# Non-Monotonic Parsing of *Fluent umm I Mean* Disfluent Sentences

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

**Joel Tetreault**
Yahoo Labs
New York, NY, USA
tetreaul@yahoo-inc.com

## Abstract

Parsing disfluent sentences is a challenging task which involves detecting disfluencies as well as identifying the syntactic structure of the sentence. While there have been several studies recently into solely detecting disfluencies at a high performance level, there has been relatively little work into joint parsing and disfluency detection that has reached that state-of-the-art performance in disfluency detection. We improve upon recent work in this joint task through the use of novel features and learning cascades to produce a model which performs at 82.6 F-score. It outperforms the previous best in disfluency detection on two different evaluations.

## 1 Introduction

Disfluencies in speech occur for several reasons: hesitations, unintentional mistakes or problems in recalling a new object (Arnold et al., 2003; Merlo and Mansur, 2004). Disfluencies are often decomposed into three types: filled pauses (IJ) such as "uh" or "huh", discourse markers (DM) such as "you know" and "I mean" and edited words (reparandum) which are repeated or corrected by the speaker (repair). The following sentence illustrates the three types:

I want a flight $\underbrace{\text{to Boston}}_{\text{Reparandum}}$ $\underbrace{\text{uh}}_{\text{IJ}}$ $\underbrace{\text{I mean}}_{\text{DM}}$ $\underbrace{\text{to Denver}}_{\text{Repair}}$

To date, there have been many studies on disfluency detection (Hough and Purver, 2013; Rasooli and Tetreault, 2013; Qian and Liu, 2013; Wang et al., 2013) such as those based on TAGs and the noisy channel model (e.g. Johnson and Charniak (2004), Zhang et al. (2006), Georgila (2009), and Zwarts and Johnson (2011)). High performance disfluency detection methods can greatly enhance the linguistic processing pipeline of a spoken dialogue system by first "cleaning" the speaker's utterance, making it easier for a parser to process correctly. A joint parsing and disfluency detection model can also speed up processing by merging the disfluency and parsing steps into one. However, joint parsing and disfluency detection models, such as Lease and Johnson (2006), based on these approaches have only achieved moderate performance in the disfluency detection task. Our aim in this paper is to show that a high performance joint approach is viable.

We build on our previous work (Rasooli and Tetreault, 2013) (henceforth RT13) to jointly detect disfluencies while producing dependency parses. While this model produces parses at a very high accuracy, it does not perform as well as the state-of-the-art in disfluency detection (Qian and Liu, 2013) (henceforth QL13). In this paper, we extend RT13 in two important ways: 1) we show that by adding a set of novel features selected specifically for disfluency detection we can outperform the current state of the art in disfluency detection in two evaluations[1] and 2) we show that by extending the architecture from two to six classifiers, we can drastically increase the speed and reduce the memory usage of the model without a loss in performance.

## 2 Non-monotonic Disfluency Parsing

In transition-based dependency parsing, a syntactic tree is constructed by a set of stack and buffer actions where the parser greedily selects an action at each step until it reaches the end of the sentence with an empty buffer and stack (Nivre, 2008). A state in a transition-based system has a stack of words, a buffer of unprocessed words and a set of arcs that have been produced in the parser history. The parser consists of a state (or a configuration)

[1] Honnibal and Johnson (2014) have a forthcoming paper based on a similar idea but with a higher performance.



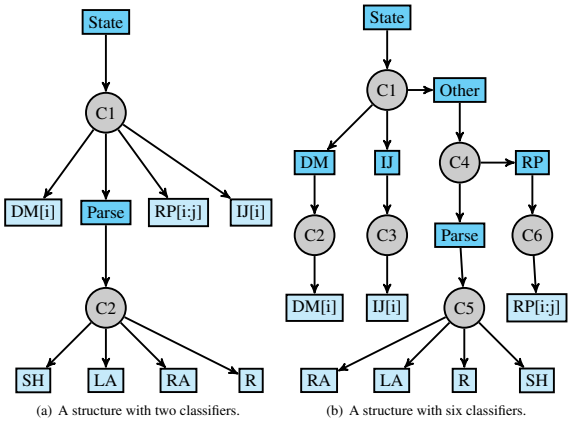
*Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 48–53,
Gothenburg, Sweden, April 26-30 2014. ©2014 Association for Computational Linguistics

which is manipulated by a set of actions. When an action is made, the parser goes to a new state.

