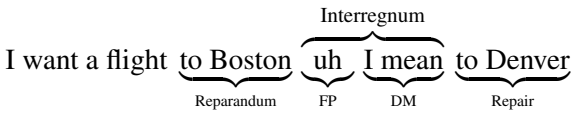
Abstract

We introduce a novel method to jointly parse and detect disfluencies in spoken utterances. Our model can use arbitrary features for parsing sentences and adapt itself with out-of-domain data. We show that our method, based on transition-based parsing, performs at a high level of accuracy for both the parsing and disfluency detection tasks. Additionally, our method is the fastest for the joint task, running in linear time.

1 Introduction

Detecting disfluencies in spontaneous speech has been widely studied by researchers in different communities including natural language processing (e.g. Qian and Liu (2013)), speech processing (e.g. Wang et al. (2013)) and psycholinguistics (e.g. Finlayson and Corley (2012)). While the percentage of spoken words which are disfluent is typically not more than ten percent (Bortfeld et al., 2001), this additional "noise" makes it much harder for spoken language systems to predict the correct structure of the sentence.

Disfluencies can be filled pauses (e.g. "uh", "um", "huh"), discourse markers (e.g. "you know", "I mean") or edited words which are repeated or corrected by the speaker. For example, in the following sentence, an edited phrase or *reparandum interval* ("to Boston") occurs with its repair ("to Denver"), a filled pause ("uh") and discourse marker ("I mean").[1]



Filled pauses and discourse markers are to some extent a fixed and closed set. The main challenge in finding disfluencies is the case where the edited phrase is neither a rough copy of its repair or has any repair phrase (i.e. discarded edited phrase). Hence, in previous work, researchers report their method performance on detecting edited phrases (reparandum) (Johnson and Charniak, 2004).

In contrast to most previous work which focuses solely on either detection or on parsing, we introduce a novel framework for jointly parsing sentences with disfluencies. To our knowledge, our work is the first model that is based on *joint dependency and disfluency detection*. We show that our model is robust enough to detect disfluencies with high accuracy, while still maintaining a high level of dependency parsing accuracy that approaches the upper bound. Additionally, our model outperforms prior work on joint parsing and disfluency detection on the disfluency detection task, and improves upon this prior work by running in linear time complexity.

The remainder of this paper is as follows. In §2, we overview some the previous work on disfluency detection. §3 describes our model. Experiments are described in §4 and Conclusions are made in §5.

* The first author worked on this project while he was a research intern in CoreNL research group, NLU lab, Nuance Communications, Sunnyvale, CA.

[1] In the literature, edited words are also known as "reparandum", and the fillers are known as "interregnum". Filled pauses are also called "Interjections".

Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 124–129,
Seattle, Washington, USA, 18-21 October 2013.

2 Related Work

Disfluency detection approaches can be divided into two different groups: text-first and speech first (Nakatani and Hirschberg, 1993). In the first approach, all prosodic and acoustic cues are ignored while in the second approach both grammatical and acoustic features are considered. For this paper, we focus on developing a text-first approach but our model is easily flexible with speech-first features because there is no restriction on the number and types of features in our model.

Among text-first approaches, the work is split between developing systems which focus specifically on disfluency detection and those which couple disfluency detection with parsing. For the former, Charniak and Johnson (2001) employ a linear classifier to predict the edited phrases in Switchboard corpus (Godfrey et al., 1992). Johnson and Charniak (2004) use a TAG-based noisy channel model to detect disfluencies while parsing with getting n-best parses from each sentence and re-ranking with a language model. The original TAG parser is not used for parsing itself and it is used just to find rough copies in the sentence. Their method achieves promising results on detecting edited words but at the expense of speed (the parser has a complexity of O(N^5). Kahn et al. (2005) use the same TAG model and add semi-automatically extracted prosodic features. Zwarts and Johnson (2011) improve the performance of TAG model by adding external language modeling information from data sets such as Gigaword in addition to using minimal expected F-loss in n-best re-ranking.