The arc-eager algorithm (Nivre, 2004) is a transition-based algorithm for dependency parsing. In the initial state of the algorithm, the buffer contains all words in the order in which they appear in the sentence and the stack contains the artificial *root* token. The actions in arc-eager parsing are left-arc (LA), right-arc (RA), reduce (R) and shift (SH). LA removes the top word in the stack by making it the dependent of the first word in the buffer; RA shifts the first word in the buffer to the stack by making it the dependent of the top stack word; R pops the top stack word and SH pushes the first buffer word into the stack.

The arc-eager algorithm is a monotonic parsing algorithm, i.e. *once an action is performed, subsequent actions should be consistent with it* (Honnibal et al., 2013). In monotonic parsing, if a word becomes a dependent of another word or acquires a dependent, other actions shall not change those dependencies that have been constructed for that word in the action history. Disfluency removal is an issue for monotonic parsing in that if an action creates a dependency relation, the other actions cannot repair that dependency relation. The main idea proposed by RT13 is to change the original arc-eager algorithm to a non-monotonic one so it is possible to repair a dependency tree while detecting disfluencies by incorporating three new actions (one for each disfluency type) into a two-tiered classification process. The structure is shown in Figure 1(a). In short, at each state the parser first decides between the three new actions and a parse action (C1). If the latter is selected, another classifier (C2) is used to select the best parse action as in normal arc eager parsing.

The three additional actions to the arc-eager algorithm to facilitate disfluency detection are as follows: **1) RP[i:j]**: From the words outside the buffer, remove words $i$ to $j$ from the sentence and tag them as *reparandum*, delete all of their dependencies and push all of their dependents onto the stack. **2) IJ[i]**: Remove the first $i$ words from the buffer (without adding any dependencies to them) and tag them as *interjection*. **3) DM[i]**: Remove the first $i$ words from the buffer (without adding any dependencies) and tag them as *discourse marker*.



(a) A structure with two classifiers. (b) A structure with six classifiers.

Figure 1: Two kinds of cascades for disfluency learning. Circles are classifiers and light-colored blocks show the final decision by the system.

## 3 Model Improvements

To improve upon RT13, we first tried to learn all actions jointly. Essentially, we added the three new actions to the original arc-eager action set. However, this method (henceforth M1) performed poorly on the disfluency detection task. We believe this stems from a feature mismatch, i.e. some of the features, such as rough copies, are only useful for reparanda while some others are useful for other actions. Speed is an additional issue. Since for each state, there are many candidates for each of the actions, the space of possible candidates makes the parsing time potentially squared.

**Learning Cascades** One possible solution for reducing the complexity of the inference is to formulate and develop learning cascades where each cascade is in charge of a subset of predictions with its specific features. For this task, it is not essential to always search for all possible phrases because only a minority of cases in speech texts are disfluent (Bortfeld et al., 2001). For addressing this problem, we propose M6, a new structure for learning cascades, shown in Figure 1(b) with a more complex structure while more efficient in terms of speed and memory. In the new structure, we do not always search for all possible phrases which will lead to an expected linear time complexity. The main processing overhead here is the number of decisions to make by classifiers but this is not as time-intensive as finding all candidate phrases in all states.

**Feature Templates** RT13 use different feature sets for the two classifiers: C2 uses the parse fea-

tures promoted in Zhang and Nivre (2011, Table 1) and C1 uses features which are shown with regular font in Figure 2. We show that one can improve RT13 by adding new features to the C1 classifier which are more appropriate for detecting reparanda (shown in bold in Figure 2). We call this new model M2E, "E" for extended. In Figure 3, the features for each classifier in RT13, M2E, M6 and M1 are described.