Georgila (2009) uses integer linear programming combined with CRF for learning disfluencies. That work shows that ILP can learn local and global constraints to improve the performance significantly. Qian and Liu (2013) achieve the best performance on the Switchboard corpus (Godfrey et al., 1992) without any additional data. They use three steps for detecting disfluencies using weighted Max-Margin Markov (M^3) network: detecting fillers, detecting edited words, and refining errors in previous steps.

Some text-first approaches treat parsing and disfluency detection jointly, though the models differ in the type of parse formalism employed. Lease and Johnson (2006) use a PCFG-based parser to parse sentences along with finding edited phrases. Miller and Schuler (2008) use a right-corner transform of binary branching structures on bracketed sentences but their results are much worse than (Johnson and Charniak, 2004). To date, none of the prior joint approaches have used a dependency formalism.

3 Joint Parsing Model

We model the problem using a deterministic transition-based parser (Nivre, 2008). These parsers have the advantage of being very accurate while being able to parse a sentence in linear time. An additional advantage is that they can use as many non-local and local features as needed.

Arc-Eager Algorithm We use the arc-eager algorithm (Nivre, 2004) which is a bottom-up parsing strategy that is used in greedy and k-beam transition-based parsers. One advantage of this strategy is that the words can get a head from their left side, before getting right dependents. This is particularly beneficial for our task, since we know that reparanda are similar to their repairs. Hence, a reparandum may get its head but whenever the parser faces a repair, it removes the reparandum from the sentence and continues its actions.

The actions in an arc-eager parsing algorithm are:

- **Left-arc (LA)**: The first word in the buffer becomes the head of the top word in the stack. The top word is popped after this action.
- **Right-arc (RA)**: The top word in the stack becomes the head of the first word in the buffer.
- **Reduce (R)**: The top word in the stack is popped.
- **Shift (SH)**: The first word in the buffer goes to the top of the stack.

Joint Parsing and Disfluency Detection We first extend the arc-eager algorithm by augmenting the action space with three new actions:

- **Reparandum (Rp[i;j])**: treat a phrase (words *i* to *j*) outside the look-ahead buffer as a reparandum. Remove them from the sentence and clear their dependencies.
- **Discourse Marker (Prn[i])**: treat a phrase in the look-ahead buffer (first *i* words) as a discourse marker and remove them from the sentence.

Stack	Buffer	Act.
flight	to Boston uh I mean ...	RA
flight to	Boston uh I mean to ...	RA
flight to Boston	**uh** I mean to Denver	Intj[1]
flight to Boston	**I mean** to Denver	Prn[1]
flight **to Boston**	to Denver	RP[2:3]
flight	to Denver	RA
flight to	Denver	RA
flight to Denver		R
flight to		R
flight		R

Figure 1: A sample transition sequence for the sentence "flight to Boston uh I mean to Denver". In the third column, only the underlined parse actions are learned by the parser (second classifier). The first classifier uses all instances for training (learns fluent words with "regular" label).

- **Interjection (Intj[i])**: treat a phrase in the look-ahead buffer (first *i* words) as a filled pause and remove them from the sentence.[2]

Our model has two classifiers. The first classifier decides between four possible actions and possible candidates in the current configuration of the sentence. These actions are the three new ones from above and a new action **Regular (Reg)**: which means do one of the original arc-eager parser actions.

At each configuration, there might be several candidates for being a *prn*, *intj* or *reparandum*, and one *regular* candidate. The candidates for being a *reparandum* are a set of words outside the look-ahead buffer and the candidates for being an *intj* or *prn* are a set of words beginning from the head of the look-ahead buffer. If the parser decides *regular* as the correct action, the second classifier predicts the best parsing transition, based on arc-eager parsing (Nivre, 2004).

For example, in the 4th state in Figure 1, there are multiple candidates for the first classifier: regular, "I" as *prn*[1] or *intj*[1], "I mean" as *prn*[2] or *intj*[2], "I mean to" as *prn*[3] or *intj*[3], "I mean to Denver" as *prn*[4] or *intj*[4], "Boston" as *rp*[3:3], "to Boston" as *rp*[2:3], and "flight to Boston" as *rp*[1:3].

[2]In the bracketed version of Switchboard corpus, reparandum is tagged with *EDITED* and discourse markers and paused fillers are tagged as *PRN* and *INTJ* respectively.