We introduce the following new features: **LIC** looks at the number of common words between the reparandum candidate and words in the buffer; e.g. if the candidate is "to Boston" and the words in the buffer are "to Denver", LIC[1] is one and LIC[2] is also one. In other words, LIC is an indicator of a rough copy. The **GPNG** (post n-gram feature) allows us to model the fluency of the resulting sentence after an action is performed, without explicitly going into it. It is the count of possible n-grams around the buffer after performing the action; e.g. if the candidate is a reparandum action, this feature introduces the n-grams which will appear after this action. For example, if the sentence is "I want a flight to Boston | to Denver" (where | is the buffer boundary) and the candidate is "to Boston" as reparandum, the sentence will look like "I want a flight | to Denver" and then we can count all possible n-grams (both lexicalized and unlexicalized) in the range i and j inside and outside the buffer. **GBPF** is a collection of baseline parse features from (Zhang and Nivre, 2011, Table 1).

The need for classifier specific features becomes more apparent in the M6 model. Each of the classifiers uses a different set of features to optimize performance. For example, LIC features are only useful for the sixth classifier while post n-gram features are useful for C2, C3 and C6. For the joint model we use the C1 features from M2B and the C1 features from M6.

| Abbr. | Description |
|---|---|
| GS[i/j] | First n Ws/POS outside $\beta$ (n=1:i/j) |
| GB[i/j] | First n Ws/POS inside $\beta$ (n=1:i/j) |
| GL[i/j] | Are n Ws/POS i/o $\beta$ equal? (n=1:i/j) |
| GT[i] | n last FGT; e.g. *parse:la* (n=1:i) |
| GTP[i] | n last FGT e.g. *parse* (n=1:i) |
| GGT[i] | n last FGT + POS of $\beta_0$ (n=1:i) |
| GGTP[i] | n last CGT + POS of $\beta_0$ (n=1:i) |
| GN[i] | (n+m)-gram of m/n POS i/o $\beta$ (n,m=1:i) |
| GIC[i] | # common Ws i/o $\beta$ (n=1:i) |
| GNR[i] | Rf. (n+m)-gram of m/n POS i/o $\beta$ (n,m=1:i) |
| **GPNG[i/j]** | PNGs from n/m Ws/POS i/o $\beta$ (m,n:1:i/j) |
| **GBPF** | Parse features (Zhang and Nivre, 2011) |
| LN[i,j] | First n Ws/POS of the cand. (n=1:i/j) |
| LD | Distance between the cand. and $s_0$ |
| LL[i,j] | first n Ws/POS of rp and $\beta$ equal? (n=1:i/j) |
| **LIC[i]** | # common Ws for rp/repair (n=1:i) |

Figure 2: Feature templates used in this paper and their abbreviations. $\beta$: buffer, $\beta_0$: first word in the buffer, $s_0$: top stack word, Ws: words, rp: reparadnum, cand.: candidate phrase, PNGs: post n-grams, FGT: fine-grained transitions and CGT: coarse-grained transitions. *Rf. n-gram*: n-gram from unremoved words in the state.

| Classifier | Features |
|---|---|
| **M2 Features** | |
| C1 (RT13) | GS[4/4], GB[4/4], GL[4/6], GT[5], GTP[5] GGT[5], GGTP[5], GN[4], GNR[4], GIC[6] LL[4/6], LD |
| C1 (M2E) | RT13 $\cup$ (LIC[6], GBPF, GPNG[4/4]) - LD |
| C2 | GBPF |
| **M6 Features** | |
| C1 | GBPF, GB[4/4], GL[4/6], GT[5], GTP[5] GGT[5], GGTP[5], GN[4], GNR[4], GIC[6] |
| C2 | GB[4/4], GT[5], GTP[5], GGT[5], GGTP[5] GN[4], GNR[4], GPNG[4/4], LD, LN[24/24] |
| C3 | GB[4/4], GT[5], GTP[5], GGT[5], GGTP[5] GN[4], GNR[4], GPNG[4/4], LD, LN[12/12] |
| C4 | GBPF, GS[4/6], GT[5], GTP[5], GGT[5] GGTP[5], GN[4], GNR[4], GIC[13] |
| C5 | GBPF |
| C6 | GBPF, LL[4/6], GPNG[4/4] LN[6/6], LD, LIC[13] |
| **M1 Features**: RT13 C1 features $\cup$ C2 features | |

Figure 3: Features for each model. M2E is the same as RT13 with extended features (bold features in Figure 2). M6 is the structure with six classifiers. Other abbreviations are described in Figure 2.