Training A transition-based parser action (our second-level classifier) is sensitive to the words in the buffer and stack. The problem is that we do not have gold dependencies for edited words in our data. Therefore, we need a parser to remove reparandum words from the buffer and push them into the stack. Since our parser cannot be trained on disfluent sentences from scratch, the first step is to train it on clean treebank data.

In the second step, we adapt parser weights by training it on disfluent sentences. Our assumption is that we do not know the correct dependencies between disfluent words and other words in the sentence. At each configuration, the parser updates itself with new instances by traversing all configurations in the sentences. In this case, if at the head of the buffer there is an *intj* or *prn* tag, the parser allows them to be removed from the buffer. If a reparandum word is not completely outside the buffer (the first two states in Figure 1), the parser decides between the four regular arc-eager actions (i.e. *left-arc*, *right-arc*, *shift*, and *reduce*). If the last word pushed into the stack is a reparandum and the first word in the buffer is a regular word, the parser removes all reparanda at the same level (in the case of nested edited words), removes their dependencies to other words and push their dependents into the stack. Otherwise, the parser performs the oracle action and adds that action as its new instance.[3]

With an adapted parser which is our second-level classifier, we can train our first-level classifier. The same procedure repeats, except that instances for disfluency detection are used for updating parameter weights for the first classifier for deciding the actions. In Figure 1, only the oracle actions (underlined) are added to the instances for updating parser weights but all first-level actions are learned by the first level classifier.

4 Experiments and Evaluation

For our experiments, we use the Switchboard corpus (Godfrey et al., 1992) with the same train/dev/test split as Johnson and Charniak (2004). As in that

[3]The reason that we use a parser instead of expanding all possible transitions for an edited word is that, the number of *regular* actions will increase and the other actions become sparser than natural.

work, incomplete words and punctuations are removed from data (except that we do not remove incomplete words that are not disfluent[4]) and all words are turned into lower-case. The main difference with previous work is that we use Switchboard *mrg* files for training and testing our model (since they contain parse trees) instead of the more commonly used Swithboard *dps* text files. *Mrg* files are a subset of *dps* files with about more than half of their size. Unfortunately, the disfluencies marked in the *dps* files are not exactly the same as those marked in the corresponding *mrg* files. Hence, our result is not completely comparable to previous work except for (Kahn et al., 2005; Lease and Johnson, 2006; Miller and Schuler, 2008).

We use Tsurgeon (Levy and Andrew, 2006) for extracting sentences from *mrg* files and use the Penn2Malt tool[5] to convert them to dependencies. Afterwards, we provide dependency trees with disfluent words being the dependent of nothing.

Learning For the first classifier, we use averaged structured Perceptron (AP) (Collins, 2002) with a minor modification. Since the first classifier data is heavily biased towards the "regular label", we modify the weight updates in the original algorithm to 2 (original is 1) for the cases where a "reparandum" is wrongly recognized as another label. We call the modified version "weighted averaged Perceptron (WAP)". We see that this simple modification improves the model accuracy.[6] For the second classifier (parser), we use the original averaged structured Perceptron algorithm. We report results on both AP and WAP versions of the parser.

Features Since for every state in the parser configuration, there are many candidates for being disfluent; we use local features as well as global features for the first classifier. Global features are mostly useful for discriminating between the four actions and local features are mostly useful for choosing a phrase as a candidate for being a disfluent phrase. The features are described in Figure 2. For the second classifier, we use the same features as (Zhang and Nivre, 2011, Table 1) except that we train our parser in a similar manner as the MaltParser (Nivre et al., 2007) without k-beam training.

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

Figure 2: Features used for learning the first classifier. Refined n-gram is the n-gram without considering words that are recognized as disfluent. Fine-grained (FG) transitions are enriched with parse actions (e.g. "regular:left-arc").

Parser Evaluation We evaluate our parser with both unlabeled attachment accuracy of correct words and precision and recall of finding the dependencies of correct words.[7] The second classifier is trained with 3 iterations in the first step and 3 iterations in the second step. We use the attachment accuracy of the parse tree of the correct sentences (without disfluencies) as the upper-bound attachment score and parsed tree of the disfluent sentences (without disfluency detection) as our lower-bound attachment score. As we can see in Table 1, WAP does a slightly better job parsing sentences. The upper-bound parsing accuracy shows that we do not lose too much information while jointly detecting disfluencies. Our parser is not comparable to (Johnson and Charniak, 2004) and (Miller and Schuler, 2008), since we use dependency relations for evaluation instead of constituencies.

Disfluency Detection Evaluation We evaluate our model on detecting edited words in the sentences

[4] E.g. I want t- go to school.

[5] `http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html`

[6] This is similar to WM^{3}N in (Qian and Liu, 2013).

[7] The parser is actually trained to do labeled attachment and labeled accuracy is about 1-1.5% lower than UAS.

	UAS	LB	UB	Pr.	Rec.	F2
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 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.

	Pr.	Rec.	F1
AP	**92.9**	71.6	80.9
WAP	85.1	**77.9**	**81.4**
KL (2005)	–	–	78.2
LJ (2006)	–	–	62.4
MS (2008)	–	–	30.6
QL (2013) – Default	–	–	81.7
QL (2013) – Optimized	–	–	**82.1**

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).

(words with "EDITED" tag in *mrg* files). 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[8] on the portion of *dps* files that have corresponding *mrg* files. 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.

[8]We use the second version of the code: `http://code.google.com/p/disfluency-detection/`. Results from the first version are 81.4 and 82.1 for the default and optimized settings.

5 Conclusion

In this paper, we have developed a fast, yet accurate, joint dependency parsing and disfluency detection model. Such a parser is useful for spoken dialogue systems which typically encounter disfluent speech and require accurate syntactic structures. The model is completely flexible with adding other features (either text or speech features).

There are still many ways of improving this framework such as using k-beam training and decoding, using prosodic and acoustic features, using out of domain data for improving the language and parsing models, and merging the two classifiers into one through better feature engineering. It is worth noting that we put the dummy *root* word in the first position of the sentence. Ballesteros and Nivre (2013) show that parser accuracy can improve by changing that position for English.

One of the main challenges in this problem is that most of the training instances are not disfluent and thus the sample space is very sparse. As seen in the experiments, we can get further improvements by modifying the weight updates in the Perceptron learner. In future work, we will explore different learning algorithms which can help us address the sparsity problem and improve the model accuracy. Another challenge is related to the parser speed, since the number of candidates and features are much greater than the number used in classical dependency parsers.

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.

References

Miguel Ballesteros and Joakim Nivre. 2013. Going to the roots of dependency parsing. *Computational Linguistics*, 39(1):5–13.

Heather Bortfeld, Silvia D. Leon, Jonathan E. Bloom, Michael F. Schober, and Susan E. Brennan. 2001. Disfluency rates in conversation: Effects of age, relationship, topic, role, and gender. *Language and Speech*, 44(2):123–147.

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *NAACL-HLT*, pages 1–9.

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

Ian R. Finlayson and Martin Corley. 2012. Disfluency in dialogue: an intentional signal from the speaker? *Psychonomic bulletin & review*, 19(5):921–928.

Kallirroi Georgila. 2009. Using integer linear programming for detecting speech disfluencies. In *NAACL-HLT*, pages 109–112.

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

Mark Johnson and Eugene Charniak. 2004. A tag-based noisy channel model of speech repairs. In *ACL*, pages 33–39.

Jeremy G. Kahn, Matthew Lease, Eugene Charniak, Mark Johnson, and Mari Ostendorf. 2005. Effective use of prosody in parsing conversational speech. In *EMNLP*, pages 233–240.

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

Roger Levy and Galen Andrew. 2006. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *LREC*, pages 2231–2234.

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

Christine Nakatani and Julia Hirschberg. 1993. A speech-first model for repair detection and correction. In *ACL*, pages 46–53.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Xian Qian and Yang Liu. 2013. Disfluency detection using multi-step stacked learning. In *NAACL-HLT*, pages 820–825.

Wen Wang, Andreas Stolcke, Jiahong Yuan, and Mark Liberman. 2013. A cross-language study on automatic speech disfluency detection. In *NAACL-HLT*, pages 703–708.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *ACL (Short Papers)*, pages 188–193.

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