## 4 Experiments and Evaluation

We evaluate our new models, M2E and M6, against prior work on two different test conditions. In the first evaluation (Eval 1), we use the parsed section of the Switchboard corpus (Godfrey et al., 1992) with the train/dev/test splits from Johnson and Charniak (2004) (JC04). All experimental settings are the same as RT13. We compare our new models against this prior work in terms of disfluency detection performance and parsing accuracy. In the second evaluation (Eval 2), we compare our work against the current best disfluency detection method (QL13) on the JC04 split as well as on a 10 fold cross-validation of the parsed section of the Switchboard. We use gold POS tags for all evaluations.

For all of the joint parsing models we use the weighted averaged Perceptron which is the same as averaged Perceptron (Collins, 2002) but with a

loss weight of two for reparandum candidates as done in prior work. The standard arc-eager parser is first trained on a "cleaned" Switchboard corpus (i.e. after removing disfluent words) with 3 training iterations. Next, it is updated by training it on the real corpus with 3 additional iterations. For the other classifiers, we use the same number of iterations determined from the development set.

**Eval 1** The disfluency detection and parse results on the test set are shown in Table 1 for the four systems (M1, RT13, M2E and M6). The joint model performs poorly on the disfluency detection task, with an F-score of 41.5, and the prior work performance which serves as our baseline (RT13) has a performance of 81.4. The extended version of this model (M2E) raises performance substantially to 82.2. This shows the utility of training the C1 classifier with additional features. Finally, the M6 classifier is the top performing model at 82.6.

| | Disfluency | | | Parse | |
|---|---|---|---|---|---|
| Model | Pr. | Rec. | F1 | UAS | F1 |
| M1 | 27.4 | **85.8** | 41.5 | 60.2 | 64.6 |
| RT13 | 85.1 | 77.9 | 81.4 | 88.1 | 87.6 |
| M2E | **88.1** | 77.0 | 82.2 | 88.1 | 87.6 |
| M6 | 87.7 | 78.1 | **82.6** | **88.4** | **87.7** |

Table 1: Comparison of joint parsing and disfluency detection methods. UAS is the unlabeled parse accuracy score.

The upperbound for the parser attachment accuracy (UAS) is 90.2 which basically means that if we have gold standard disfluencies and remove disfluent words from the sentence and then parse the sentence with a regular parser, the UAS will be 90.2. If we had used the regular parser to parse the disfluent sentences, the UAS for correct words would be 70.7. As seen in Table 1, the best parser UAS is 88.4 (M6) which is very close to the upperbound, however RT13, M2E and M6 are nearly indistinguishable in terms of parser performance.

**Eval 2** To compare against QL13, we use the second version of the publicly provided code and modify it so it uses gold POS tags and retrain and optimize it for the parsed section of the Switchboard corpus (these are known as *mrg* files, and are a subset of the section of the Switchboard corpus used in QL13, known as *dps* files). Since their system has parameters tuned for the *dps* Switchboard corpus we retrained it for a fair comparison. As in the reimplementation of RT13, we have evaluated the QL13 system with optimal number of training iterations (10 iterations). As seen in Table 2, although the annotation in the *mrg* files is less precise than in the *dps* files, M6 outperforms all models on the JC04 split thus showing the power of the new features and new classifier structure.

| Model | JC04 split | xval |
|---|---|---|
| RT13 | 81.4 | 81.6 |
| QL13 (optimized) | 82.5 | 82.2 |
| M2E | 82.2 | **82.8** |
| M6 | **82.6** | 82.7 |

Table 2: Disfluency detection results (F1 score) on JC04 split and with cross-validation (xval)

To test for robustness of our model, we perform 10-fold cross validation after clustering files based on their name alphabetic order and creating 10 data splits. As seen in Table 2, the top model is actually M2E, nudging out M6 by 0.1. More noticeable is the difference in performance over QL13 which is now 0.6.

**Speed and memory usage** Based on our Java implementation on a 64-bit 3GHz Intel CPU with 68GB of memory, the speed for M6 (36 ms/sent) is 3.5 times faster than M2E (128 ms/sent) and 5.2 times faster than M1 (184 ms/sent) and it requires half of the nonzero features overall compared to M2E and one-ninth compared to M1.

## 5 Conclusion and Future Directions

In this paper, we build on our prior work by introducing rich and novel features to better handle the detection of reparandum and by introducing an improved classifier structure to decrease the uncertainty in decision-making and to improve parser speed and accuracy. We could use early updating (Collins and Roark, 2004) for learning the greedy parser which is shown to be useful in greedy parsing (Huang and Sagae, 2010). K-beam parsing is a way to improve the model though at the expense of speed. The main problem with k-beam parsers is that it is complicated to combine classifier scores from different classifiers. One possible solution is to modify the three actions to work on just one word per action, thus the system will run in completely linear time with one classifier and k-beam parsing can be done by choosing better features for the joint parser. A model similar to this idea is designed by Honnibal and Johnson (2014).

**Acknowledgement** We would like to thank the reviewers for their comments and useful insights. The bulk of this research was conducted while both authors were working at Nuance Communication, Inc.'s Laboratory for Natural Language Understanding in Sunnyvale, CA.

## References

Jennifer E. Arnold, Maria Fagnano, and Michael K. Tanenhaus. 2003. Disfluencies signal theee, um, new information. *Journal of Psycholinguistic Research*, 32(1):25–36.

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.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics.

Kallirroi Georgila. 2009. Using integer linear programming for detecting speech disfluencies. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 109–112, Boulder, Colorado. Association for Computational Linguistics.

John J. Godfrey, Edward C. Holliman, and Jane McDaniel. 1992. Switchboard: Telephone speech corpus for research and development. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-92)*, volume 1, pages 517–520.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disuency detection and dependency parsing. *Transactions of the Association for Computational Linguistics (TACL)*, to appear.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172, Sofia, Bulgaria. Association for Computational Linguistics.

Julian Hough and Matthew Purver. 2013. Modelling expectation in the self-repair processing of annotat-, um, listeners. In *The 17th Workshop on the Semantics and Pragmatics of Dialogue*.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden. Association for Computational Linguistics.

Mark Johnson and Eugene Charniak. 2004. A TAG-based noisy channel model of speech repairs. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 33–39, Barcelona, Spain.

Matthew Lease and Mark Johnson. 2006. Early deletion of fillers in processing conversational speech. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 73–76, New York City, USA. Association for Computational Linguistics.

Sandra Merlo and Letıcia Lessa Mansur. 2004. Descriptive discourse: topic familiarity and disfluencies. *Journal of Communication Disorders*, 37(6):489–503.

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

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 *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 820–825, Atlanta, Georgia. Association for Computational Linguistics.

Mohammad Sadegh Rasooli and Joel Tetreault. 2013. Joint parsing and disfluency detection in linear time. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 124–129, Seattle, Washington, USA. Association for Computational Linguistics.

Wen Wang, Andreas Stolcke, Jiahong Yuan, and Mark Liberman. 2013. A cross-language study on automatic speech disfluency detection. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 703–708, Atlanta, Georgia. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In

*Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics.

Qi Zhang, Fuliang Weng, and Zhe Feng. 2006. A progressive feature selection algorithm for ultra large feature spaces. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 561–568, Sydney, Australia. Association for Computational Linguistics.

Simon Zwarts and Mark Johnson. 2011. The impact of language models and loss functions on repair disfluency detection. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 703–711, Portland, Oregon, USA. Association for Computational Linguistics.